

# ClickHouse چیست؟

ClickHouse یک مدیریت دیتابیس (DBMS) ستون گرا برای پردازش تحلیلی آنلاین (OLAP) می باشد.

در یک مدیریت دیتابیس ردیف گرا، داده ها به فرم زیر ذخیره سازی می شوند:

EventTime	GoodEvent	Title	JavaEnable	WatchID	Row
05:19:20 2016-05-18	1	Investor Relations	1	5385521489354350662	#0
08:10:20 2016-05-18	1	Contact us	0	5385521490329509958	#1
07:38:00 2016-05-18	1	Mission	1	5385521489953706054	#2
...	...	...	...	...	N#

به این صورت، تمام مقادیر مربوط به یک سطر (رکورد) به صورت فیزیکی و در کنار یکدیگر ذخیره سازی می شوند.

دیتابیس های MySQL, Postgres و MS SQL Server از انواع دیتابیس های ردیف گرا می باشند.

در یک دیتابیس ستون گرا، داده ها به شکل زیر ذخیره سازی می شوند:

N#	#2	#1	#0	:Row
...	5385521489953706054	5385521490329509958	5385521489354350662	:WatchID
...	1	0	1	:JavaEnable
...	Mission	Contact us	Investor Relations	:Title
...	1	1	1	:GoodEvent
...	07:38:00 2016-05-18	08:10:20 2016-05-18	05:19:20 2016-05-18	:EventTime

این مثال ها تنها نشان می دهند که داده ها منظم شده اند.

مقادیر ستون های مختلف به صورت جدا، و داده های مربوط به یک ستون در کنار یکدیگر ذخیره می شوند.

مثال های از دیتابیس های ستون گرا: Vertica, Paracel (Actian Matrix, Amazon Redshift), Sybase IQ, Exasol, Infobright, InfiniDB, MonetDB (VectorWise, Actian Vector), LucidDB, SAP HANA, Google Dremel, Google PowerDrill, Druid, +kdb.

ترتیب های مختلف برای ذخیره سازی داده ها، مناسب سناریو های مختلف هستند. سناریو دسترسی به داده اشاره دارد به، چه query هایی ساخته شده اند، چند وقت به چند وقت، در چه مقداری، چقدر داده در هنگام اجرای هر query خوانده می شود، چند رکورد، چند ستون و چند بایت؛ رابطه ی بین خوانده و نوشتن داده؛ سایز دیتاسی فعال مورد استفاده و نحوه ی استفاده آن به صورت محلی؛ آیا از تراکنش استفاده می شود؛ چگونه داده ها جدا می شوند؛ نیازمندی ها برای replication داده ها و یکپارچگی منطقی داده ها؛ نیازمندی ها برای latency و throughput برای هر نوع از query، و...

مهمتر از بالا بودن لود سیستم، سفارشی کردن سیستم مطابق با نیازمندی های سناریو می باشد، و این سفارشی سازی در ادامه دقیق تر می شود. هیچ سیستمی وجود ندارد که مناسب انجام سناریو های متفاوت (بسیار متفاوت) باشد. اگر یک سیستم برای اجرای سناریو های مختلف آماده شده باشد، در زمان بالا بودن لود، سیستم تمام سناریوها را به صورت ضعیف handle می کند.

## ویژگی های کلیدی یک سناریو OLAP

- اکثریت درخواست های برای خواندن می باشد.
- داده ها به صورت batch های بزرگ (> 1000 رکورد) وارد می شوند، نه به صورت تکی؛ یا اینکه اصلا بروز نمی شوند.
- داده ها به دیتاسی، اضافه می شوند و تغیر پیدا نمی کنند.

- برای خواندن، تعداد زیادی از رکورد ها از دیتابیس استخراج می شوند، اما فقط چند ستون از رکورد ها.
- جداول "wide" هستند، به این معنی تعداد زیادی ستون دارند.
- query ها نسبتا کم هستند (معمولا صدها query در ثانیه به ازای هر سرور یا کمتر)
- برای query های ساده، زمان تاخیر 50 میلی ثانیه مجاز باشد.
- مقادیر ستون ها کوچک باشد: اعداد و رشته های کوتاه (برای مثال 60 بایت به ازای هر url)
- نیازمند throughput بالا در هنگام اجرای یک query (بالای یک میلیارد رکورد در هر ثانیه به ازای هر سرور)
- تراکنش واجب نیست.
- نیازمندی کم برای consistency بودن داده ها.
- فقط یک جدول بزرگ به ازای هر query وجود دارد. تمام جداول کوچک هستند، به جز یکی.
- نتیجه query به طول قابل توجهی کوچکتر از source داده ها می باشد. به عبارتی دیگر در یک query، داده ها فیلتر یا تجمیع می شوند، پس نتایج در RAM یک سرور فیت می شوند.

خوب خیلی ساده می توان دید که سناریو های OLAP خیلی متفاوت تر از دیگر سناریو های محبوب هستند (مثل OLTP یا Key-Value). پس اگر میخواهید performance مناسب داشته باشید، استفاده از دیتابیس های OLTP یا Key-Value برای اجرای query های OLAP معنی ندارد. برای مثال، اگر شما از دیتابیس MongoDB یا Redis برای آنالیز استفاده کنید، قطعا performance بسیار ضعیف تری نسبت به دیتابیس های OLAP خواهید داشت.

## دلایل برتری دیتابیس های ستون گرا برای سناریو های OLAP

دیتابیس های ستون گرا مناسب سناریو های OLAP هستند (حداقل 100 برابر در بیشتر query ها سرعت پردازش آنها بهتر است). دلایل این برتری در پایین شرح داده شده است، اما آسانترش این هست که به صورت visually این تفاوت را ببینیم:

### ردیف گرا



### ستون گرا



تفاوت را دیدید؟ بیشتر بخوانید تا یاد بگیرید چرا این اتفاق رخ میدهد.

## Input/output

1. برای query های تحلیلی، تنها چند ستون از تمام ستون های جدول نیاز به خواندن دارد. در یک دیتابیس ستون گرا، شما فقط داده ی مورد نیاز را می خوانید. برای مثال، اگر شما نیاز به 5 ستون از 100 ستون را دارید، شما می توانید انتظار 20 برابر کاهش I/O را داشته باشید.
2. از آنجایی که داده در بسته ها خوانده می شوند، فشردگی داده های ستون ها برای فشردگی سازی ساده می باشند. این باعث کاهش نرخ I/O در ادامه می شود.

3. با توجه به کاهش I/O، داده های بیشتری در system cache قرار می گیرند.

برای مثال، query "تعداد رکوردها به ازای هر بستر نیازمندی" نیازمند خواندن ستون "آیدی بستر آگهی"، که 1 بایت بدون فشرده طول می کشد، خواهد بود. اگر بیشتر ترافیک مربوط به بسترهای نیازمندی نبود، شما می توانید انتظار حداقل 10 برابر فشرده سازی این ستون را داشته باشید. زمانی که از الگوریتم فشرده سازی quick استفاده می کنید، عملیات decompression داده ها با سرعت حداقل چندین گیگابایت در ثانیه انجام می شود. به عبارت دیگر، این query توانایی پردازش تقریباً چندین میلیارد رکورد در ثانیه به ازای یک سرور را دارد. این سرعت در عمل واقعی و دست یافتنی است.

▼ مثال

```
clickhouse-client $
ClickHouse client version 0.0.52053
Connecting to localhost:9000
Connected to ClickHouse server version 0.0.52053

SELECT CounterID, count() FROM hits GROUP BY CounterID ORDER BY count() DESC LIMIT 20 (:

SELECT
    CounterID
    ,count()
FROM hits
GROUP BY CounterID
ORDER BY count() DESC
LIMIT 20

+----+-----+-----+
| CounterID | count() |
+----+-----+-----+
| 56057344  | 114208  |
| 51619590  | 115080  |
| 44658301  | 3228    |
| 42045932  | 38230   |
| 42042158  | 145263  |
| 38297270  | 91244   |
| 26647572  | 154139  |
| 24112755  | 150748  |
| 21302571  | 242232  |
| 13507087  | 338158  |
| 12229491  | 62180   |
| 12187441  | 82264   |
| 12148031  | 232261  |
| 11438516  | 146272  |
| 11403636  | 168777  |
| 11227824  | 4120072 |
| 10519739  | 10938808 |
| 9047015   | 74088   |
| 8837972   | 115079  |
| 8205961   | 337234  |
+----+-----+-----+

(rows in set. Elapsed: 0.153 sec. Processed 1.00 billion rows, 4.00 GB (6.53 billion rows/s., 26.10 GB/s 20

(:
```

## CPU

از آنجایی که اجرای یک query نیازمند پردازش تعداد زیادی سطر می باشد، این کمک می کند تا تمام عملیات ها به جای ارسال به سطرهای جداگانه، برای کل بردار ارسال شود، یا برای ترکیب query engine به طوری که هیچ هزینه ی ارسالی وجود ندارد. اگر این کار رو نکنید، با هر half-decent disk subsystem، تفسیرگر query ناگزیر است که CPU را متوقف کند. این منطقی است که در صورت امکان هر دو کار ذخیره سازی داده در ستون ها و پردازش ستون ها با هم انجام شود.

دو راه برای انجام این کار وجود دارد:

1. یک موتور بردار. تمام عملیات ها به جای مقادیر جداگانه، برای بردارها نوشته شوند. این به این معنیست که شما خیلی از مواقع نیازی

به صدا کردن عملیات ها ندارید، و هزینه انتقال ناچیز است. کد عملیاتی شامل یک چرخه داخلی بهینه شده است.

2. Code generation. کد تولید شده برای query دارای تمام تماس های غیرمستقیم در آن است.

این در یک دیتابیس نرمال انجام نمی شود، چرا که برای اجرای query های ساده این کارها منطقی نیست. هرچند، استثناهای هم وجود دارد. برای مثال، MemSQL از code generation برای کاهش latency در هنگام پردازش query های SQL استفاده می کند. (برای مقایسه، مدیریت دیتابیس های آنالیزی نیازمند بهینه سازی توان عملیاتی (throughput) هستند نه latency).

توجه کنید که برای کارایی query language، CPU باید SQL یا MDX باشد، یا حداقل یک بردارد (J, K) باشد. query برای بهینه سازی باید فقط دارای حلقه های implicit باشد.

## ویژگی های برجسته ClickHouse

### مدیریت دیتابیس ستون گرای واقعی

در یک مدیریت دیتابیس ستون گرای واقعی، هیچ مقداری فضای اضافی برای ذخیره سازی ندارد. برای مثال، این به این معنیست که برای مقادیر، constant-length باید پشتیبانی شوند تا از ذخیره سازی طول مقدار به عنوان یک عدد integer کنار مقدار جلوگیری شود. در این مورد، یک میلیارد مقدار Uint8 باید در واقع در حالت غیرفشرده 1 گیگابایت فضا اشغال کند، در غیراین صورت به شدت بر عملکرد CPU تاثیر میگذارد. این خیلی مهم هست که داده ها به صورت compact ذخیره سازی شوند حتی زمانی که uncompressed هستند، از آنجا که سرعت (سرعت CPU Usage) (decompress) عمدتا به حجم داده های uncompress بستگی دارد.

این بسیار قابل توجه است چون سیستم هایی وجود دارند که توانایی ذخیره سازی مقادیر ستون ها را به صورت جداگانه دارند، اما به دلیل بهینه سازی آنها برای دیگر سناریو ها، نمیتوانند به طور موثر پردازش های تحلیلی انجام دهند. برای مثال HBase، BigTable، Cassandra و HyperTable. در این سیستم ها، شما توان عملیاتی حدود صدها هزار سطر در ثانیه را دارید، اما نه صدها میلیون سطر در ثانیه.

همچنین توجه داشته باشید که ClickHouse یک مدیریت دیتابیس است نه فقط یک دیتابیس. ClickHouse اجازه میدهد که در زمان اجرا اقدام به ساخت جدول، دیتابیس کنید، داده load کنید و query های خود را بدون restart و یا reconfigure مجدد سرور، اجرا کنید

### فشرده سازی داده ها

بعضی دیتابیس های ستون گرا (InfiniDB CE یا MonetDB) از فشرده سازی داده ها استفاده نمی کنند. با این حال، فشرده سازی داده ها برای رسیدن به عملکرد عالی ضروری است.

### Disk storage of data

خیلی از مدیریت دیتابیس های ستون گرا (مثل SAP HANA و Google PowerDrill) فقط داخل RAM کار می کنند. این رویکرد منجر به تخصیص بودجه سخت افزاری بالا برای تحقق آنالیز های real-time می شود. ClickHouse برای کار بر روی هارد دیسک های معمولی طراحی شده است، که هزینه ی برای هر گیگابایت داده را تضمین می کند، اما اگر SSD و RAM موجود باشد به طور کامل مورد استفاده قرار می گیرد.

### پردازش موازی روی چندین هسته

query های بزرگ به طور طبیعی با استفاده از تمام منابع موجود در روی سرور فعلی، موازی سازی می شوند.

### پردازش توزیع شده بر روی چندین سرور

تقریبا هیچ کدام از DBMS هایی که بالاتر ذکر شد، از اجرای query ها به صورت توزیع شده پشتیبانی نمی کنند. در ClickHouse، داده ها می توانند در shard های مختلف مستقر شوند. هر shard میتواند گروهی از replica ها برای بالا بردن تحمل پذیری در برابر خطا (fault tolerance) را داشته باشد. یک query به صورت موازی بر روی تمامی shard های اجرا می شود. این برای کاربر شفاف است.

### پشتیبانی SQL

اگر شما با SQL آشنا باشید، ما واقعا نمیتوانیم در مورد پشتیبانی از SQL صحبت کنیم. تمام توابع اسم های مختلفی دارند. با این حال، این یک زبان بر پایه SQL هست که میتواند در بسیاری از موارد با SQL متفاوت باشد. JOIN ها پشتیبانی می شود. subquery ها در FROM، IN و JOIN پشتیبانی می شود. Dependent subquery ها پشتیبانی نمی شود.

ClickHouse زبان بر پایه SQL است که در بسیاری از موارد از استاندارد SQL پیروی می کند. GROUP BY، ORDER BY، scalar subquery ها در FROM، IN و JOIN پشتیبانی می شود. subquery های مرتبط و window function ها پشتیبانی نمی شود.

## موتور بردارد

داده ها نه فقط براساس ستون ها ذخیره می شوند، بلکه با استفاده از برادرها (بخشی از ستون ها) پردازش می شوند. این قابلیت باعث رسیدن به بهره وری بالای CPU می شود.

## بروزرسانی داده ها به صورت Real-Time

ClickHouse از جداول دارای Primary Key پشتیبانی می کند. به منظور اجرای query های range بر روی Primary Key، داده ها به صورت افزایشی (مرتب شده) با استفاده از merge tree ذخیره سازی می شوند. با توجه به این، داده ها می توانند به طور پیوسته به جدول اضافه شوند. هیچ نوع lock در هنگام مصرف داده ها وجود ندارد.

## Index

داشتن داده ها به صورت فیزیکی و مرتب شده براساس Primary Key این قابلیت را می دهد که استخراج کردن داده برای مقدار خاص و یا مقادیر range با کمترین latency، یعنی کمتر از چند هزار میلی ثانیه ممکن شود.

## مناسب برای query های آنلاین

latency پایین به این معنی است که query ها بدون delay و بدون تلاش برای رسیدن به پیش پاسخ (از قبل محاسبه شده) دقیقا در همان لحظه که کاربر در حال load صفحه است پردازش شوند. به عبارتی دیگر، آنلاین

## پشتیبانی از محاسبات تقریبی

ClickHouse روش های مختلفی برای کسب دقیق performance ارائه می دهد:

1. توابع Aggregate برای محاسبات تقریبی تعداد مقادیر متمایز (median)، (distinct و quantity ها
2. اجرای یک query بر پایه بخشی از داده ها (داده ی sample) و دریافت خروجی تقریبی. در این مورد داده ی نسبتا کمتری از دیسک بازیابی می شود.
3. اجرای یک Aggregation برای تعداد محدودی از کلید های تصافی، به جای تمام کلید ها. در شرایط خاص برای توزیع کلید در داده ها، این روش کمک می کند به نتایج منطقی برسیم با استفاده از منابع کمتر.

## Replication داده ها و integrity

ClickHouse از روش asynchronous multimaster replication استفاده می کند. بعد از نوشتن داده در یکی از replica های موجود، داده به صورت توزیع شده به بقیه replica ها منتقل می شود. این سیستم داده های مشابه را در replica های مختلف نگه داری می کند. در اکثر موارد که سیستم fail می شوند، داده ها به صورت اتوماتیک restore می شوند و یا در موارد پیچیده به صورت نیمه اتوماتیک restore می شوند.

برای اطلاعات بیشتر، به بخش **replication داده ها** مراجعه کنید.

## ویژگی های از ClickHouse که می تواند معایبی باشد.

1. بدون پشتیبانی کامل از تراکنش
2. عدم توانایی برای تغییر و یا حذف داده های در حال حاضر وارد شده با سرعت بالا و تاخیر کم. برای پاک کردن و یا اصلاح داده ها، به عنوان مثال برای پیروی از **GDPR**، دسته ای پاک و به روزرسانی وجود دارد. حال توسعه می باشد.
3. Sparse index باعث می شود ClickHouse چندان مناسب اجرای پرسمان های point query برای دریافت یک ردیف از داده ها با استفاده از کلید آنها نباشد.

## Performance

با توجه به نتایج تست های Yandex، ClickHouse بهترین عملکرد را برای سناریوهای عملیاتی قابل مقایسه با دیگر سیستم های در کلاس خود را از خود نشان داد. این تست ها شامل بالاترین توان عملیاتی برای query های طولانی، و کمترین latency برای query های کوتاه بود. نتایج این تست های در **صفحه ی جدا** موجود است.

benchmark های زیادی وجود دارند که این نتایج را تایید می کنند. میتوانید این نتایج را جستجو کنید و یا **این لینک های benchmark** مستقل را ببینید.

## توان عملیاتی برای یک query بزرگ

توان عملیاتی می تواند به صورت تعداد سطر در ثانیه و یا تعداد مگابایت در ثانیه اندازه گیری شود. اگر داده ها در page cache قرار داشته باشند، یک query برای اجرا شدن بر روی سخت افزارهای مدرن چندان پیچیده نخواهد بود و با سرعت تقریباً 2 تا 10 گیگابایت در ثانیه برای داده های غیرفشرده و در یک سرور پردازش خواهد شد (برای ساده ترین موارد، سرعت ممکن است به 30 گیگابایت در ثانیه برسد). اگر داده ها در page cache قرار نداشته باشند، سرعت محدود به دیسک و همچنین چگونگی فشرده سازی داده ها بر روی دیسک می باشد. برای مثال اگر یک دیسک اجازه ی خواندن داده ها با سرعت 400 مگابایت در ثانیه را بدهد، و داده ها با نرخ 3 فشرده سازی شده باشند، سرعت در حدود 1.2 گیگابایت در ثانیه خواهد بود. برای گرفتن تعداد رکورد در ثانیه، سرعت بایت در ثانیه را تقسیم بر کل سایز ستون ها مورد استفاده در query می کنیم. برای مثال اگر 10 بایت از ستوه ها استخراج می شود، سرعت در حدود 100 تا 200 میلیون سطر در ثانیه می باشد.

سرعت پردازش در سرویس توزیع شده تقریباً به صورت خطی افزایش پیدا می کند، اما فقط وقتی که نتایج سطرهای به دست آمده از aggragation یا مرتب سازی زیاد بزرگ نباشند.

## Latency در زمان پردازش query های کوتاه

اگر یک query از Primary Key استفاده کند و تعداد زیادی از سطر ها را برای پردازش select نکند (صدها هزار)، و از تعداد زیادی ستون استفاده نکند، اگر داده ها در page cache قرار داشته باشند، ما میتوانیم انتظار latency کمتر از 50 میلی ثانیه را داشته باشیم. در غیر این صورت محاسبه زمان براساس تعداد seek ها انجام خواهد گرفت. اگر شما از هارد های دیسکی استفاده می کنید، برای سیستمی که overload ندارد، محاسبه تقریبی latency با استفاده از این فرمول ممکن است: زمان (10 ms seek) \* تعداد ستون های مورد نیاز در query \* تعداد قطعات داده

## توان عملیاتی در هنگام پردازش تعداد زیادی از query های کوتاه

تحت شرایط مشابه، ClickHouse توانایی رسیدگی به چند صد query در ثانیه به ازای یک سرور را دارد (بالای چند هزار در ثانیه در بهترین مورد). از آنجایی که این سناریو در مدیریت دیتابیس های آنالیزی معمول نیست، بهتر است نهایتاً انتظار چند صد query در ثانیه را داشته باشید.

## Performance در هنگام درج داده ها

پیشنهاد می کنیم درج داده ها را به صورت دسته ای و حداقل 100 سطر در هر دسته انجام دهید و یا بیش از یک درخواست insert در ثانیه را نداشته باشید. در هنگام درج داده در جدول MergeTree از یک dump جدا شده با tab، سرعت درج داده از 50 تا 200 مگابایت در ثانیه می باشد. اگر سطر های درج شده حدود 1 کیلوبایت باشند، سرعت حدود 50 هزار تا 200 هزار سطر در ثانیه می باشد. اگر سطر ها کوچک باشند بازدهی بالایی در تعداد سطر در ثانیه خواهیم داشت. در 500 -> Banner System Data هزار سطر در ثانیه، در 1 -> Graphite data (میلیون سطر در ثانیه). برای بهبود کارایی، شما می توانید چندین insert را به صورت موازی اجرا کنید، که در این حالت کارایی سیستم به صورت خطی افزایش می یابد.

## Yandex.Metrica use case

ClickHouse در ابتدا برای قدرت به Yandex.Metrica دومین بستر آنالیز وب در دنیا توسعه داده شد، و همچنان جز اصلی آن است. ClickHouse اجازه می دهند که با بیش از 13 تریلیون رکورد در دیتابیس و بیش از 20 میلیارد event در روز، گزارش های مستقیم (On the fly) از داده های non-aggregate تهیه کنیم. این مقاله پیشینه ی تاریخی در ارتباط با اهداف اصلی ClickHouse قبل از آنکه به یک محصول open source تبدیل شود، می دهد.

Yandex.Metrica تولید گزارش های برپایه بازدید و session ها به صورت on the fly و با استفاده از بخش های دلخواه و دوره ی زمانی توسط کاربر انتخاب می شود را انجام می دهد. aggregate های پیچیده معمولاً مورد نیاز هستند، مانند تعداد بازدیدکنندگان unique، داده های جدید برای تهیه گزارش گیری به صورت real-time می رسند.

از آوریل 2014، Yandex.Metrica تقریباً 12 میلیارد event شامل page view و click در روز دریافت کرد. تمام این event ها باید به ترتیب برای ساخت گزارش های سفارشی ذخیره سازی می شدند. یک query ممکن است نیاز به اسکن کردن میلیون ها سطر با زمان کمتر از چند صد میلی ثانیه، یا چند صد میلیون سطر در عرض چند ثانیه داشته باشد.

## استفاده در Yandex.Metrica و دیگر سرویس های Yandex

ClickHouse با چندین اهداف در Yandex.Metrica استفاده می شود. وظیفه اصلی آن ساخت گزارش های آنلاین از داده های non-aggregate می باشد. ClickHouse در یک کلاستر با سایز 374 سرور، که بیش از 20.3 تریلیون سطر در دیتابیس را دارد مورد استفاده قرار می گیرد. اندازه فشرده داده ها، بدون شمارش داده های تکراری و replication، حدود 2 پتابایت می باشد. اندازه ی غیرفشرده داده ها (در فرمت TSV) حدوداً 17 پتابایت می باشد.

ClickHouse همچنین در موارد زیر استفاده می شود:

- ذخیره سازی داده ها برای Session replay از Yandex.Metrica.
- پردازش داده های Intermediate.
- ساخت گزارش های سراسری از آنالیز ها.
- اجرای query ها برای debug کردن موتور Yandex.Metrica.
- آنالیز لاگ های به دست آمده از API ها و user interface.

ClickHouse حداقل در دوازده جای دیگر سرویس Yandex نصب شده است: در search verticals, Market, Direct, Business, Analytics, Mobile Development, AdFox, سرویس های شخصی و..

## داده های Aggregate , Non-Aggregate

یک دیدگاه محبوب وجود دارد که شما باید، داده های خود را به منظور کاهش اندازه داده ها Aggregate کنید.

اما به دلایل زیر، aggregate کردن داده ها راه حل بسیار محدودی است:

- شما باید لیست گزارش های از قبل تعریف شده توسط کاربر که نیاز به تهیه گزارش آنها را دارید، داشته باشید.
- کاربر نمیتواند گزارش های سفارشی تهیه کند.
- در هنگام aggregate کردن تعداد بسیار زیاد key، اندازه ی داده ها کم نمی شود و aggregate بی فایده است.
- برای تعداد زیادی از گزارش ها، aggregate های متنوع و تغییرپذیر زیادی وجود دارد. (انفجار ترکیبی).
- هنگام aggregate کردن key ها با cardinality بالا (مثل URL ها)، اندازه داده ها به اندازه کافی کاهش پیدا نمی کند (کمتر از دو برابر).
- به این دلیل اندازه ی داده ها با aggregate کردن ممکن است به جای شکستن، رشد هم بکند.
- کاربر تمام گزارش هایی که ما تولید کردیم را نگاه نمی کند. بخش بزرگی از محاسبات بی فایده است.
- یکپارچگی منطقی داده ها ممکن است برای aggregate های مختلف نقض شود.

اگر ما هیچ چیزی را aggregate نکنیم و با داده های non-aggregate کار کنیم، در واقع این ممکن است باعث کاهش اندازه ی محاسبات شود.

با این حال، با aggregate کردن، بخش قابل توجهی از کار به صورت آفلاین انجام می شود و نسبتاً آرام به پایان می رسد. در مقابل، محاسبات آنلاین به دلیل اینکه کاربر منتظر نمایش نتایج می باشد، نیازمند محاسبه سریع تا جایی که ممکن است می باشد.

Yandex.Metrica دارای یک سیستم تخصصی برای aggregate کردن داده ها به اسم Metrage می باشد، که برای اکثریت گزارش های مورد استفاده قرار می گیرد. شروع سال 2009، Yandex.Metrica همچنین از یک دیتابیس تخصصی OLAP برای داده های non-aggregate به نام OLAPServer، که قبلاً برای ساخت گزارش ها استفاده می شد، استفاده می کرد. OLAPServer به خوبی روی داده های Non-Aggregate کار می کرد، اما محدودیت های بسیار زیادی داشت که اجازه ی استفاده در تمام گزارش های دلخواه را نمی داد. مواردی از قبیل عدم پشتیبانی از data type ها (فقط عدد)، و عدم توانایی در بروزرسانی افزایشی داده ها به صورت real-time (این کار فقط به rewrite کردن داده ها به صورت روزانه امکان پذیر بود). OLAPServer یک مدیریت دیتابیس نبود اما یک دیتابیس تخصصی بود.

برای حذف محدودیت های OLAPServer و حل مشکلات کار با داده های Non-Aggregate برای تمام گزارش ها، ما مدیریت دیتابیس ClickHouse را توسعه دادیم..

## شروع به کار

### نیازمندی های سیستم

این یک سیستم چند سکویی (Cross-Platform) نمی باشد. این ابزار نیاز به 12.04 (Linux Ubuntu Precise) یا جدیدتر، با معماری x86\_64 و پشتیبانی از SSE 4.2 می باشد. برای چک کردن SSE 4.2 خروجی دستور زیر را بررسی کنید:

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

پیشنهاد می کنیم از Ubuntu TrustyT، Ubuntu Xenial یا Ubuntu Precise استفاده کنید. ترمینال باید از UTF-8 پشتیبانی کند. (به صورت پیش فرض در Ubuntu پشتیبانی می شود).

### نصب

#### نصب از طریق پکیج های Debian/Ubuntu

در فایل [etc/apt/sources.list/](https://etc/apt/sources.list/) (یا در یک فایل جدا [Repo/](https://etc/apt/sources.list.d/clickhouse.list) [etc/apt/sources.list.d/clickhouse.list](https://etc/apt/sources.list.d/clickhouse.list)) زیر را اضافه کنید:



```
/deb http://repo.yandex.ru/clickhouse/deb/stable/ main
```

اگر شما میخواهید جدیدترین نسخه ی تست را استفاده کنید، 'stable' رو به 'testing' تغییر بدید.

سپس دستورات زیر را اجرا کنید:

```
sudo apt-get install dirmngr # optional
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv E0C56BD4 # optional
sudo apt-get update
sudo apt-get install clickhouse-client clickhouse-server
```

شما همچنین می توانید از طریق لینک زیر پکیج ClickHouse را به صورت دستی دانلود و نصب کنید:

[./https://repo.yandex.ru/clickhouse/deb/stable/main](https://repo.yandex.ru/clickhouse/deb/stable/main)

ClickHouse دارای تنظیمات محدودیت دسترسی می باشد. این تنظیمات در فایل 'users.xml' (کنار 'config.xml') می باشد. به صورت پیش فرض دسترسی برای کاربر 'default' از همه جا بدون نیاز به پسورد وجود دارد. 'user/default/networks' را مشاهده کنید. برای اطلاعات بیشتر قسمت "تنظیمات فایل ها" را مشاهده کنید.

## نصب از طریق Source

برای Compile، دستورالعمل های فایل build.md را دنبال کنید:

شما میتوانید پکیج را compile و نصب کنید. شما همچنین می توانید بدون نصب پکیج از برنامه ها استفاده کنید.

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

برای سرور، یک کانالوگ با دیتا بسازید، مانند

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

(قابل تنظیم در تنظیمات سرور). 'chown' را برای کاربر دلخواه اجرا کنید.

به مسیر لاگ ها در تنظیمات سرور توجه کنید (src/dbms/programs/config.xml).

## روش های دیگر نصب

/Docker image: <https://hub.docker.com/r/yandex/clickhouse-server>

پکیج RPM برای CentOS یا <https://github.com/Altinity/clickhouse-rpm-install> RHEL:

Gentoo: [emerge clickhouse](#)

## راه اندازی

برای استارت سرور (به صورت daemon)، دستور زیر را اجرا کنید:

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

لاگ های دایرکتوری `/var/log/clickhouse-server/` directory/ را مشاهده کنید.

اگر سرور استارت نشد، فایل تنظیمات را بررسی کنید `/etc/clickhouse-server/config.xml`.

شما همچنین می توانید سرور را از طریق کنسول راه اندازی کنید:

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

در این مورد که مناسب زمان توسعه می باشد، لاگ ها در کنسول پرینت می شوند. اگر فایل تنظیمات در دایرکتوری جاری باشد، نیازی به مشخص کردن 'config-file--' نمی باشد. به صورت پیش فرض از 'config.xml/' استفاده می شود.

شما می توانید از کلاینت command-line برای اتصال به سرور استفاده کنید:

```
clickhouse-client
```



پارامترهای پیش فرض، نشان از اتصال به localhost:9000 از طرف کاربر 'default' بدون پسورد را می دهد. از کلاینت میتوان برای اتصال به یک سرور remote استفاده کرد. مثال:

```
clickhouse-client --host=example.com
```

برای اطلاعات بیشتر، بخش "کلاینت Command-line" را مشاهده کنید.

چک کردن سیستم:

```
milovidov@hostname:~/work/metrica/src/dbms/src/Client$ ./clickhouse-client
.ClickHouse client version 0.0.18749
.Connecting to localhost:9000
.Connected to ClickHouse server version 0.0.18749

SELECT 1 (:

SELECT 1

┌──1──┐
│ 1 │
└───┘

.rows in set. Elapsed: 0.003 sec 1

(:
```

**تبریک میگم، سیستم کار می کنه!**

برای ادامه آزمایشات، شما میتوانید دیتاست های تستی را دریافت و امتحان کنید.

## OnTime

دانلود داده ها:

```
`for s in `seq 1987 2018
do
`for m in `seq 1 12
do
wget https://transtats.bts.gov/PREZIP/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_${s}_${m}.zip
done
done
```

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

ساخت جدول:

```
) `CREATE TABLE `ontime
,Year` UInt16`
,Quarter` UInt8`
,Month` UInt8`
,DayofMonth` UInt8`
,DayOfWeek` UInt8`
,FlightDate` Date`
,(UniqueCarrier` FixedString(7`
,AirlineID` Int32`
,Carrier` FixedString(2`
,TailNum` String`
,FlightNum` String`
,OriginAirportID` Int32`
,OriginAirportSeqID` Int32`
,OriginCityMarketID` Int32`
,(Origin` FixedString(5`
,OriginCityName` String`
,(OriginState` FixedString(2`
,OriginStateFips` String`
```

,OriginStateName` String`  
,OriginWac` Int32`  
,DestAirportID` Int32`  
,DestAirportSeqID` Int32`  
,DestCityMarketID` Int32`  
, (Dest` FixedString(5`  
,DestCityName` String`  
, (DestState` FixedString(2`  
,DestStateFips` String`  
,DestStateName` String`  
,DestWac` Int32`  
,CRSDepTime` Int32`  
,DepTime` Int32`  
,DepDelay` Int32`  
,DepDelayMinutes` Int32`  
,DepDel15` Int32`  
,DepartureDelayGroups` String`  
,DepTimeBlk` String`  
,TaxiOut` Int32`  
,WheelsOff` Int32`  
,WheelsOn` Int32`  
,TaxiIn` Int32`  
,CRSArrTime` Int32`  
,ArrTime` Int32`  
,ArrDelay` Int32`  
,ArrDelayMinutes` Int32`  
,ArrDel15` Int32`  
,ArrivalDelayGroups` Int32`  
,ArrTimeBlk` String`  
,Cancelled` UInt8`  
, (CancellationCode` FixedString(1`  
,Diverted` UInt8`  
,CRSElapsedTime` Int32`  
,ActualElapsedTime` Int32`  
,AirTime` Int32`  
,Flights` Int32`  
,Distance` Int32`  
,DistanceGroup` UInt8`  
,CarrierDelay` Int32`  
,WeatherDelay` Int32`  
,NASDelay` Int32`  
,SecurityDelay` Int32`  
,LateAircraftDelay` Int32`  
,FirstDepTime` String`  
,TotalAddGTime` String`  
,LongestAddGTime` String`  
,DivAirportLandings` String`  
,DivReachedDest` String`  
,DivActualElapsedTime` String`  
,DivArrDelay` String`  
,DivDistance` String`  
,Div1Airport` String`  
,Div1AirportID` Int32`  
,Div1AirportSeqID` Int32`  
,Div1WheelsOn` String`  
,Div1TotalGTime` String`  
,Div1LongestGTime` String`  
,Div1WheelsOff` String`  
,Div1TailNum` String`  
,Div2Airport` String`  
,Div2AirportID` Int32`  
,Div2AirportSeqID` Int32`  
,Div2WheelsOn` String`  
,Div2TotalGTime` String`  
,Div2LongestGTime` String`  
,Div2WheelsOff` String`  
,Div2TailNum` String`  
,Div3Airport` String`  
,Div3AirportID` Int32`  
,Div3AirportSeqID` Int32`  
,Div3WheelsOn` String`  
,Div3TotalGTime` String`  
,Div3LongestGTime` String`  
,Div3WheelsOff` String`  
,Div3TailNum` String`  
,Div4Airport` String`  
,Div4AirportID` Int32`  
,Div4AirportSeqID` Int32`  
,Div4WheelsOn` String`  
,Div4TotalGTime` String`  
,Div4LongestGTime` String`  
,Div4WheelsOff` String`  
,Div4TailNum` String`  
,Div5Airport` String`  
,Div5AirportID` Int32`  
,Div5AirportSeqID` Int32`  
,Div5WheelsOn` String`  
,Div5TotalGTime` String`  
,Div5LongestGTime` String`  
,Div5WheelsOff` String`  
,Div5TailNum` String`  
,Div6Airport` String`  
,Div6AirportID` Int32`  
,Div6AirportSeqID` Int32`  
,Div6WheelsOn` String`  
,Div6TotalGTime` String`  
,Div6LongestGTime` String`  
,Div6WheelsOff` String`  
,Div6TailNum` String`  
,Div7Airport` String`  
,Div7AirportID` Int32`  
,Div7AirportSeqID` Int32`  
,Div7WheelsOn` String`  
,Div7TotalGTime` String`  
,Div7LongestGTime` String`  
,Div7WheelsOff` String`  
,Div7TailNum` String`  
,Div8Airport` String`  
,Div8AirportID` Int32`  
,Div8AirportSeqID` Int32`  
,Div8WheelsOn` String`  
,Div8TotalGTime` String`  
,Div8LongestGTime` String`  
,Div8WheelsOff` String`  
,Div8TailNum` String`  
,Div9Airport` String`  
,Div9AirportID` Int32`  
,Div9AirportSeqID` Int32`  
,Div9WheelsOn` String`  
,Div9TotalGTime` String`  
,Div9LongestGTime` String`  
,Div9WheelsOff` String`  
,Div9TailNum` String`  
,Div10Airport` String`  
,Div10AirportID` Int32`  
,Div10AirportSeqID` Int32`  
,Div10WheelsOn` String`  
,Div10TotalGTime` String`  
,Div10LongestGTime` String`  
,Div10WheelsOff` String`  
,Div10TailNum` String`  
,Div11Airport` String`  
,Div11AirportID` Int32`  
,Div11AirportSeqID` Int32`  
,Div11WheelsOn` String`  
,Div11TotalGTime` String`  
,Div11LongestGTime` String`  
,Div11WheelsOff` String`  
,Div11TailNum` String`  
,Div12Airport` String`  
,Div12AirportID` Int32`  
,Div12AirportSeqID` Int32`  
,Div12WheelsOn` String`  
,Div12TotalGTime` String`  
,Div12LongestGTime` String`  
,Div12WheelsOff` String`  
,Div12TailNum` String`  
,Div13Airport` String`  
,Div13AirportID` Int32`  
,Div13AirportSeqID` Int32`  
,Div13WheelsOn` String`  
,Div13TotalGTime` String`  
,Div13LongestGTime` String`  
,Div13WheelsOff` String`  
,Div13TailNum` String`  
,Div14Airport` String`  
,Div14AirportID` Int32`  
,Div14AirportSeqID` Int32`  
,Div14WheelsOn` String`  
,Div14TotalGTime` String`  
,Div14LongestGTime` String`  
,Div14WheelsOff` String`  
,Div14TailNum` String`  
,Div15Airport` String`  
,Div15AirportID` Int32`  
,Div15AirportSeqID` Int32`  
,Div15WheelsOn` String`  
,Div15TotalGTime` String`  
,Div15LongestGTime` String`  
,Div15WheelsOff` String`  
,Div15TailNum` String`  
,Div16Airport` String`  
,Div16AirportID` Int32`  
,Div16AirportSeqID` Int32`  
,Div16WheelsOn` String`  
,Div16TotalGTime` String`  
,Div16LongestGTime` String`  
,Div16WheelsOff` String`  
,Div16TailNum` String`  
,Div17Airport` String`  
,Div17AirportID` Int32`  
,Div17AirportSeqID` Int32`  
,Div17WheelsOn` String`  
,Div17TotalGTime` String`  
,Div17LongestGTime` String`  
,Div17WheelsOff` String`  
,Div17TailNum` String`  
,Div18Airport` String`  
,Div18AirportID` Int32`  
,Div18AirportSeqID` Int32`  
,Div18WheelsOn` String`  
,Div18TotalGTime` String`  
,Div18LongestGTime` String`  
,Div18WheelsOff` String`  
,Div18TailNum` String`  
,Div19Airport` String`  
,Div19AirportID` Int32`  
,Div19AirportSeqID` Int32`  
,Div19WheelsOn` String`  
,Div19TotalGTime` String`  
,Div19LongestGTime` String`  
,Div19WheelsOff` String`  
,Div19TailNum` String`  
,Div20Airport` String`  
,Div20AirportID` Int32`  
,Div20AirportSeqID` Int32`  
,Div20WheelsOn` String`  
,Div20TotalGTime` String`  
,Div20LongestGTime` String`  
,Div20WheelsOff` String`  
,Div20TailNum` String`  
,Div21Airport` String`  
,Div21AirportID` Int32`  
,Div21AirportSeqID` Int32`  
,Div21WheelsOn` String`  
,Div21TotalGTime` String`  
,Div21LongestGTime` String`  
,Div21WheelsOff` String`  
,Div21TailNum` String`  
,Div22Airport` String`  
,Div22AirportID` Int32`  
,Div22AirportSeqID` Int32`  
,Div22WheelsOn` String`  
,Div22TotalGTime` String`  
,Div22LongestGTime` String`  
,Div22WheelsOff` String`  
,Div22TailNum` String`  
,Div23Airport` String`  
,Div23AirportID` Int32`  
,Div23AirportSeqID` Int32`  
,Div23WheelsOn` String`  
,Div23TotalGTime` String`  
,Div23LongestGTime` String`  
,Div23WheelsOff` String`  
,Div23TailNum` String`  
,Div24Airport` String`  
,Div24AirportID` Int32`  
,Div24AirportSeqID` Int32`  
,Div24WheelsOn` String`  
,Div24TotalGTime` String`  
,Div24LongestGTime` String`  
,Div24WheelsOff` String`  
,Div24TailNum` String`  
,Div25Airport` String`  
,Div25AirportID` Int32`  
,Div25AirportSeqID` Int32`  
,Div25WheelsOn` String`  
,Div25TotalGTime` String`  
,Div25LongestGTime` String`  
,Div25WheelsOff` String`  
,Div25TailNum` String`  
,Div26Airport` String`  
,Div26AirportID` Int32`  
,Div26AirportSeqID` Int32`  
,Div26WheelsOn` String`  
,Div26TotalGTime` String`  
,Div26LongestGTime` String`  
,Div26WheelsOff` String`  
,Div26TailNum` String`  
,Div27Airport` String`  
,Div27AirportID` Int32`  
,Div27AirportSeqID` Int32`  
,Div27WheelsOn` String`  
,Div27TotalGTime` String`  
,Div27LongestGTime` String`  
,Div27WheelsOff` String`  
,Div27TailNum` String`  
,Div28Airport` String`  
,Div28AirportID` Int32`  
,Div28AirportSeqID` Int32`  
,Div28WheelsOn` String`  
,Div28TotalGTime` String`  
,Div28LongestGTime` String`  
,Div28WheelsOff` String`  
,Div28TailNum` String`  
,Div29Airport` String`  
,Div29AirportID` Int32`  
,Div29AirportSeqID` Int32`  
,Div29WheelsOn` String`  
,Div29TotalGTime` String`  
,Div29LongestGTime` String`  
,Div29WheelsOff` String`  
,Div29TailNum` String`  
,Div30Airport` String`  
,Div30AirportID` Int32`  
,Div30AirportSeqID` Int32`  
,Div30WheelsOn` String`  
,Div30TotalGTime` String`  
,Div30LongestGTime` String`  
,Div30WheelsOff` String`  
,Div30TailNum` String`  
,Div31Airport` String`  
,Div31AirportID` Int32`  
,Div31AirportSeqID` Int32`  
,Div31WheelsOn` String`  
,Div31TotalGTime` String`  
,Div31LongestGTime` String`  
,Div31WheelsOff` String`  
,Div31TailNum` String`  
,Div32Airport` String`  
,Div32AirportID` Int32`  
,Div32AirportSeqID` Int32`  
,Div32WheelsOn` String`  
,Div32TotalGTime` String`  
,Div32LongestGTime` String`  
,Div32WheelsOff` String`  
,Div32TailNum` String`  
,Div33Airport` String`  
,Div33AirportID` Int32`  
,Div33AirportSeqID` Int32`  
,Div33WheelsOn` String`  
,Div33TotalGTime` String`  
,Div33LongestGTime` String`  
,Div33WheelsOff` String`  
,Div33TailNum` String`  
,Div34Airport` String`  
,Div34AirportID` Int32`  
,Div34AirportSeqID` Int32`  
,Div34WheelsOn` String`  
,Div34TotalGTime` String`  
,Div34LongestGTime` String`  
,Div34WheelsOff` String`  
,

```
,Div2LongestGTime` String`  
,Div2WheelsOff` String`  
,Div2TailNum` String`  
,Div3Airport` String`  
,Div3AirportID` Int32`  
,Div3AirportSeqID` Int32`  
,Div3WheelsOn` String`  
,Div3TotalGTime` String`  
,Div3LongestGTime` String`  
,Div3WheelsOff` String`  
,Div3TailNum` String`  
,Div4Airport` String`  
,Div4AirportID` Int32`  
,Div4AirportSeqID` Int32`  
,Div4WheelsOn` String`  
,Div4TotalGTime` String`  
,Div4LongestGTime` String`  
,Div4WheelsOff` String`  
,Div4TailNum` String`  
,Div5Airport` String`  
,Div5AirportID` Int32`  
,Div5AirportSeqID` Int32`  
,Div5WheelsOn` String`  
,Div5TotalGTime` String`  
,Div5LongestGTime` String`  
,Div5WheelsOff` String`  
Div5TailNum` String`  
(ENGINE = MergeTree(FlightDate, (Year, FlightDate), 8192 (
```

Load داده ها:

```
for i in *.zip; do echo $i; unzip -cq $i '*.csv' | sed 's/\.00//g' | clickhouse-client --host=example-perftest01j --query="INSERT INTO  
ontime FORMAT CSVWithNames"; done
```

query ها:

Q0

```
;(select avg(c1) from (select Year, Month, count(*) as c1 from ontime group by Year, Month
```

Q1. تعداد پروازهای به تفکیک روز از تاریخ 2000 تا 2008

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

Q2. تعداد پروازهای بیش از 10 دقیقه تاخیر خورده، گروه بندی براساس روزهای هفته از سال 2000 تا 2008

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

Q3. تعداد تاخیرها براساس airport از سال 2000 تا 2008

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

Q4. تعداد تاخیرها براساس carrier در سال 78

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

Q5. درصد تاخیرها براساس carrier در سال 2007

```

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

```

نسخه ی بهتر query

```

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

```

Q6. مانند query قبلی اما برای طیف وسیعی از سال های 2000 تا 2008

```

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

```

نسخه ی بهتر query

```

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

```

Q7. درصد تاخیر بیش از 10 دقیقه پروازها به تفکیک سال

```

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

```

نسخه ی بهتر query

```

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

```

Q8. مقصدهای پرطرفدار براساس تعداد اتصال های مستقیم شهرها برای سال 2000 تا 2010

```

SELECT DestCityName, uniqExact(OriginCityName) AS u FROM ontime WHERE Year >= 2000 and Year <= 2010 GROUP BY
;DestCityName ORDER BY u DESC LIMIT 10

```

Q9.

```

;select Year, count(*) as c1 from ontime group by Year

```

Q10.

```

select
,min(Year), max(Year), Carrier, count(*) as cnt
,sum(ArrDelayMinutes>30) as flights_delayed
round(sum(ArrDelayMinutes>30)/count(*),2) as rate
FROM ontime
WHERE
('DayOfWeek not in (6,7) and OriginState not in ('AK', 'HI', 'PR', 'VI
('and DestState not in ('AK', 'HI', 'PR', 'VI
'and FlightDate < '2010-01-01
GROUP by Carrier
HAVING cnt > 100000 and max(Year) > 1990
ORDER by rate DESC
;LIMIT 1000

```

query های بیشتر:

```

(SELECT avg(cnt) FROM (SELECT Year,Month,count(*) AS cnt FROM ontime WHERE DepDel15=1 GROUP BY Year,Month
(select avg(c1) from (select Year,Month,count(*) as c1 from ontime group by Year,Month
;SELECT DestCityName, uniqExact(OriginCityName) AS u FROM ontime GROUP BY DestCityName ORDER BY u DESC LIMIT 10
SELECT OriginCityName, DestCityName, count() AS c FROM ontime GROUP BY OriginCityName, DestCityName ORDER BY c DESC
;LIMIT 10
;SELECT OriginCityName, count() AS c FROM ontime GROUP BY OriginCityName ORDER BY c DESC LIMIT 10

```

این تست های performance توسط Vadim Tkachenko انجام شده است. برای اطلاعات بیشتر به لینک های زیر مراجعه کنید:

<https://www.percona.com/blog/2009/10/02/analyzing-air-traffic-performance-with-infobright-and-monetdb>

- [/https://www.percona.com/blog/2009/10/26/air-traffic-queries-in-luciddb](https://www.percona.com/blog/2009/10/26/air-traffic-queries-in-luciddb)
- [/https://www.percona.com/blog/2009/11/02/air-traffic-queries-in-infinidb-early-alpha](https://www.percona.com/blog/2009/11/02/air-traffic-queries-in-infinidb-early-alpha)
- <https://www.percona.com/blog/2014/04/21/using-apache-hadoop-and-impala-together-with-mysql-for-data-analysis>
- [/https://www.percona.com/blog/2016/01/07/apache-spark-with-air-ontime-performance-data](https://www.percona.com/blog/2016/01/07/apache-spark-with-air-ontime-performance-data)
- <http://nickmakos.blogspot.ru/2012/08/analyzing-air-traffic-performance-with.html>

## داده های تاکسی New York

### چطور داده های raw را import کنیم

برای توضیحات بیشتر در ارتباط با دیتاست و موارد مربوط به دانلود به دو لینک <https://github.com/toddwschneider/nyc-taxi-data> و <http://tech.marksblogg.com/billion-nyc-taxi-rides-redshift.html> مراجعه کنید.

دانلود فایل ها حدود 277 گیگابایت داده ی غیرفشرده در قالب فایل های CSV می باشد. دانلود با استفاده از بیش از یک کانکشن 1 Gbit نزدیک 1 ساعت طول می کشد (دانلود موازی از [s3.amazonaws.com](http://s3.amazonaws.com) حداقل نصف کانال 1 Gbit رو جبران می کند). بعضی از فایل ها ممکن است به طول کامل دانلود نشوند. اندازه فایل ها را بررسی کنید و اگر فایلی مشکوک بود، مجدداً دانلود کنید.

بعضی از فایل ها ممکن است دارای سطرهای نامعتبر باشد. با اجرای دستورات زیر این موارد برطرف می شود:

```
sed -E '/(.*){18,}/d' data/yellow_tripdata_2010-02.csv > data/yellow_tripdata_2010-02.csv
sed -E '/(.*){18,}/d' data/yellow_tripdata_2010-03.csv > data/yellow_tripdata_2010-03.csv
mv data/yellow_tripdata_2010-02.csv_ data/yellow_tripdata_2010-02.csv
mv data/yellow_tripdata_2010-03.csv_ data/yellow_tripdata_2010-03.csv
```

سپس داده ها باید در PostgreSQL پیش پردازش شوند. این کار نقاط انتخابی چند ضلعی را ایجاد می کند (برای مطابقت با نقاط بر روی نقشه با مناطق شهر نیویورک) و تمام داده ها را با استفاده از JOIN در یک جدول flat و denormal ترکیب می کند. برای این کار شما نیاز به نصب PostgreSQL با پشتیبانی از PostGIS دارید.

در هنگام اجرای `initialize_database.sh` مراقب باشید و به صورت دستی مجدداً تمام جداول را چک کنید.

PostgreSQL تقریباً 20 تا 30 دقیقه برای پردازش هر ماه زمان نیاز میگیرد، در مجموع حدود 48 ساعت این عملیات طول می کشد.

از طریق دستور زیر شما می توانید تعداد سطرهای دانلود شده را دریافت کنید:

```
"time psql nyc-taxi-data -c "SELECT count(*) FROM trips
count    ###
1298979494
(row 1)

real    7m9.164s
```

(در یکی از پست های مقالات Mark Litwintschik این کمی بیشتر از 1.1 میلیارد سطر گزارش شده است).

حجم داده ها در PostgreSQL 370 گیگابایت می باشد.

Export گیری داده ها از PostgreSQL:

```
COPY
)
,SELECT trips.id
,trips.vendor_id
,trips.pickup_datetime
,trips.dropoff_datetime
,trips.store_and_fwd_flag
,trips.rate_code_id
,trips.pickup_longitude
,trips.pickup_latitude
,trips.dropoff_longitude
,trips.dropoff_latitude
,trips.passenger_count
,trips.trip_distance
```

```

,trips.fare_amount
,trips.extra
,trips.mta_tax
,trips.tip_amount
,trips.tolls_amount
,trips.ehail_fee
,trips.improvement_surcharge
,trips.total_amount
,trips.payment_type
,trips.trip_type
,trips.pickup
,trips.dropoff

,cab_types.type cab_type

,weather.precipitation_tenths_of_mm rain
,weather.snow_depth_mm
,weather.snowfall_mm
,weather.max_temperature_tenths_degrees_celsius max_temp
,weather.min_temperature_tenths_degrees_celsius min_temp
,weather.average_wind_speed_tenths_of_meters_per_second wind

,pick_up.gid pickup_nyct2010_gid
,pick_up.ctlabel pickup_ctlabel
,pick_up.borocode pickup_borocode
,pick_up.borocode pickup_borocode
,pick_up.borocode pickup_borocode
,pick_up.ct2010 pickup_ct2010
,pick_up.borocode pickup_borocode
,pick_up.cdeligibil pickup_cdeligibil
,pick_up.ntacode pickup_ntacode
,pick_up.ntaname pickup_ntaname
,pick_up.puma pickup_puma

,drop_off.gid dropoff_nyct2010_gid
,drop_off.ctlabel dropoff_ctlabel
,drop_off.borocode dropoff_borocode
,drop_off.borocode dropoff_borocode
,drop_off.ct2010 dropoff_ct2010
,drop_off.borocode dropoff_borocode
,drop_off.cdeligibil dropoff_cdeligibil
,drop_off.ntacode dropoff_ntacode
,drop_off.ntaname dropoff_ntaname
drop_off.puma dropoff_puma
FROM trips
LEFT JOIN cab_types
ON trips.cab_type_id = cab_types.id
LEFT JOIN central_park_weather_observations_raw weather
ON weather.date = trips.pickup_datetime::date
LEFT JOIN nyct2010 pick_up
ON pick_up.gid = trips.pickup_nyct2010_gid
LEFT JOIN nyct2010 drop_off
ON drop_off.gid = trips.dropoff_nyct2010_gid
;'TO '/opt/milovidov/nyc-taxi-data/trips.tsv (

```

snapshot از داده ها با سرعت 50 مگابایت در ثانیه انجام می شود. در هنگام ایجاد PostgreSQL snapshot، داده ها را با سرعت 28 مگابایت در ثانیه از روی می خواند. این کار حدود 5 ساعت زمان میبرد. نتیجه کار فایل TSV با حجم 590612904969 بایت می باشد.

ساخت جدول temporary در ClickHouse:



```

CREATE TABLE trips
)
,trip_id          UInt32
,vendor_id        String
,pickup_datetime  DateTime
,(dropoff_datetime Nullable(DateTime
,(store_and_fwd_flag Nullable(FixedString(1
,(rate_code_id     Nullable(UInt8
,(pickup_longitude Nullable(Float64
,(pickup_latitude  Nullable(Float64
,(dropoff_longitude Nullable(Float64
,(dropoff_latitude  Nullable(Float64
,(passenger_count  Nullable(UInt8
,(trip_distance    Nullable(Float64
,(fare_amount      Nullable(Float32
,(extra            Nullable(Float32
,(mta_tax          Nullable(Float32
,(tip_amount       Nullable(Float32
,(tolls_amount     Nullable(Float32
,(ehail_fee        Nullable(Float32
,(improvement_surcharge Nullable(Float32
,(total_amount     Nullable(Float32
,(payment_type     Nullable(String
,(trip_type        Nullable(UInt8
,(pickup           Nullable(String
,(dropoff          Nullable(String
,(cab_type         Nullable(String
,(precipitation     Nullable(UInt8
,(snow_depth       Nullable(UInt8
,(snowfall         Nullable(UInt8
,(max_temperature  Nullable(UInt8
,(min_temperature  Nullable(UInt8
,(average_wind_speed Nullable(UInt8
,(pickup_nyct2010_gid Nullable(UInt8
,(pickup_ctlabel   Nullable(String
,(pickup_borocode   Nullable(UInt8
,(pickup_boroname   Nullable(String
,(pickup_ct2010     Nullable(String
,(pickup_boroct2010 Nullable(String
,(pickup_cdeligibil Nullable(FixedString(1
,(pickup_ntacode    Nullable(String
,(pickup_ntaname    Nullable(String
,(pickup_puma       Nullable(String
,(dropoff_nyct2010_gid Nullable(UInt8
,(dropoff_ctlabel   Nullable(String
,(dropoff_borocode   Nullable(UInt8
,(dropoff_boroname   Nullable(String
,(dropoff_ct2010     Nullable(String
,(dropoff_boroct2010 Nullable(String
,(dropoff_cdeligibil Nullable(String
,(dropoff_ntacode    Nullable(String
,(dropoff_ntaname    Nullable(String
,(dropoff_puma       Nullable(String
;ENGINE = Log (

```

برای تبدیل فیلدها به data type های صحیح تر و در صورت امکان، حذف NULL ها لازم است.

```
time clickhouse-client --query="INSERT INTO trips FORMAT TabSeparated" < trips.tsv
```

```
real 75m56.214s
```

داده ها با سرعت 112 تا 140 مگابایت در ثانیه خوانده می شوند. load کردن داده ها در جدول Log Type در یک 76 Stream، دقیقه زمان کشید. این داده ها در این جدول 142 گیگابایت فضا اشغال می کنند.

(import کردن داده ها به صورت مستقیم از Postgres با استفاده از **COPY ... TO PROGRAM** هم امکان پذیر است.)

متأسفانه، تمام فیلد های مرتبط با آب و هوا (precipitation...average\_wind\_speed) با Null پر شدند. به خاطر همین، ما از دیتاست نهایی اینها رو حذف کردیم.

برای شروع، ما یک جدول در یک سرور ایجاد کردیم. بعداً ما یک جدول توزیع شده می سازیم.

یک جدول خلاصه ایجاد و پر کنید:

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

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

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

CAST(assumeNotNull(pickup_ntaname) AS Enum16('' = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-
Prince's Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park
City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13,
'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle
Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn
Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' =
28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31,
'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Claremont-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36,
'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South'
= 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Douglaston-Little
Neck' = 45, 'Dyker Heights' = 46, 'East Concourse Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush Forest Hills' = 49,
```

'Neck' = 43, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52, 'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' = 61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82, 'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90, 'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton' = 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99, 'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner's Harbor-Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' = 107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North Riverdale-Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123, 'Oakwood-Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126, 'Old Astoria' = 127, 'Old Town-Dongan Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-Country Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomono-Flushing Heights-Hillcrest' = 134, 'Port Richmond' = 135, 'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139, 'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143, 'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156, 'Springfield Gardens South-Brookville' = 157, 'Spuyten Duyvil-Kingsbridge' = 158, 'St. Albans' = 159, 'Stapleton-Rosebank' = 160, 'Starrett City' = 161, 'Steinway' = 162, 'Stuyvesant Heights' = 163, 'Stuyvesant Town-Cooper Village' = 164, 'Sunset Park East' = 165, 'Sunset Park West' = 166, 'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill' = 167, 'Turtle Bay-East Midtown' = 168, 'University Heights-Morris Heights' = 169, 'Upper East Side-Carnegie Hill' = 170, 'Upper West Side' = 171, 'Van Cortlandt Village' = 172, 'Van Nest-Morris Park-Westchester Square' = 173, 'Washington Heights North' = 174, 'Washington Heights South' = 175, 'West Brighton' = 176, 'West Concourse' = 177, 'West Farms-Bronx River' = 178, 'West New Brighton-New Brighton-St. George' = 179, 'West Village' = 180, 'Westchester-Unionport' = 181, 'Westerleigh' = 182, 'Whitestone' = 183, 'Williamsbridge-Olinville' = 184, 'Williamsburg' = 185, 'Windsor Terrace' = 186, 'Woodhaven' = 187, 'Woodlawn-Wakefield' = 188, 'Woodside' = 189, 'Yorkville' = 190, 'park-cemetery-etc-Bronx' = 191, 'park-cemetery-etc-Brooklyn' = 192, 'park-cemetery-etc-Manhattan' = 193, 'park-cemetery-etc-Queens' = 194, 'park-cemetery-etc-Staten Island' = 195)) AS pickup\_ntaname

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

,assumeNotNull(dropoff\_nyct2010\_gid) AS dropoff\_nyct2010\_gid

,toFloat32(ifNull(dropoff\_ctlabel, '0')) AS dropoff\_ctlabel

,assumeNotNull(dropoff\_borocode) AS dropoff\_borocode

CAST(assumeNotNull(dropoff\_boroname) AS Enum8('Manhattan' = 1, 'Queens' = 4, 'Brooklyn' = 3, '' = 0, 'Bronx' = 2, 'Staten Island' = 5)) AS dropoff\_boroname

,toFixedString(ifNull(dropoff\_ct2010, '000000'), 6) AS dropoff\_ct2010

,toFixedString(ifNull(dropoff\_boroc2010, '0000000'), 7) AS dropoff\_boroc2010

,CAST(assumeNotNull(ifNull(dropoff\_cdeligibil, '')) AS Enum8('' = 0, 'E' = 1, 'I' = 2)) AS dropoff\_cdeligibil

,toFixedString(ifNull(dropoff\_ntacode, '0000'), 4) AS dropoff\_ntacode

CAST(assumeNotNull(dropoff\_ntaname) AS Enum16('' = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-Prince's Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31, 'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Claremont-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Douglaston-Little Neck' = 45, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52, 'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' = 61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes

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

```
toUInt16(ifNull(dropoff_puma, '0')) AS dropoff_puma
```

FROM trips

این کار با سرعت 428 هزار رکورد در ثانیه و 3030 ثانیه طول خواهد کشید. برای load سریعتر، شما می توانید یک جدول با موتور Log به جای MergeTree بسازید. در این مورد، دانلود سریعتر از 200 ثانیه کار می کند.

این جدول 126 گیابایت فضا بر روی دیسک اشغال می کند.

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

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

```
|-(formatReadableSize(sum(bytes-r
```

GiB 126.18 |

در میان چیزهای دیگر، شما می‌تونید از دستور OPTIMIZE بر روی MergeTree استفاده کنید، اما از آنجایی که بدون این دستور همه چیز خوب است، اجرای این دستور ضروری نیست..

## نتایج بر روی یک سرور

:Q1

```
SELECT cab type, count(*) FROM trips mergetree GROUP BY cab type
```

```
.seconds 0.490
```

:Q2

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

```
.seconds 1.224
```

:Q3

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

.seconds 2.104

:Q4

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

.seconds 3.593

کانفیگ سرور به این صورت بود:

,Two Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz, 16 physical kernels total

,GiB RAM 128

8x6 TB HD on hardware RAID-5

زمان اجرا query ها از زمان دومین اجرا بهتر می شود، چون query ها داده ها رو از فایل system cache می خوانند. در ادامه cache دیگری رخ نمیدهد: داده ها در هر اجرا خوانده و پردازش شده اند.

ساخت جداول در سه سرور:

در هر سرور دستور زیر را اجرا کنید:

```
CREATE TABLE default.trips_mergetree_third ( trip_id UInt32, vendor_id Enum8('1' = 1, '2' = 2, 'CMT' = 3, 'VTS' = 4, 'DDS' = 5, 'B02512' = 10, 'B02598' = 11, 'B02617' = 12, 'B02682' = 13, 'B02764' = 14), pickup_date Date, pickup_datetime DateTime, dropoff_date Date, dropoff_datetime DateTime, store_and_fwd_flag UInt8, rate_code_id UInt8, pickup_longitude Float64, pickup_latitude Float64, dropoff_longitude Float64, dropoff_latitude Float64, passenger_count UInt8, trip_distance Float64, fare_amount Float32, extra Float32, mta_tax Float32, tip_amount Float32, tolls_amount Float32, ehail_fee Float32, improvement_surcharge Float32, total_amount Float32, payment_type_ Enum8('UNK' = 0, 'CSH' = 1, 'CRE' = 2, 'NOC' = 3, 'DIS' = 4), trip_type UInt8, pickup FixedString(25), dropoff FixedString(25), cab_type Enum8('yellow' = 1, 'green' = 2, 'uber' = 3), pickup_nyct2010_gid UInt8, pickup_ctlabel Float32, pickup_borocode UInt8, pickup_boroname Enum8('' = 0, 'Manhattan' = 1, 'Bronx' = 2, 'Brooklyn' = 3, 'Queens' = 4, 'Staten Island' = 5), pickup_ct2010 FixedString(6), pickup_boroct2010 FixedString(7), pickup_cdeligibil Enum8('' = 0, 'E' = 1, 'I' = 2), pickup_ntacode FixedString(4), pickup_ntaname Enum16('' = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-Prince's Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31, 'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Claremont-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Douglaston-Little Neck' = 45, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52, 'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' = 61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82, 'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90, 'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton' = 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99, 'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner's Harbor-Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' = 107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North Riverdale-Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123, 'Oakwood-Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126,
```



'Old Astoria' = 127, 'Old Town-Dongan Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-Country Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomonok-Flushing Heights-Hillcrest' = 134, 'Port Richmond' = 135, 'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139, 'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143, 'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156, 'Springfield Gardens South-Brookville' = 157, 'Spuyten Duyvil-Kingsbridge' = 158, 'St. Albans' = 159, 'Stapleton-Rosebank' = 160, 'Starrett City' = 161, 'Steinway' = 162, 'Stuyvesant Heights' = 163, 'Stuyvesant Town-Cooper Village' = 164, 'Sunset Park East' = 165, 'Sunset Park West' = 166, 'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill' = 167, 'Turtle Bay-East Midtown' = 168, 'University Heights-Morris Heights' = 169, 'Upper East Side-Carnegie Hill' = 170, 'Upper West Side' = 171, 'Van Cortlandt Village' = 172, 'Van Nest-Morris Park-Westchester Square' = 173, 'Washington Heights North' = 174, 'Washington Heights South' = 175, 'West Brighton' = 176, 'West Concourse' = 177, 'West Farms-Bronx River' = 178, 'West New Brighton-New Brighton-St. George' = 179, 'West Village' = 180, 'Westchester-Unionport' = 181, 'Westerleigh' = 182, 'Whitestone' = 183, 'Williamsbridge-Olinville' = 184, 'Williamsburg' = 185, 'Windsor Terrace' = 186, 'Woodhaven' = 187, 'Woodlawn-Wakefield' = 188, 'Woodside' = 189, 'Yorkville' = 190, 'park-cemetery-etc-Bronx' = 191, 'park-cemetery-etc-Brooklyn' = 192, 'park-cemetery-etc-Manhattan' = 193, 'park-cemetery-etc-Queens' = 194, 'park-cemetery-etc-Staten Island' = 195), pickup\_puma UInt16, dropoff\_nyct2010\_gid UInt8, dropoff\_ctlabel Float32, dropoff\_borocode UInt8, dropoff\_boroname Enum8('' = 0, 'Manhattan' = 1, 'Bronx' = 2, 'Brooklyn' = 3, 'Queens' = 4, 'Staten Island' = 5), dropoff\_ct2010 FixedString(6), dropoff\_boroc2010 FixedString(7), dropoff\_cdeligibil Enum8('' = 0, 'E' = 1, 'I' = 2), dropoff\_ntacode FixedString(4), dropoff\_ntaname Enum16('' = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-Prince's Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31, 'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Claremont-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Douglaston-Little Neck' = 45, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52, 'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' = 61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82, 'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90, 'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton' = 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99, 'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner's Harbor-Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' = 107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North Riverdale-Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123, 'Oakwood-Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126, 'Old Astoria' = 127, 'Old Town-Dongan Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-Country Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomonok-Flushing Heights-Hillcrest' = 134, 'Port Richmond' = 135, 'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139, 'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143, 'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156, 'Springfield Gardens South-Brookville' = 157, 'Spuyten Duyvil-Kingsbridge' = 158, 'St. Albans' = 159, 'Stapleton-Rosebank' = 160, 'Starrett City' = 161, 'Steinway' = 162, 'Stuyvesant Heights' = 163, 'Stuyvesant Town-Cooper Village' = 164, 'Sunset Park East' = 165, 'Sunset Park West' = 166, 'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill' = 167, 'Turtle Bay-East Midtown' = 168, 'University Heights-Morris Heights' = 169, 'Upper East Side-Carnegie Hill' = 170, 'Upper West Side' = 171, 'Van Cortlandt Village' = 172, 'Van Nest-Morris Park-Westchester Square' = 173, 'Washington Heights North' = 174, 'Washington Heights South' = 175, 'West Brighton' = 176, 'West Concourse' = 177, 'West Farms-Bronx River' = 178, 'West New Brighton-New Brighton-St. George' = 179, 'West Village' = 180, 'Westchester-Unionport' = 181, 'Westerleigh' = 182, 'Whitestone' = 183, 'Williamsbridge-Olinville' = 184, 'Williamsburg' = 185, 'Windsor Terrace' = 186, 'Woodhaven' = 187, 'Woodlawn-Wakefield' = 188, 'Woodside' = 189, 'Yorkville' = 190, 'park-cemetery-etc-Bronx' = 191, 'park-cemetery-etc-Brooklyn' = 192, 'park-cemetery-etc-Manhattan' = 193,

```
'park-cemetery-etc-Queens' = 194, 'park-cemetery-etc-Staten Island' = 195), dropoff_puma UInt16) ENGINE = MergeTree(pickup_date, pickup_datetime, 8192
```

بر روی سرور source دستور زیر را وارد کنید:

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

query زیر داده‌ها را توزیع مجدد می‌کند:

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

این query 2454 ثانیه زمان میبرد.

در سه سرور:

Q1: 0.212 ثانیه.

Q2: 0.438 ثانیه.

Q3: 0.733 ثانیه.

Q4: 1.241 ثانیه.

از آنجایی که query ها به صورت خطی scale شده اند، از نتایج به دست آمده شگفتی وجود ندارد.

ما همچنین نتایج زیر را از اجرای این query در کلاستر 140 سرور دریافت کردیم:

Q1: 0.028 ثانیه.

Q2: 0.043 ثانیه.

Q3: 0.051 ثانیه.

Q4: 0.072 ثانیه.

در این مورد، زمان پردازش query براساس latency شبکه مشخص می‌شود. ما این query ها را با استفاده از یک مشتری واقع در دیتاستر Yandex در فنلاند در یک کلاستر روسیه دریافت کردیم، که latency آن حدود 20 میلی ثانیه به نتایج اضافه کرد.

## نتایج

نودها	Q1	Q2	Q3	Q4
1	0.490	1.224	2.104	3.593
3	0.212	0.438	0.733	1.241
140	0.028	0.043	0.051	0.072

## بنچمارک AMPLab Big Data

ببینید [/https://amplab.cs.berkeley.edu/benchmark](https://amplab.cs.berkeley.edu/benchmark)

با یک اکانت مجانی در <https://aws.amazon.com> ثبت نام کنید. شما نیاز به ایمیل، شماره تلفن و credit card دارید. یک Access key جدید از [https://console.aws.amazon.com/iam/home?nc2=h\\_m\\_sc#security\\_credential](https://console.aws.amazon.com/iam/home?nc2=h_m_sc#security_credential) دریافت کنید.

در کنسول این دستورات را وارد کنید:

```
sudo apt-get install s3cmd
mkdir tiny; cd tiny
./s3cmd sync s3://big-data-benchmark/pavlo/text-deflate/tiny
.. cd
mkdir 1node; cd 1node
./s3cmd sync s3://big-data-benchmark/pavlo/text-deflate/1node
.. cd
mkdir 5nodes; cd 5nodes
./s3cmd sync s3://big-data-benchmark/pavlo/text-deflate/5nodes
.. cd
```

این query های ClickHouse را اجرا کنید:



```

CREATE TABLE rankings_tiny
)
,pageURL String
,pageRank UInt32
avgDuration UInt32
;ENGINE = Log (

CREATE TABLE uservisits_tiny
)
,sourceIP String
,destinationURL String
,visitDate Date
,adRevenue Float32
,UserAgent String
,(cCode FixedString(3
,(lCode FixedString(6
,searchWord String
duration UInt32
;(ENGINE = MergeTree(visitDate, visitDate, 8192 (

CREATE TABLE rankings_1node
)
,pageURL String
,pageRank UInt32
avgDuration UInt32
;ENGINE = Log (

CREATE TABLE uservisits_1node
)
,sourceIP String
,destinationURL String
,visitDate Date
,adRevenue Float32
,UserAgent String
,(cCode FixedString(3
,(lCode FixedString(6
,searchWord String
duration UInt32
;(ENGINE = MergeTree(visitDate, visitDate, 8192 (

CREATE TABLE rankings_5nodes_on_single
)
,pageURL String
,pageRank UInt32
avgDuration UInt32
;ENGINE = Log (

CREATE TABLE uservisits_5nodes_on_single
)
,sourceIP String
,destinationURL String
,visitDate Date
,adRevenue Float32
,UserAgent String
,(cCode FixedString(3
,(lCode FixedString(6
,searchWord String
duration UInt32
;(ENGINE = MergeTree(visitDate, visitDate, 8192 (

```

به کنسول برگردید و دستورات زیر را مجددا اجرا کنید:

```
for i in tiny/rankings/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perftest01j --
query="INSERT INTO rankings_tiny FORMAT CSV"; done
for i in tiny/uservisits/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perftest01j --
query="INSERT INTO uservisits_tiny FORMAT CSV"; done
for i in 1node/rankings/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perftest01j --
query="INSERT INTO rankings_1node FORMAT CSV"; done
for i in 1node/uservisits/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perftest01j --
query="INSERT INTO uservisits_1node FORMAT CSV"; done
for i in 5nodes/rankings/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perftest01j --
query="INSERT INTO rankings_5nodes_on_single FORMAT CSV"; done
for i in 5nodes/uservisits/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perftest01j --
query="INSERT INTO uservisits_5nodes_on_single FORMAT CSV"; done
```

data sample های گرفتن query

```
SELECT pageURL, pageRank FROM rankings_1node WHERE pageRank > 1000

(SELECT substring(sourceIP, 1, 8), sum(adRevenue) FROM uservisits_1node GROUP BY substring(sourceIP, 1, 8

SELECT
,sourceIP
,sum(adRevenue) AS totalRevenue
avg(pageRank) AS pageRank
FROM rankings_1node ALL INNER JOIN
)
SELECT
,sourceIP
,destinationURL AS pageURL
adRevenue
FROM uservisits_1node
('WHERE (visitDate > '1980-01-01') AND (visitDate < '1980-04-01
USING pageURL (
GROUP BY sourceIP
ORDER BY totalRevenue DESC
LIMIT 1
```

# WikiStat

ببینید: [/http://dumps.wikimedia.org/other/pagecounts-raw](http://dumps.wikimedia.org/other/pagecounts-raw)

ساخت جدول:

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

load دیتا

```
for i in {2007..2016}; do for j in {01..12}; do echo $i-$j >&2; curl -sSL "http://dumps.wikimedia.org/other/pagecounts-raw/$i/$i-$j/" | grep -oE 'pagecounts-[0-9]+-[0-9]+\.gz'; done; done | sort | uniq | tee links.txt
cat links.txt | while read link; do wget http://dumps.wikimedia.org/other/pagecounts-raw/${echo $link | sed -r 's/pagecounts-([0-9]{4})([0-9]{2})([0-9]{2})-[0-9]+\.gz/\1'}/${echo $link | sed -r 's/pagecounts-([0-9]{4})([0-9]{2})([0-9]{2})-[0-9]+\.gz/\1-\2'}/${link}; done
ls -l /opt/wikistat/ | grep gz | while read i; do echo $i; gzip -cd /opt/wikistat/$i | ./wikistat-loader --time="$(echo -n $i | sed -r 's/pagecounts-([0-9]{4})([0-9]{2})([0-9]{2})-([0-9]{2})([0-9]{2})([0-9]{2})\.\gz/\1-\2-\3 \4-00-00/')" | clickhouse-client --query="INSERT INTO wikistat FORMAT TabSeparated"; done
```

## ترابایت از لاگ های کلیک از سرویس Criteo

داده ها را از <http://labs.criteo.com/downloads/download-terabyte-click-logs> دانلود کنید.

جدول را برای import لاگ ها ایجاد کنید:

```
CREATE TABLE criteo_log (date Date, clicked UInt8, int1 Int32, int2 Int32, int3 Int32, int4 Int32, int5 Int32, int6 Int32, int7 Int32, int8 Int32, int9 Int32, int10 Int32, int11 Int32, int12 Int32, int13 Int32, cat1 String, cat2 String, cat3 String, cat4 String, cat5 String, cat6 String, cat7 String, cat8 String, cat9 String, cat10 String, cat11 String, cat12 String, cat13 String, cat14 String, cat15 String, cat16 String, cat17 String, cat18 String, cat19 String, cat20 String, cat21 String, cat22 String, cat23 String, cat24 String, cat25 String, cat26 String) ENGINE = Log
```

داده ها را دانلود کنید:

```
for i in {00..23}; do echo $i; zcat datasets/criteo/day_${i#0}.gz | sed -r 's/^/2000-01-'\${i/00/24}'\t/' | clickhouse-client --host=example-perftest01j --query="INSERT INTO criteo_log FORMAT TabSeparated"; done
```

یک جدول برای داده های تبدیل شده ایجاد کنید:

```

CREATE TABLE critео
)
,date Date
,clicked UInt8
,int1 Int32
,int2 Int32
,int3 Int32
,int4 Int32
,int5 Int32
,int6 Int32
,int7 Int32
,int8 Int32
,int9 Int32
,int10 Int32
,int11 Int32
,int12 Int32
,int13 Int32
,icat1 UInt32
,icat2 UInt32
,icat3 UInt32
,icat4 UInt32
,icat5 UInt32
,icat6 UInt32
,icat7 UInt32
,icat8 UInt32
,icat9 UInt32
,icat10 UInt32
,icat11 UInt32
,icat12 UInt32
,icat13 UInt32
,icat14 UInt32
,icat15 UInt32
,icat16 UInt32
,icat17 UInt32
,icat18 UInt32
,icat19 UInt32
,icat20 UInt32
,icat21 UInt32
,icat22 UInt32
,icat23 UInt32
,icat24 UInt32
,icat25 UInt32
,icat26 UInt32
(ENGINE = MergeTree(date, intHash32(icat1), (date, intHash32(icat1)), 8192 (

```

داده ها را از لاگ raw انتقال و به جدول دوم وارد کنید:

```

INSERT INTO critео SELECT date, clicked, int1, int2, int3, int4, int5, int6, int7, int8, int9, int10, int11, int12, int13,
reinterpretAsUInt32(unhex(cat1)) AS icat1, reinterpretAsUInt32(unhex(cat2)) AS icat2, reinterpretAsUInt32(unhex(cat3)) AS icat3,
reinterpretAsUInt32(unhex(cat4)) AS icat4, reinterpretAsUInt32(unhex(cat5)) AS icat5, reinterpretAsUInt32(unhex(cat6)) AS icat6,
reinterpretAsUInt32(unhex(cat7)) AS icat7, reinterpretAsUInt32(unhex(cat8)) AS icat8, reinterpretAsUInt32(unhex(cat9)) AS icat9,
reinterpretAsUInt32(unhex(cat10)) AS icat10, reinterpretAsUInt32(unhex(cat11)) AS icat11, reinterpretAsUInt32(unhex(cat12)) AS
icat12, reinterpretAsUInt32(unhex(cat13)) AS icat13, reinterpretAsUInt32(unhex(cat14)) AS icat14,
reinterpretAsUInt32(unhex(cat15)) AS icat15, reinterpretAsUInt32(unhex(cat16)) AS icat16, reinterpretAsUInt32(unhex(cat17)) AS
icat17, reinterpretAsUInt32(unhex(cat18)) AS icat18, reinterpretAsUInt32(unhex(cat19)) AS icat19,
reinterpretAsUInt32(unhex(cat20)) AS icat20, reinterpretAsUInt32(unhex(cat21)) AS icat21, reinterpretAsUInt32(unhex(cat22)) AS
icat22, reinterpretAsUInt32(unhex(cat23)) AS icat23, reinterpretAsUInt32(unhex(cat24)) AS icat24,
;reinterpretAsUInt32(unhex(cat25)) AS icat25, reinterpretAsUInt32(unhex(cat26)) AS icat26 FROM critео_log

;DROP TABLE critео_log

```

# Star Schema بنچمارک

از لینک روبرو dbgen رو کامپایل کنید. <https://github.com/vadimtk/ssb-dbgen>

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

در هنگام پردازش چند warnings نمایش داده می شود که مشکلی نیست و طبیعی است.

dbgen و dists.dss را در یک جا با 800 گیگابایت فضای حالی دیسک قرار دهید.

تولید داده ها:

```
dbgen -s 1000 -T c/.
dbgen -s 1000 -T l/.
```

ساخت جداول در ClickHouse

```

) CREATE TABLE lineorder
,LO_ORDERKEY      UInt32
,LO_LINENUMBER    UInt8
,LO_CUSTKEY       UInt32
,LO_PARTKEY       UInt32
,LO_SUPPKEY       UInt32
,LO_ORDERDATE     Date
,LO_ORDERPRIORITY String
,LO_SHIPPRIORITY  UInt8
,LO_QUANTITY      UInt8
,LO_EXTENDEDPRICE UInt32
,LO_ORDTOTALPRICE UInt32
,LO_DISCOUNT     UInt8
,LO_REVENUE        UInt32
,LO_SUPPLYCOST    UInt32
,LO_TAX           UInt8
,LO_COMMITDATE    Date
LO_SHIPMODE       String
;(Engine=MergeTree(LO_ORDERDATE,(LO_ORDERKEY,LO_LINENUMBER,LO_ORDERDATE)),8192(

) CREATE TABLE customer
,C_CUSTKEY      UInt32
,C_NAME        String
,C_ADDRESS     String
,C_CITY        String
,C_NATION      String
,C_REGION      String
,C_PHONE       String
,C_MKTSEGMENT  String
C_FAKEDATE     Date
;(Engine=MergeTree(C_FAKEDATE,(C_CUSTKEY,C_FAKEDATE)),8192(

) CREATE TABLE part
,P_PARTKEY      UInt32
,P_NAME         String
,P_MFGR         String
,P_CATEGORY     String
,P_BRAND        String
,P_COLOR        String
,P_TYPE         String
,P_SIZE        UInt8
,P_CONTAINER    String
P_FAKEDATE     Date
;(Engine=MergeTree(P_FAKEDATE,(P_PARTKEY,P_FAKEDATE)),8192(

;()CREATE TABLE lineorderd AS lineorder ENGINE = Distributed(perftest_3shards_1replicas, default, lineorder, rand
;()CREATE TABLE customerd AS customer ENGINE = Distributed(perftest_3shards_1replicas, default, customer, rand
;()CREATE TABLE partd AS part ENGINE = Distributed(perftest_3shards_1replicas, default, part, rand

```

برای تست بر روی یک سرور، فقط از جداول MergeTree استفاده کنید. برای تست توزیع شده، شما نیاز به کانفیگ `perftest_3shards_1replicas` در فایل کانفیگ را دارید. در ادامه جداول MergeTree را در هر سرور ایجاد کنید و موارد بالا را توزیع کنید.

دانلود داده ها (تغییر `customer` به `customerd` در نسخه ی توزیع شده):

```

"cat customer.tbl | sed 's/$/2000-01-01/' | clickhouse-client --query "INSERT INTO customer FORMAT CSV
"cat lineorder.tbl | clickhouse-client --query "INSERT INTO lineorder FORMAT CSV

```

## Anonymized Yandex.Metrica Data

Dataset consists of two tables containing anonymized data about hits (`hits_v1`) and visits (`visits_v1`) of .Yandex.Metrica. Each of the tables can be downloaded as a compressed `tsv.xz` file or as prepared partitions

## Obtaining Tables from Prepared Partitions

## :Download and import hits

```
curl -O https://clickhouse-datasets.s3.yandex.net/hits/partitions/hits_v1.tar
tar xvf hits_v1.tar -C /var/lib/clickhouse # path to ClickHouse data directory
check permissions on unpacked data, fix if required ##
sudo service clickhouse-server restart
"clickhouse-client --query "SELECT COUNT(*) FROM datasets.hits_v1
```

## :Download and import visits

```
curl -O https://clickhouse-datasets.s3.yandex.net/visits/partitions/visits_v1.tar
tar xvf visits_v1.tar -C /var/lib/clickhouse # path to ClickHouse data directory
check permissions on unpacked data, fix if required ##
sudo service clickhouse-server restart
"clickhouse-client --query "SELECT COUNT(*) FROM datasets.visits_v1
```

# Obtaining Tables from Compressed tsv-file

## Download and import hits from compressed tsv-file

```
curl https://clickhouse-datasets.s3.yandex.net/hits/tsv/hits_v1.tsv.xz | unxz --threads=`nproc` > hits_v1.tsv
now create table ##
"clickhouse-client --query "CREATE DATABASE IF NOT EXISTS datasets
clickhouse-client --query "CREATE TABLE datasets.hits_v1 ( WatchID UInt64, JavaEnable UInt8, Title String, GoodEvent Int16,
EventTime DateTime, EventDate Date, CounterID UInt32, ClientIP UInt32, ClientIP6 FixedString(16), RegionID UInt32, UserID
UInt64, CounterClass Int8, OS UInt8, UserAgent UInt8, URL String, Referer String, URLDomain String, RefererDomain String,
Refresh UInt8, IsRobot UInt8, RefererCategories Array(UInt16), URLCategories Array(UInt16), URLRegions Array(UInt32),
RefererRegions Array(UInt32), ResolutionWidth UInt16, ResolutionHeight UInt16, ResolutionDepth UInt8, FlashMajor UInt8,
FlashMinor UInt8, FlashMinor2 String, NetMajor UInt8, NetMinor UInt8, UserAgentMajor UInt16, UserAgentMinor FixedString(2),
CookieEnable UInt8, JavascriptEnable UInt8, IsMobile UInt8, MobilePhone UInt8, MobilePhoneModel String, Params String,
IPNetworkID UInt32, TrafficSourceID Int8, SearchEngineID UInt16, SearchPhrase String, AdvEngineID UInt8, IsArtificial UInt8,
WindowClientWidth UInt16, WindowClientHeight UInt16, ClientTimeZone Int16, ClientEventTime DateTime, SilverlightVersion1
UInt8, SilverlightVersion2 UInt8, SilverlightVersion3 UInt32, SilverlightVersion4 UInt16, PageCharset String, CodeVersion UInt32,
IsLink UInt8, IsDownload UInt8, IsNotBounce UInt8, FUniqID UInt64, HID UInt32, IsOldCounter UInt8, IsEvent UInt8, IsParameter
UInt8, DontCountHits UInt8, WithHash UInt8, HitColor FixedString(1), UTCEventTime DateTime, Age UInt8, Sex UInt8, Income
UInt8, Interests UInt16, Robotness UInt8, GeneralInterests Array(UInt16), RemoteIP UInt32, RemoteIP6 FixedString(16),
WindowName Int32, OpenerName Int32, HistoryLength Int16, BrowserLanguage FixedString(2), BrowserCountry FixedString(2),
SocialNetwork String, SocialAction String, HTTPError UInt16, SendTiming Int32, DNSTiming Int32, ConnectTiming Int32,
ResponseStartTiming Int32, ResponseEndTiming Int32, FetchTiming Int32, RedirectTiming Int32, DOMInteractiveTiming Int32,
DOMContentLoadedTiming Int32, DOMCompleteTiming Int32, LoadEventStartTiming Int32, LoadEventEndTiming Int32,
NSToDOMContentLoadedTiming Int32, FirstPaintTiming Int32, RedirectCount Int8, SocialSourceNetworkID UInt8,
SocialSourcePage String, ParamPrice Int64, ParamOrderID String, ParamCurrency FixedString(3), ParamCurrencyID UInt16,
GoalsReached Array(UInt32), OpenstatServiceName String, OpenstatCampaignID String, OpenstatAdID String,
OpenstatSourceID String, UTMSource String, UTMMedium String, UTMCampaign String, UTMContent String, UTMTerm String,
FromTag String, HasGCLID UInt8, RefererHash UInt64, URLHash UInt64, CLID UInt32, YCLID UInt64, ShareService String,
ShareURL String, ShareTitle String, ParsedParams Nested(Key1 String, Key2 String, Key3 String, Key4 String, Key5 String,
ValueDouble Float64), IslandID FixedString(16), RequestNum UInt32, RequestTry UInt8) ENGINE = MergeTree() PARTITION BY
toYYYYMM(EventDate) ORDER BY (CounterID, EventDate, intHash32(UserID)) SAMPLE BY intHash32(UserID) SETTINGS
"index_granularity = 8192
import data ##
cat hits_v1.tsv | clickhouse-client --query "INSERT INTO datasets.hits_v1 FORMAT TSV" --max_insert_block_size=100000
optionally you can optimize table ##
"clickhouse-client --query "OPTIMIZE TABLE datasets.hits_v1 FINAL
"clickhouse-client --query "SELECT COUNT(*) FROM datasets.hits_v1
```

## Download and import visits from compressed tsv-file



```
curl https://clickhouse-datasets.s3.yandex.net/visits/tsv/visits_v1.tsv.xz | unxz --threads=`nproc` > visits_v1.tsv
now create table ##
"clickhouse-client --query "CREATE DATABASE IF NOT EXISTS datasets
clickhouse-client --query "CREATE TABLE datasets.visits_v1 ( CounterID UInt32, StartDate Date, Sign Int8, IsNew UInt8, VisitID
UInt64, UserID UInt64, StartTime DateTime, Duration UInt32, UTCStartTime DateTime, PageViews Int32, Hits Int32, IsBounce
UInt8, Referrer String, StartURL String, ReferrerDomain String, StartURLDomain String, EndURL String, LinkURL String,
IsDownload UInt8, TrafficSourceID Int8, SearchEngineID UInt16, SearchPhrase String, AdvEngineID UInt8, PlacelD Int32,
ReferrerCategories Array(UInt16), URLCategories Array(UInt16), URLRegions Array(UInt32), ReferrerRegions Array(UInt32),
IsYandex UInt8, GoalReachesDepth Int32, GoalReachesURL Int32, GoalReachesAny Int32, SocialSourceNetworkID UInt8,
SocialSourcePage String, MobilePhoneModel String, ClientEventTime DateTime, RegionID UInt32, ClientIP UInt32, ClientIP6
FixedString(16), RemoteIP UInt32, RemoteIP6 FixedString(16), IPNetworkID UInt32, SilverlightVersion3 UInt32, CodeVersion
UInt32, ResolutionWidth UInt16, ResolutionHeight UInt16, UserAgentMajor UInt16, UserAgentMinor UInt16, WindowClientWidth
UInt16, WindowClientHeight UInt16, SilverlightVersion2 UInt8, SilverlightVersion4 UInt16, FlashVersion3 UInt16, FlashVersion4
UInt16, ClientTimeZone Int16, OS UInt8, UserAgent UInt8, ResolutionDepth UInt8, FlashMajor UInt8, FlashMinor UInt8, NetMajor
UInt8, NetMinor UInt8, MobilePhone UInt8, SilverlightVersion1 UInt8, Age UInt8, Sex UInt8, Income UInt8, JavaEnable UInt8,
CookieEnable UInt8, JavascriptEnable UInt8, IsMobile UInt8, BrowserLanguage UInt16, BrowserCountry UInt16, Interests UInt16,
Robotness UInt8, GeneralInterests Array(UInt16), Params Array(String), Goals Nested(ID UInt32, Serial UInt32, EventTime
DateTime, Price Int64, OrderID String, CurrencyID UInt32), WatchIDs Array(UInt64), ParamSumPrice Int64, ParamCurrency
FixedString(3), ParamCurrencyID UInt16, ClickLogID UInt64, ClickEventID Int32, ClickGoodEvent Int32, ClickEventTime
DateTime, ClickPriorityID Int32, ClickPhraseID Int32, ClickPageID Int32, ClickPlacelD Int32, ClickTypeID Int32, ClickResourceID
Int32, ClickCost UInt32, ClickClientIP UInt32, ClickDomainID UInt32, ClickURL String, ClickAttempt UInt8, ClickOrderID UInt32,
ClickBannerID UInt32, ClickMarketCategoryID UInt32, ClickMarketPP UInt32, ClickMarketCategoryName String,
ClickMarketPPName String, ClickAWAPSCampaignName String, ClickPageName String, ClickTargetType UInt16,
ClickTargetPhraseID UInt64, ClickContextType UInt8, ClickSelectType Int8, ClickOptions String, ClickGroupBannerID Int32,
OpenstatServiceName String, OpenstatCampaignID String, OpenstatAdID String, OpenstatSourceID String, UTMSource String,
UTMMedium String, UTMCampaign String, UTMContent String, UTMTerm String, FromTag String, HasGCLID UInt8, FirstVisit
DateTime, PredLastVisit Date, LastVisit Date, TotalVisits UInt32, TrafficSource Nested(ID Int8, SearchEngineID UInt16,
AdvEngineID UInt8, PlacelD UInt16, SocialSourceNetworkID UInt8, Domain String, SearchPhrase String, SocialSourcePage String),
Attendance FixedString(16), CLID UInt32, YCLID UInt64, NormalizedReferrerHash UInt64, SearchPhraseHash UInt64,
ReferrerDomainHash UInt64, NormalizedStartURLHash UInt64, StartURLDomainHash UInt64, NormalizedEndURLHash UInt64,
TopLevelDomain UInt64, URLScheme UInt64, OpenstatServiceNameHash UInt64, OpenstatCampaignIDHash UInt64,
OpenstatAdIDHash UInt64, OpenstatSourceIDHash UInt64, UTMSourceHash UInt64, UTMMediumHash UInt64,
UTMCampaignHash UInt64, UTMContentHash UInt64, UTMTermHash UInt64, FromHash UInt64, WebVisorEnabled UInt8,
WebVisorActivity UInt32, ParsedParams Nested(Key1 String, Key2 String, Key3 String, Key4 String, Key5 String, ValueDouble
Float64), Market Nested(Type UInt8, GoalID UInt32, OrderID String, OrderPrice Int64, PP UInt32, DirectPlacelD UInt32,
DirectOrderID UInt32, DirectBannerID UInt32, GoodID String, GoodName String, GoodQuantity Int32, GoodPrice Int64), IslandID
FixedString(16)) ENGINE = CollapsingMergeTree(StartDate, intHash32(UserID), (CounterID, StartDate, intHash32(UserID), VisitID),
"(8192, Sign
import data ##
cat visits_v1.tsv | clickhouse-client --query "INSERT INTO datasets.visits_v1 FORMAT TSV" --max_insert_block_size=100000
optionally you can optimize table ##
"clickhouse-client --query "OPTIMIZE TABLE datasets.visits_v1 FINAL
"clickhouse-client --query "SELECT COUNT(*) FROM datasets.visits_v1
```

## Queries

Examples of queries to these tables (they are named **test.hits** and **test.visits**) can be found among **stateful tests** and **performance tests** of ClickHouse

## رابط ها

ClickHouse دو اینترفیس شبکه را فراهم می کند (هر دو می توانند به صورت اختیاری در TLS برای امنیت اضافی پیچیده شوند):

- **HTTP**, که مستند شده و به راحتی به طور مستقیم استفاده می شود.
- **بومی TCP**, که دارای سر بار کمتر است.

اگرچه در بیشتر موارد توصیه می شود از ابزار یا کتابخانه مناسب استفاده کنید تا به طور مستقیم با آن ها ارتباط برقرار نکنید. به طور رسمی توسط یانداکس پشتیبانی می شوند عبارتند از: \* **خط فرمان خط** \* **راننده JDBC** \* **راننده ODBC**

همچنین برای کار با ClickHouse طیف گسترده ای از کتابخانه های شخص ثالث وجود دارد: \* **کتابخانه های مشتری** \* **ادغام** \* **رابط های بصری**

# کلاينت Command-line

برای کار از طریق محیط ترمینال میتوانید از دستور `clickhouse-client` استفاده کنید

```
clickhouse-client $
.ClickHouse client version 0.0.26176
.Connecting to localhost:9000
.Connected to ClickHouse server version 0.0.26176
(:
```

کلاينت از آپشن های command-line و فایل های کانفیگ پشتیبانی می کند. برای اطلاعات بیشتر بخش "[پیکربندی](#)" را مشاهده کنید.

## استفاده

کلاينت می تواند به دو صورت `interactive` و `batch` (non-interactive) مورد استفاده قرار گیرد. برای استفاده از حالت `batch`، پارامتر `query` را مشخص کنید، و یا داده ها ره به `stdin` ارسال کنید (کلاينت تایید می کند که `stdin` ترمینال نیست) و یا از هر 2 استفاده کنید. مشابه `HTTP interface`، هنگامی که از از پارامتر `query` و ارسال داده ها به `stdin` استفاده می کنید، درخواست، ترکیبی از پارامتر `query`، `line feed`، و داده ها در `stdin` است. این کار برای `query` های بزرگ `INSERT` مناسب است.

مثالی از استفاده کلاينت برای اجرای دستور `INSERT` داده

```
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"
'some text', '2016-08-14 00:00:00', 3
'some more text', '2016-08-14 00:00:01', 4
_EOF_

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

در حالت `Batch`، فرمت داده ها به صورت پیش فرض به صورت `TabSeparated` می باشد. شما میتوانید فرمت داده ها رو در هنگام اجرای `query` و با استفاده از شرط `FORMAT` مشخص کنید.

به طور پیش فرض شما فقط می توانید یک `query` را در حالت `batch` اجرا کنید. برای ساخت چندین `query` از یک "اسکریپت"، از پارامتر `--multiline` استفاده کنید. این روش برای تمام `query` ها به جز `INSERT` کار می کند. نتایج `query` ها به صورت متوالی و بدون `separator` اضافه تولید می شوند. به طور مشابه برای پردازش تعداد زیادی از `query` ها شما می توانید از 'clickhouse-client' برای هر `query` استفاده کنید. دقت کنید که ممکن است حدود 10 میلی ثانیه تا زمان راه اندازی برنامه 'clickhouse-client' زمان گرفته شود.

در حالت `interactive`، شما یک `command line` برای درج `query` های خود دریافت می کنید.

اگر 'multiline' مشخص نشده باشد (به صورت پیش فرض): برای اجرای یک `query`، دکمه `Enter` را بزنید. سیمی کالن در انتهای `query` اجباری نیست. برای درج یک `query` چند خطی (multiline)، دکمه `Y` یا `Backslash` را قبل از `line feed` فشار دهید. بعد از فشردن `Enter`، شما برای درج خط بعدی `query` درخواست خواهید شد.

اگر چند خطی (multiline) مشخص شده باشد: برای اجرای `query`، در انتها سیمی کالن را وارد کنید و سپس `Enter` بزنید. اگر سیمی کالن از انتهای خط حذف می شد، از شما برای درج خط جدید `query` درخواست می شد.

تنها یک `query` اجرا می شود. پس همه چیز بعد از سیمی کالن `ignore` می شود.

شما میتوانید از `G` به جای سیمی کالن یا بعد از سیمی کالن استفاده کنید. این علامت، فرمت `Vertical` را نشان می دهد. در این فرمت، هر مقدار در یک خط جدا چاپ می شود که برای جداول عریض مناسب است. این ویژگی غیرمعمول برای سازگاری با `MySQL CLI` اضافه شد.

`command line` بر پایه 'readline' (و 'history' یا 'libedit'، به بدون کتابخانه بسته به build) می باشد. به عبارت دیگر، این محیط از `shortcut` های آشنا استفاده می کند و `history` دستورات را نگه می دارد. `history` ها در فایل `clickhouse-client-history./~` نوشته می شوند.

به صورت پیش فرض فرمت خروجی `PrettyCompact` می باشد. شما میتوانید از طریق دستور `FORMAT` در یک `query`، یا با مشخص کردن `G` در انتهای `query`، استفاده از آرگومان های `format--` یا `vertical--` یا از کانفیگ فایل کلاينت، فرمت خروجی را مشخص کنید.

برای خروج از کلاینت، Ctrl-D (یا Ctrl+C) را فشار دهید؛ و یا یکی از دستورات زیر را به جای اجرای query اجرا کنید: "quit", "exit", "logout", "exit;", "quit;", "logout;", "q", "Q", ":q"

در هنگام اجرای یک query، کلاینت موارد زیر را نمایش می دهد:

1. Progress، که بیش از 10 بار در ثانیه بروز نخواهد شد ( به صورت پیش فرض). برای query های سریع، progress ممکن است زمانی برای نمایش پیدا نکند.
2. فرمت کردن query بعد از عملیات پارس کردن، به منظور دیدار کردن query.
3. نمایش خروجی با توجه به نوع فرمت.
4. تعداد لاین های خروجی، زمان پاس شدن query، و میانگیم سرعت پردازش query.

شما میتوانید query های طولانی را با فشردن Ctrl-C کنسل کنید. هر چند، بعد از این کار همچنان نیاز به انتظار چند ثانیه ای برای قطع کردن درخواست توسط سرور می باشید. امکان کنسل کردن یک query در مراحل خاص وجود ندارد. اگر شما صبر نکنید و برای بار دوم Ctrl+C را وارد کنید از client خارج می شوید.

کلاینت commant-line اجازه ی پاس دادن داده های external (جداول موقت external) را برای query ها می دهد. برای اطلاعات بیشتر به بخش "داده های External برای پردازش query" مراجعه کنید.

## پیکربندی

شما میتوانید، پارامتر ها را به **clickhouse-client** (تمام پارامترها دارای مقدار پیش فرض هستند) از دو روش زیر پاس بدید:

از طریق Command Line

گزینه های Command-line مقادیر پیش فرض در ستینگ و کانفیگ فایل را نادیده میگیرد.

کانفیگ فایل ها.

ستینگ های داخل کانفیگ فایل، مقادیر پیش فرض را نادیده می گیرد.

## گزینه های Command line

- **host, -h--** نام سرور، به صورت پیش فرض 'localhost' است. شما میتوانید یکی از موارد نام و یا IPv4 و یا IPv6 را در این گزینه مشخص کنید.
- **port--** پورت اتصال به ClickHouse. مقدار پیش فرض: 9000. دقت کنید که پرت اینترفیس HTTP و اینترفیس native متفاوت است.
- **user, -u--** نام کاربری جهت اتصال. پیش فرض: default.
- **password--** پسورد جهت اتصال. پیش فرض: خالی
- **query, -q--** مشخص کردن query برای پردازش در هنگام استفاده از حالت non-interactive.
- **database, -d--** انتخاب دیتابیس در بدو ورود به کلاینت. مقدار پیش فرض: دیتابیس مشخص شده در تنظیمات سرور (پیش فرض 'default')
- **multiline, -m--** اگر مشخص شود، یعنی اجازه ی نوشتن query های چند خطی را بده. (بعد از Enter، query را ارسال نکن).
- **multiquery, -n--** اگر مشخص شود، اجازه ی اجرای چندین query که از طریق کاما جدا شده اند را می دهد. فقط در حالت non-interactive کار می کند.
- **format, -f--** مشخص کردن نوع فرمت خروجی
- **vertical, -E--** اگر مشخص شود، از فرمت Vertical برای نمایش خروجی استفاده می شود. این گزینه مشابه 'format=Vertical--' می باشد. در این فرمت، هر مقدار در یک خط جدید چاپ می شود، که در هنگام نمایش جداول عریض مفید است.
- **time, -t--** اگر مشخص شود، در حالت non-interactive زمان اجرای query در 'stderr' چاپ می شود.
- **stacktrace--** اگر مشخص شود stack trase مربوط به اجرای query در هنگام رخ دادن یک exception چاپ می شود.
- **config-file-** نام فایل پیکربندی.

## فایل های پیکربندی

**clickhouse-client** به ترتیب اولویت زیر از اولین فایل موجود برای ست کردن تنظیمات استفاده می کند:

- **config-file-** مشخص شده در پارامتر
- **clickhouse-client.xml/.**
- **clickhouse-client/config.xml./~\**
- **etc/clickhouse-client/config.xml/**

مثالی از یک کانفیگ فایل

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

## رابط بومی (TCP)

پروتکل بومی در [خط فرمان خط] (cli.md)، برای برقراری ارتباط بین سرور در طی پردازش پرس و جو توزیع شده، و همچنین در سایر برنامه های ++ C استفاده می شود. متأسفانه، پروتکل ClickHouse بومی هنوز مشخصات رسمی ندارد، اما می توان آن را از کد منبع ClickHouse (شروع از اینجا) و / یا با رهگیری و تجزیه و تحلیل ترافیک TCP.

## HTTP interface

HTTP interface به شما امکان استفاده از ClickHouse در هر پلتفرم با هر زمان برنامه نویسی را می دهد. ما از این Interface برای زبان های Java و Perl به مانند shell استفاده می کنیم. در دیگر دپارتمان ها، HTTP interface در Python، Perl، و Go استفاده می شود. HTTP Interface محدود تر از native interface می باشد، اما سازگاری بهتری دارد.

به صورت پیش فرض، clickhouse-server به پرت 8123 در HTTP گوش می دهد. (میتونه در کانفیگ فایل تغییر پیدا کنه). اگر شما یک درخواست GET / بدون پارامتر بسازید، رشته ی "OK" رو دریافت می کنید (به همراه line feed در انتها). شما می توانید از این درخواست برای اسکرپیت های health-check استفاده کنید.

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

درخواست های خود را پارامتر 'query'، یا با متد POST، یا ابتدای query را در پارامتر 'query' ارسال کنید، و بقیه را در POST (بعدا توضیح خواهیم داد که چرا این کار ضروری است). سایت URL محدود به 16 کیلوبایت است، پس هنگام ارسال query های بزرگ، اینو به خاطر داشته باشید

اگر درخواست موفق آمیز باشد، استاتوس کد 200 را دریافت می کنید و نتایج در بدنه response می باشد. اگر اروری رخ دهد، استاتوس کد 500 را دریافت می کنید و توضیحات ارور در بدنه ی reponse قرار می گیرد.

در هنگام استفاده از متد 'readonly'، 'GET' ست می شود. به عبارت دیگر، برای query هایی که قصد تغییر دیتا را دارند، شما فقط از طریق متد POST می توانید این تغییرات را انجام دهید. شما میتونید query رو در بدنه ی POST یا به عنوان پارامتر های URL ارسال کنید.

مثال:

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

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

echo -ne 'GET /?query=SELECT%201 HTTP/1.0\r\n\r\n' | nc localhost 8123 $
HTTP/1.0 200 OK
Connection: Close
Date: Fri, 16 Nov 2012 19:21:50 GMT

1
```

همانطور که می بینید، curl is somewhat inconvenient in that spaces must be URL escaped، هر چند wget همه چیز را خودش escape می کنه، ما توصیه به استفاده از اون رو نمی کنیم، چون wget به خوبی با HTTP 1.1 در هنگام استفاده از هدر های keep-alive و Transfer-Encoding: chunked کار نمی کند.

```
-@ 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
```

اگر بخشی از query در پارامتر ارسال شود، و بخش دیگر در POST، یک line feed بین دو بخش وارد می شود. مثال (این کار نمی کند):

```
-@ 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
```

به صورت پیش فرض، داده ها با فرمت TabSeparated بر میگردند. (برای اطلاعات بیشتر بخش "فرمت" را مشاهده کنید). شما میتوانید از دستور FORMAT در query خود برای ست کردن فرمتی دیگر استفاده کنید.

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

```
┌───┐
│ 1 │
├───┤
│ 1 │
└───┘
```

برای query های INSERT متد POST ضروری است. در این مورد، شما می توانید ابتدای query خود را در URL parameter بنویسید، و از POST برای پاس داده ها برای درج استفاده کنید. داده ی برای درج می تواند، برای مثال یک دامپ tab-separated شده از MySQL باشد. به این ترتیب، query INSERT جایگزین LOAD DATA LOCAL INFILE از MySQL می شود.

مثال: ساخت جدول

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

استفاده از query INSERT برای درج داده:

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

داده ها میتوانند جدا از پارامتر query ارسال شوند:

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

شما می توانید هر نوع فرمت دیتایی مشخص کنید. فرمت 'Values' دقیقاً مشابه زمانی است که شما INSERT INTO t VALUES را می نویسید:

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

برای درج داده ها از یک دامپ tab-separate، فرمت مشخص زیر را وارد کنید:

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

به دلیل پردازش موازی، نتایج query با ترتیب زردوم چاپ می شود:

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

```
7
8
9
10
11
12
1
2
3
4
5
6
```

حذف جدول:

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

برای درخواست هایی موفق که داده ای از جدول بر نمیگردد، بدنه response خالی است.

شما می توانید از فرمت فشرده سازی داخلی ClickHouse در هنگام انتقال داده ها استفاده کنید. این فشرده سازی داده، یک فرمت غیراستاندارد است، و شما باید از برنامه مخصوص فشرده سازی ClickHouse برای استفاده از آن استفاده کنید. (این برنامه در هنگام نصب پکیج clickhouse-client نصب شده است)

اگر شما در URL پارامتر 'compress=1' را قرار دهید، سرور داده های ارسالی به شما را فشرده سازی می کند. اگر شما پارامتر 'decompress=1' را در URL ست کنید، سرور داده های ارسالی توسط متد POST را decompress می کند.

همچنین استفاده از فشرده سازی استاندارد gzip در HTTP ممکن است. برای ارسال درخواست POST و فشرده سازی آن به صورت gzip، هدر **Content-Encoding: gzip** را به request خود اضافه کنید. برای اینکه response، ClickHouse فشرده شده به صورت gzip برای شما ارسال کند، ابتدا باید **enable\_http\_compression** را در تنظیمات ClickHouse فعال کنید و در ادامه هدر **Accept-Encoding: gzip** را به درخواست خود اضافه کنید.

شما می توانید از این کار برای کاهش ترافیک شبکه هنگام ارسال مقدار زیادی از داده ها یا برای ایجاد dump هایی که بلافاصله فشرده می شوند، استفاده کنید.

شما می توانید از پارامتر 'database' در URL برای مشخص کردن دیتابیس پیش فرض استفاده کنید.

```
-$ 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
```

به صورت پیش فرض، دیتابیس ثبت شده در تنظیمات سرور به عنوان دیتابیس پیش فرض مورد استفاده قرار می گیرد، این دیتابیس 'default' نامیده می شود. از سوی دیگر، شما می توانید همیشه نام دیتابیس را با دات و قبل از اسم جدول مشخص کنید.

نام کاربری و پسورد می توانند به یکی از دو روش زیر ست شوند:

1. استفاده از HTTP Basic Authentication. مثال:

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

2. با دو پارامتر 'user' و 'password' در URL. مثال:

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

اگر نام کاربری مشخص نشود، نام کاربری 'default' استفاده می شود. اگر پسورد مشخص نشود، پسورد خالی استفاده می شود. شما همچنین می توانید از پارامتر های URL برای مشخص کردن هر تنظیمی برای اجرای یک query استفاده کنید. مثال: [http://localhost:8123/?profile=web&max\\_rows\\_to\\_read=1000000000&query=SELECT+1](http://localhost:8123/?profile=web&max_rows_to_read=1000000000&query=SELECT+1)

برای اطلاعات بیشتر بخش "تنظیمات" را مشاهده کنید.

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

برای اطلاعات بیشتر در مورد دیگر پارامترها، بخش "SET" را ببینید.

به طور مشابه، شما می توانید از ClickHouse Session در HTTP استفاده کنید. برای این کار، شما نیاز به ست کردن پارامتر `session_id` برای درخواست را دارید. شما می توانید از هر رشته ای برای ست کردن Session ID استفاده کنید. به صورت پیش فرض، یک session بعد از 60 ثانیه از اجرای آخرین درخواست نابود می شود. برای تغییر زمان timeout، باید `default_session_timeout` را در تنظیمات سرور تغییر دهید و یا پارامتر `session_timeout` در هنگام درخواست ست کنید. برای بررسی وضعیت status از پارامتر `session_check=1` استفاده کنید. با یک session تنها یک query در لحظه می توان اجرا کرد.

گزینه ای برای دریافت اطلاعات progress اجرای query وجود دارد. این گزینه هدر X-ClickHouse-Progress می باشد. برای این کار تنظیم `send_progress_in_http_headers` در کانفیگ فایل فعال کنید.

درخواست در حال اجرا، در صورت لاست شدن کانکشن HTTP به صورت اتوماتیک متوقف نمی شود. پارس کردن و فرمت کردن داده ها در سمت سرور صورت می گیرد، و استفاده از شبکه ممکن است ناکارآمد باشد. پارامتر 'query\_id' می تونه به عنوان query id پاس داده شود (هر رشته ای قابل قبول است). برای اطلاعات بیشتر بخش "تنظیمات، `replace_running_query`" را مشاهده کنید.

پارامتر 'quote\_key' می تواند به عنوان quote key پاس داده شود (هر رشته ای قابل قبول است). برای اطلاعات بیشتر بخش "Quotas" را مشاهده کنید.

HTTP interface اجازه ی پاس دادن داده های external (جداول موقت external) به query را می دهد. برای اطلاعات بیشتر، بخش "داده های External برای پردازش query" را مشاهده کنید.

## بافرینگ Response

شما می توانید در سمت سرور قابلی بافرینگ response را فعال کنید. پارامترهای `buffer_size` و `wait_end_of_query` در URL برای این هدف ارائه شده اند.

`buffer_size` تعیین کننده ی اندازه ی نتیجه (بایت) برای بافر شدن در حافظه سرور است. اگر بدنه ی نتیجه بیش بزرگتر از این threshold باشد، بافر بر روی HTTP channel نوشته می شود و باقی مانده ی دیتا به صورت مستقیم به HTTP Channel ارسال می شود.

برای اینکه مطمئن بشید که کل response بافر شده است، `wait_end_of_query=1` را ست کنید. در این مورد، داده های که در حافظه ذخیره نمی شوند، در یک فایل موقت سمت سرور بافر می شوند.

مثال:

```
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'
```

از بافرینگ به منظور اجتناب از شرایطی که یک خطای پردازش query رخ داده بعد از response کد و هدر های ارسال شده به کلاینت استفاده کنید. در این شرایط، پیغام خطا در انتهای بنده response نوشته می شود، و در سمت کلاینت، پیغام خطا فقط از طریق مرحله پارس کردن قابل شناسایی است.

## فرمت های Input و Output



فرمت تعیین می کند که چگونه داده ها پس از اجرای SELECT (چگونه نوشته شده و چگونه توسط سرور فرمت شده) به شما بر می گردد، و چگونه آن برای INSERT ها پذیرفته شده (چگونه آن توسط سرور پارس و خوانده می شود).

جدول زیر لیست فرمت های پشتیبانی شده برای هر نوع از query ها را نمایش می دهد.

Format | INSERT | SELECT

## درایور JDBC

درایور رسمی JDBC برای ClickHouse وجود دارد. برای اطلاعات بیشتر [اینجا](#) را ببینید.

درایور JDBC توسط سازمان های دیگر اجرا می شوند.

[ClickHouse-Native-JDBC](#)

•

## درایور ODBC

درایور رسمی ODBC برای ClickHouse وجود دارد. برای اطلاعات بیشتر [اینجا](#) را ببینید.

## کتابخانه های مشتری شخص ثالث

سلب مسئولیت

Yandex نه حفظ کتابخانه ها در زیر ذکر شده و نشده انجام هر آزمایش های گسترده ای برای اطمینان از کیفیت آنها.

Python

•

[infi.clickhouse\\_orm](#)

◦

[clickhouse-driver](#)

◦

[clickhouse-client](#)

◦

PHP

•

[phpClickHouse](#)

◦

[clickhouse-php-client](#)

◦

[clickhouse-client](#)

◦

[PhpClickHouseClient](#)

◦

Go

•

[clickhouse](#)

◦

[go-clickhouse](#)

◦

[mailrugo-clickhouse](#)

◦

[golang-clickhouse](#)

◦

NodeJs

•

[\(clickhouse \(Nodejs](#)

◦

[node-clickhouse](#)

◦

Perl

•

[perl-DBD-ClickHouse](#)

◦

[HTTP-ClickHouse](#)

◦

[AnyEvent-ClickHouse](#)

◦

Ruby

•

[\(clickhouse \(Ruby](#)

◦

R

•

[clickhouse-r](#)

◦

[RClickhouse](#)

◦

Java

•

[clickhouse-client-java](#)

◦

Scala	•
clickhouse-scala-client	◦
Kotlin	•
AORM	◦
#C	•
ClickHouse.Ado	◦
ClickHouse.Net	◦
++C	•
clickhouse-cpp	◦
Elixir	•
clickhouseex	◦
Nim	•
nim-clickhouse	◦

ما این کتابخانه ها را تست نکردیم. آنها به صورت تصادفی انتخاب شده اند.

## کتابخانه ادغام ثالث

سلب مسئولیت

Yandex نه حفظ کتابخانه ها در زیر ذکر شده و نشده انجام هر آزمایش های گسترده ای برای اطمینان از کیفیت آنها.

## محصولات زیربنایی

سیستم های مدیریت پایگاه داده رابطه ای	•
MySQL	◦
ProxySQL	■
clickhouse-mysql-data-reader	■
horgh-replicator	■
PostgreSQL	◦
clickhousedb_fdw	■
(infi.clickhouse_orm استفاده می کند infi.clickhouse_fdw)	■
pg2ch	■
MSSQL	◦
ClickHouseMightrator	■
صف پیام	•
Kafka	◦
(Go client استفاده می کند clickhouse_sinker)	■
فروشگاه شی	•
S3	◦
clickhouse-backup	■
ارکستراسیون کانتینر	•
Kubernetes	◦
clickhouse-operator	■
مدیریت تنظیمات	•
puppet	◦
innogames/clickhouse	■
mfedotov/clickhouse	■

نظارت بر

Graphite

graphouse

carbon-clickhouse

Grafana

clickhouse-grafana

Prometheus

clickhouse\_exporter

PromHouse

Nagios

check\_clickhouse

Zabbix

clickhouse-zabbix-template

Sematest

clickhouse ادغام

ثبت نام

rsyslog

omclickhouse

fluentd

loghouse (برای Kubernetes)

logagent

logagent output-plugin-clickhouse

جغرافیایی

MaxMind

clickhouse-maxmind-geoip

## اکوسیستم زبان برنامه نویسی

Python

SQLAlchemy

sqlalchemy-clickhouse (استفاده می کند infi.clickhouse\_orm)

pandas

pandahouse

R

dplyr

RClickhouse (استفاده می کند clickhouse-cpp)

Java

Hadoop

clickhouse-hdfs-loader (استفاده می کند JDBC)

Scala

Akka

clickhouse-scala-client

#C

ADO.NET

ClickHouse.Ado

ClickHouse.Net

ClickHouse.Net.Migrations

Elixir

Ecto

clickhouse\_ecto

## third-party visual interface های توسعه دهندگان

متن باز

## Tabix

interface تحت وب برای ClickHouse در پروژه **Tabix**.

ویژگی ها:

- کار با ClickHouse به صورت مستقیم و از طریق مرورگر، بدون نیاز به نرم افزار اضافی.
- ادیتور query به همراه syntax highlighting.
- ویژگی Auto-completion برای دستورات.
- ابزارهایی برای آنالیز گرافیکی اجرای query.
- گزینه های Color scheme.

**مستندات Tabix**

## HouseOps

**HouseOps** نرم افزار Desktop برای سیستم عامل های Linux و OSX و Windows می باشد..

ویژگی ها:

- ابزار Query builder به همراه syntax highlighting. نمایش نتایج به صورت جدول و JSON Object.
- خروجی نتایج به صورت csv و JSON Object.
- P نمایش Processes List ها به همراه توضیحات، ویژگی حالت record و kill کردن process ها.
- نمودار دیتابیس به همراه تمام جداول و ستون ها به همراه اطلاعات اضافی.
- نمایش آسان ساینر ستون ها.
- تنظیمات سرور.
- مدیریت دیتابیس (بزودی);
- مدیریت کاربران (بزودی);
- آنالیز داده ها به صورت Real-Time (بزودی);
- مانیتورینگ کلاستر/زیرساخت (بزودی);
- مدیریت کلاستر (بزودی);
- مانیتورینگ کافکا و جداول replicate (بزودی);
- و بسیاری از ویژگی های دیگر برای شما.

## LightHouse

**LightHouse** رابط کاربری سبک وزن برای ClickHouse است.

امکانات:

- لیست جدول با فیلتر کردن و ابر داده.
- پیش نمایش جدول با فیلتر کردن و مرتب سازی.
- اعداد نمایش داده شده فقط خواندنی

## DBeaver

**DBeaver** - مشتری دسکتاپ دسکتاپ دسکتاپ با پشتیبانی ClickHouse.

امکانات:

- توسعه پرس و جو با برجسته نحو
- پیش نمایش جدول
- تکمیل خودکار

## clickhouse-cli

**clickhouse-cli** یک مشتری خط فرمان برای ClickHouse است که در پایتون 3 نوشته شده است.

امکانات:

- تکمیل خودکار
- نحو برجسته برای نمایش داده ها و خروجی داده ها.
- پشتیبانی از Pager برای خروجی داده.
- دستورات پست سفارشی مانند PostgreSQL.

# تجاری

## DataGrip

**DataGrip** IDE پایگاه داده از JetBrains با پشتیبانی اختصاصی برای ClickHouse است. این ابزار همچنین به سایر ابزارهای مبتنی بر IntelliJ تعیبه شده است: PyCharm, IntelliJ IDEA, GoLand, PhpStorm و دیگران.

امکانات:

- تکمیل کد بسیار سریع
- نحو برجسته ClickHouse.
- پشتیبانی از ویژگی های خاص برای ClickHouse، برای مثال ستون های تویی، موتورهای جدول.
- ویرایشگر داده.
- Refactorings.
- جستجو و ناوبری

## سرورهای پروکسی از توسعه دهندگان شخص ثالث

**chproxy**، یک پراکسی HTTP و تعادل بار برای پایگاه داده ClickHouse است.

امکانات

- مسیریابی و پاسخ دهی کاربر به کاربر.
- محدودیت انعطاف پذیر
- تمديد SSL certificate به صورت خودکار.

اجرا شده در برو

## KittenHouse

**KittenHouse** طراحی شده است که یک پروکسی محلی بین ClickHouse و سرور برنامه باشد در صورتی که غیر ممکن است یا ناخوشایند بافر کردن اطلاعات INSERT در قسمت درخواست شما.

امکانات:

- بافر حافظه در حافظه و درایو.
- مسیریابی در جدول
- تعادل بار و بررسی سلامت.

اجرا شده در برو

## ClickHouse-Bulk

**ClickHouse-Bulk** یک ClickHouse جمع کننده ساده است.

امکانات:

- درخواست گروهی و ارسال توسط آستانه یا فاصله.
- چند سرور از راه دور
- احراز هویت پایه

اجرا شده در برو

## Data types

ClickHouse قابلیت ذخیره سازی انواع type های مختلف برای ذخیره داده ها در جداول را دارا می باشد.

این بخش انواع data type های قابل پشتیبانی در ClickHouse را شرح می دهد، همچنین ملاحظات آنها در هنگام استفاده آنها را شرح می دهد.

# UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64

اعداد با طول مشخص (Fixed-length) با یا بدون sign

## Int محدوده ی

[Int8 - [-128 : 127

[Int16 - [-32768 : 32767

[Int32 - [-2147483648 : 2147483647

[Int64 - [-9223372036854775808 : 9223372036854775807

- 
- 
- 
- 

## UInt محدوده ی

[UInt8 - [0 : 255

[UInt16 - [0 : 65535

[UInt32 - [0 : 4294967295

[UInt64 - [0 : 18446744073709551615

- 
- 
- 
- 

# Float32, Float64

اعداد Float.

Type های float در ClickHouse مشابه C می باشد:

Float32 - float

Float64 - double

- 
- 

توصیه می کنیم که داده ها را هرزمان که امکان پذیره است به جای float به صورت int ذخیره کنید. برای مثال: تبدیل دقت اعداد به یک مقدار int، مثل سرعت page load در قالب میلی ثانیه.

## استفاده از اعداد Float

محاسبات با اعداد با Float ممکن است خطای round شدن را ایجاد کنند.

- 

```
SELECT 1 - 0.9
```

```
┌-(minus(1, 0.9)──┐
│ 0.09999999999999998 │
└────────────────┘
```

- نتایج محاسبات بسته به متد محاسباتی می باشد (نوع processor و معماری سیستم).
- محاسبات Float ممکن اسن نتایجی مثل infinity (inf) و "NaN" (Not-a-number) داشته باشد. این در هنگام پردازش نتایج محاسبات باید مورد توجه قرار گیرد.
- هنگام خواندن اعداد float از سطر ها، نتایج ممکن است نزدیک به اعداد machine-representable نباشد.

## Inf و NaN

در مقابل استاندارد SQL، ClickHouse موارد زیر مربوط به اعداد float پشتیبانی می کند:

Inf - Infinity

- 

```
SELECT 0.5 / 0
```

```
┌-(divide(0.5, 0)──┐
│ inf              │
└────────────────┘
```

-Inf - Negative infinity

-

```
SELECT -0.5 / 0
```

```
┌--(divide(-0.5, 0--r
| inf-          |
└──────────┘
```

.NaN - Not a number

```
SELECT 0 / 0
```

```
┌--(divide(0, 0--r
| nan          |
└──────────┘
```

قوانین مربوط به مرتب سازی Nan را در بخش ORDER BY clause ببینید.

## Decimal(P, S), Decimal32(S), Decimal64(S), (Decimal128(S

Signed fixed point numbers that keep precision during add, subtract and multiply operations. For division least .(significant digits are discarded (not rounded

### Parameters

- P - precision. Valid range: [ 1 : 38 ]. Determines how many decimal digits number can have (including .(fraction
- S - scale. Valid range: [ 0 : P ]. Determines how many decimal digits fraction can have

:Depending on P parameter value Decimal(P, S) is a synonym for

(P from [ 1 : 9 ] - for Decimal32(S -

(P from [ 10 : 18 ] - for Decimal64(S -

(P from [ 19 : 38 ] - for Decimal128(S -

### Decimal value ranges

- ( (Decimal32(S) - ( -1 \* 10^(9 - S), 1 \* 10^(9 - S
- ( (Decimal64(S) - ( -1 \* 10^(18 - S), 1 \* 10^(18 - S
- ( (Decimal128(S) - ( -1 \* 10^(38 - S), 1 \* 10^(38 - S

.For example, Decimal32(4) can contain numbers from -99999.9999 to 99999.9999 with 0.0001 step

### Internal representation

Internally data is represented as normal signed integers with respective bit width. Real value ranges that can be .stored in memory are a bit larger than specified above, which are checked only on conversion from string

Because modern CPU's do not support 128 bit integers natively, operations on Decimal128 are emulated. Because .of this Decimal128 works significantly slower than Decimal32/Decimal64

### Operations and result type

.(Binary operations on Decimal result in wider result type (with any order of arguments

- (Decimal64(S1) Decimal32(S2) -> Decimal64(S
- (Decimal128(S1) Decimal32(S2) -> Decimal128(S
- (Decimal128(S1) Decimal64(S2) -> Decimal128(S

:Rules for scale

- .(add, subtract: S = max(S1, S2
- .multiply: S = S1 + S2
- .divide: S = S1

Operations between Decimal and Float32/Float64 are not defined. If you really need them, you can explicitly cast one of argument using toDecimal32, toDecimal64, toDecimal128 or toFloat32, toFloat64 builtins. Keep in mind .that the result will lose precision and type conversion is computationally expensive operation

Some functions on Decimal return result as Float64 (for example, var or stddev). Intermediate calculations might still be performed in Decimal, which might lead to different results between Float64 and Decimal inputs with same .values

## Overflow checks

During calculations on Decimal, integer overflows might happen. Excessive digits in fraction are discarded (not rounded). Excessive digits in integer part will lead to exception

```
SELECT toDecimal32(2, 4) AS x, x / 3
```

-(x—divide(toDecimal32(2, 4), 3——r  
| 0.6666 | 2.0000 |

```
SELECT toDecimal32(4.2, 8) AS x, x * x
```

```
.DB::Exception: Scale is out of bounds
```

```
SELECT toDecimal32(4.2, 8) AS x, 6 * x
```

```
.DB::Exception: Decimal math overflow
```

Overflow checks lead to operations slowdown. If it is known that overflows are not possible, it makes sense to disable checks using `decimal_check_overflow` setting. When checks are disabled and overflow happens, the result will be incorrect

```
;SET decimal_check_overflow = 0
SELECT toDecimal32(4.2, 8) AS x, 6 * x
```

17.74967296-	4.20000000
--------------	------------

:Overflow checks happen not only on arithmetic operations, but also on value comparison

```
SELECT toDecimal32(1, 8) < 100
```

```
.DB::Exception: Can't compare
```

## Boolean مقادير

type مخصوص مقادیر boolean وجود ندارد. از Uint8 و محدود شده به 0 و 1 می توان استفاده کرد.

# String

String یک type برای قرار دادن رشته با طول دلخواه می باشد. این طول محدود نمی باشد. این مقدار می تواند شامل مجموعه ای دلخواه از بایت ها، از جمله null byte باشد. String type جایگزین type های VARCHAR, BLOB, CLOB و ... دیگر DBMS ها می باشد.

## Encodings



ClickHouse مفهومی به نام encoding ندارد. String ها می توانند شامل مجموعه ای بایت ها باشند که با همان شکل که نوشته می شوند به همان شکل هم در خروجی دیده شوند. اگر شما نیاز به ذخیره سازی متن دارید، توصیه می کنیم از UTF-8 استفاده کنید. حداقل اگر ترمینال شما از UTF-8 (پیشنهاد شده)، استفاده می کند، شما می توانید به راحتی مقادیر خود را نوشته و بخوانید. به طور مشابه توابع خاصی برای کار با رشته های متنوع وجود دارند که تخیل این فرضیه عمل می کنند که رشته شامل مجوعه ای از بایت ها می باشند که نماینده ی متن های UTF-8 هستند. برای مثال تابع 'length' برای محاسبه طول رشته براساس بایت است، در حالی که تابع 'lengthUTF8' برای محاسبه طول رشته بر اساس UNICODE می باشد.

## (FixedString(N

یک رشته با طول ثابت N بایت (fixed-length) (نه تعداد کاراکتر یا N code point) باید یک عدد طبیعی مثبت باشد. هنگامی که سرور رشته ای با اندازه ی کمتر از N میخواند (مثل زمان پارس کردن برای INSERT داده ها)، سمت راست رشته به اندازه فضای خالی باقی مانده به بایت، null درج می شود. زمانی که سرور رشته ای بزرگتر از N میخواند، پیغام خطا بر میگردد. زمانی که سرور یک رشته با طول ثابت را می نویسد (مثلا در زمانی که خروجی یک SELECT را برمیگرداند)، بایت های null از انتهای رشته حذف نمی شوند، و در خروجی می آیند. توجه داشته باشید که این رفتار متفاوت از رفتار MySQL برای Char می باشد (زمانی که رشته با space پر می شود و در خروجی space ها حذف می شود).

توابع کمتری نسبت به String برای FixedString(N وجود دارد، و برای استفاده کمتر مناسب است.

## UUID

A universally unique identifier (UUID) is a 16-byte number used to identify records. For detailed information about the UUID, see [Wikipedia](#)

The example of UUID type value is represented below

```
61f0c404-5cb3-11e7-907b-a6006ad3dba0
```

If you do not specify the UUID column value when inserting a new record, the UUID value is filled with zero

```
00000000-0000-0000-0000-000000000000
```

## How to generate

To generate the UUID value, ClickHouse provides the [generateUUIDv4](#) function

## Usage example

### Example 1

This example demonstrates creating a table with the UUID type column and inserting a value into the table

```
CREATE TABLE t_uuid (x UUID, y String) ENGINE=TinyLog (:

INSERT INTO t_uuid SELECT generateUUIDv4(), 'Example 1 (:

SELECT * FROM t_uuid (:

+-----+-----+
| x      | y      |
+-----+-----+
| 417ddc5d-e556-4d27-95dd-a34d84e46a50 | Example 1 |
+-----+-----+
```

### Example 2

In this example, the UUID column value is not specified when inserting a new record

```
('INSERT INTO t_uuid (y) VALUES ('Example 2 (:
```

```
SELECT * FROM t_uuid (:
```

x   y	
417ddc5d-e556-4d27-95dd-a34d84e46a50	Example 1
Example 2	00000000-0000-0000-0000-000000000000

## Restrictions

The UUID data type only supports functions which **String** data type also supports (for example, **min**, **max**, and **count**).

The UUID data type is not supported by arithmetic operations (for example, **abs**) or aggregate functions, such as **sum** and **avg**.

## Date

**Date**, دو بایت به ازای هر تاریخ که به صورت عددی و از تاریخ 1970-01-01 می باشد ذخیره می کند (unsigned). این type به شما اجازه ی ذخیره سازی تاریخ های از ابتدای Unix Epoch تا بالاترین مقدار قابل پشتیبانی توسط این استاندارد را می دهد (در حال حاضر بالاترین مقدار این روش سال 2106 می باشد، اما سال آخری که به طور کامل پشتیبانی می شود سال 2105 است). کمترین مقدار این type در خروجی 00-00-0000 می باشد.

**Date** بدون time zone ذخیره می شود.

## DateTime

تاریخ با ساعت 4 بایت به صورت Unix timestamp ذخیره می کند (unsigned). به شما اجازه ی ذخیره سازی در محدوده ی تایپ **Date** را می دهد. حداقل مقدار در خروجی 00:00:00 00-00-0000 می باشد. زمان با دقت تا یک ثانیه ذخیره می شود.

## Time zones

این type از **text** به باینری تبدیل می شود، و در هنگام برگشت با توجه به time zone سرور، در زمانی که کلاینت یا سرور شروع به کار می کند تبدیل می شود. در فرمت **text**، اطلاعات DST از دست می رود.

به صورت پیش فرض زمانی که کلاینت به سرور کانکت می شود، timezone آن به تنظیم سرور تغییر می کند. شما می توانید این رفتار را با فعال سازی گزینه **use\_client\_time\_zone--** در محیط command-line تغییر دهید.

پس در هنگام کار با تاریخ متنی (برای مثال زمانی که دامپ **text** می گیرید)، به خاطر داشته باشید که ممکن است به دلیل تغییرات DST، نتایج مبهمی در خروجی ببینید، و ممکن است در صورت تغییر time zone مشکلی با مطابقت خروجی با داده ها وجود داشته باشد.

## Enum

**Enum8** یا **Enum16**، به شما اجازه ی ذخیره سازی مجموعه ای محدود از رشته ها را می دهد که کارآمدتر از **String** می باشد.

مثال:

```
(Enum8('hello' = 1, 'world' = 2
```

مقدار داخل این ستون می تواند یکی از دو مقدار 'hello' یا 'world' باشد.

هرکدام از مقادیر یک عدد در محدوده ی **128 ... 127** برتی **Enum8** و در محدوده ی **32768 ... 32767** برای **Enum16** می باشد. تمام رشته ها و اعداد باید متفاوت باشند. رشته ی خالی مجاز است. اگر این type مشخص شده باشد (در تعریف جدول)، اعداد می توانند به صورت دلخواه مرتب شوند. با این حال، ترتیب در اینجا مهم نیست.

در RAM، این type مشابه **Int8** یا **Int16** ذخیره می شود. هنگام خواندن در فرم متنی، ClickHouse مقدار را به صورت String پارس می کند و رشته ی مربوطه را در مقادیر Enum جستجو می کند. اگر رشته را پیدا کند یک expectation پرتاب می شود. در هنگام خواندن در فرمت text، رشته خواند می شود و مقدار عددی مربوطه مورد بررسی قرار می گیرد. اگر مقدار پیدا نشود expectation پرتاب می شود. هنگام نوشتن، مقدار با رشته ی مربوط نوشته می شود. اگر داده ی ستون دارای garbage باشد، (اعدادی که از مجموعه ی معتبر نیستند)، یک expectation پرتاب می شود. هنگام خواندن و نوشتن به صورت باینری، این type شبیه به Int8 و Int16 کار می کند. The implicit default value is the value with the lowest number

در حین **ORDER BY, GROUP BY, IN, DISTINCT** و...، این type رفتاری مشابه با اعداد دارد. برای مثال ORDER BY به صورت عددی اینها رو مرتب می کند. اپراتورهای مقایسه ای و مساوی عمل مشابهی در Enums دارند.

مقادیر Enum نمیتوانند با اعداد مقایسه شوند. مقادیر Enum را می توان با رشته ی ثابت مقایسه کرد. اگر رشته ی مقایسه، مقدار معتبری برای مقایسه به Enum نداشته باشد، یک expetion رخ می دهد. اپراتور IN در Enum پشتیبانی می شود؛ به این صورت که ستون enum در سمت چپ و مجموعه از رشته ها در سمت راست قرار می گیرند. رشته ها مقادیر مربوط به Enum هستند.

خیلی از اپراتورهای عددی و رشته ای برای مقادیر Enum تعریف نشده اند. برای مثال جمع یک عدد با Enum یا کانکت کردن یک رشته با Enum. با این حال Enum دارای تابعی به نام **toString** برای برگشت داده به صورت رشته می باشد.

همچنین مقادیر Enum با استفاده از تابع **toT** قابل تبدیل به type های عددی هستند. جایی که T یک type عددی است. زمانی که T به نوع عددی Enum مربوط می شود، این تبدیل هیچ هزینه ای ندارد. اگر فقط مجموعه از مقادیر تغییر کند، Enum بدون هیچ هزینه ای می تواند ALTER شود. اضافه یا حذف کردن عضوهای Enum با استفاده از Alter ممکن است (حذف یک عضو به شرطی که تا الان در جدول از آن استفاده نشده باشد امن است). به عنوان یک محافظ، تغییر مقدار عددی که قبلا توسط اعضای Enum تعریف شده است منجر به expectation می شود.

استفاده از ALTER برای تبدیل Enum8 به Enum16 یا برعکس، ممکن است، دقیقا شبیه به Int8 به Int16.

## (Array(T

آرایه ای از عناصر با تایپ T. تایپ T می تواند هر Type باشد، از جمله یک آرایه. ما توصیه به استفاده از آرایه های multidimensional نمی کنیم، چون آنها به خوبی پشتیبانی نمی شوند (برای مثال، شما نمی تونید در جداولی که موتور آنها MergeTree است، آرایه های multidimensional ذخیره سازی کنید).

## (...AggregateFunction(name, types\_of\_arguments

حالت متوسط از توابع aggregate. برای دریافت آن، از توابع aggregate به همراه پسوند '-State' استفاده کنید. برای اطلاعات بیشتر قسمت "AggregatingMergeTree" را ببینید.

## (... ,Tuple(T1, T2

Tuple ها نمیتوانند در جدول نوشته شوند (به غیر جداول Memory). آنها برای گروه بندی موقت ستون ها مورد استفاده قرار می گیرند. ستون های می توانند گروه بندی شوند اگر از عبارت In در query استفاده کنیم، و برای تعیین فرمت پارامترهای خاصی از توابع لامبدا. برای اطلاعات بیشتر "اپراتور IN"، "توابع Higher order" را ببینید.

Tuple می تواند در خروجی نتیجه query در حال اجرا باشند. در این مورد، برای فرمت های text به غیر از JSON\*, مقادیر به صورت comma-separate داخل براکت قرار میگیرند. در فرمت های JSON\* مقادیر tuple به صورت آرایه در خروجی می آیند.

## (Nullable(TypeName

Allows to store special marker (**NULL**) that denotes "missing value" alongside normal values allowed by **TypeName**. For example, a **Nullable(Int8)** type column can store **Int8** type values, and the rows that don't have a value will store **.NULL**

For a **TypeName**, you can't use composite data types **Array** and **Tuple**. Composite data types can contain **Nullable** ((type values, such as **Array(Nullable(Int8**

.A **Nullable** type field can't be included in table indexes

.**NULL** is the default value for any **Nullable** type, unless specified otherwise in the ClickHouse server configuration

## Storage features

To store **Nullable** type values in table column, ClickHouse uses a separate file with **NULL** masks in addition to normal file with values. Entries in masks file allow ClickHouse to distinguish between **NULL** and default value of corresponding data type for each table row. Because of additional file, **Nullable** column consumes additional .storage space compared to similar normal one

Note

.Using **Nullable** almost always negatively affects performance, keep this in mind when designing your databases

## Usage example

```
CREATE TABLE t_null(x Int8, y Nullable(Int8)) ENGINE TinyLog (:
```

```
CREATE TABLE t_null
)
```

```
,x Int8
(y Nullable(Int8
(
ENGINE = TinyLog
```

```
.Ok
```

```
.rows in set. Elapsed: 0.012 sec 0
```

```
(INSERT INTO t_null VALUES (1, NULL), (2, 3 (:
```

```
INSERT INTO t_null VALUES
```

```
.Ok
```

```
.rows in set. Elapsed: 0.007 sec 1
```

```
SELECT x + y FROM t_null (:
```

```
SELECT x + y
FROM t_null
```

```
┌──(plus(x, y--r
| NULL      |
| 5         |
└──────────┘
```

```
.rows in set. Elapsed: 0.144 sec 2
```

## Nested data structures

### (... ,Nested(Name1 Type1, Name2 Type2

ساختار داده ی nested شبیه به یک جدول nested می باشد. پارامترهای ساختار داده nested، نام ستون ها و type های آنها مشابه دستور CREATE، مشخص می شود. هر سطر از جدول می تواند به هر تعداد سطر در ساختار داده nested مربوط شود.

مثال:

```
CREATE TABLE test.visits
)
,CounterID UInt32
,StartDate Date
,Sign Int8
,IsNew UInt8
,VisitID UInt64
,UserID UInt64
...
Goals Nested
)
,ID UInt32
,Serial UInt32
,EventTime DateTime
,Price Int64
,OrderID String
,CurrencyID UInt32
,(
...
(ENGINE = CollapsingMergeTree(StartDate, intHash32(UserID), (CounterID, StartDate, intHash32(UserID), VisitID), 8192, Sign (
```

این مثال **Goals** را به عنوان یک ساختار داده nested تعریف می کند، که می تواند شامل داده های مربوط به conversion (اهداف رسیده) باشد. هر سطر در جدول **visit** می تواند با صفر یا چند conversion ارتباط داشته باشد.

فقط تا یک لول از nested پشتیبانی می شود. ستون های nested دارای آرایه، با آرایه های multidimensional یکسان هستند، پس محدودیت در پشتیبانی دارند (جداول با موتور MergeTree از ستون های multidimensional پشتیبانی نمی کنند).

در بیشتر موارد، هنگام کار با ساختار داده ی nested، ستون های آن مشخص شده اند. برای این کار، نام ستون ها با استفاده از دات جدا می شوند. این ستون ها آرایه ای از انواع type ها تشکیل می دهند. تمام آرایه های یک ساختار داده ی nested دارای طول ثابت هستند.

مثال:

```
SELECT
,Goals.ID
Goals.EventTime
FROM test.visits
WHERE CounterID = 101500 AND length(Goals.ID) < 5
LIMIT 10
```

s.ID	Goals.EventTime
	['16:42:27 2014-03-17','16:38:48 2014-03-17','16:38:10 2014-03-17']   [1073752,591325,591325]
	['00:28:25 2014-03-17']   [1073752]
	['10:46:20 2014-03-17']   [1073752]
2014-03-17','22:18:07 2014-03-17','22:17:55 2014-03-17','13:59:20 2014-03-17']	[1073752,591325,591325,591325]
['22:18:51	
	[]   []
['14:36:21 2014-03-17','14:07:47 2014-03-17','11:37:06 2014-03-17']	[1073752,591325,591325]
	[]   []
	[]   []
['00:46:05 2014-03-17','00:46:05 2014-03-17']	[591325,1073752]
2014-03-17','18:51:21 2014-03-17','13:30:26 2014-03-17','13:28:33 2014-03-17']	[1073752,591325,591325,591325]
['18:51:45	

ساده ترین راه برای فکر کردن به یک ساختار داده nested این است که، یک nested مجموعه ای از آرایه های چند ستونی با طول ثابت است.

تنها جایی که یک دستور SELECT می تواند کل ساختار داده ی nested را به جای مشخص کردن ستون های آن قرار دهد، عبارت ARRAY JOIN است. برای اطلاعات بیشتر "ARRAY JOIN clause" را ببینید. مثال:

```
SELECT
,Goal.ID
Goal.EventTime
FROM test.visits
ARRAY JOIN Goals AS Goal
WHERE CounterID = 101500 AND length(Goals.ID) < 5
LIMIT 10
```

Goal.ID	Goal.EventTime
16:38:10 2014-03-17	1073752
16:38:48 2014-03-17	591325
16:42:27 2014-03-17	591325
00:28:25 2014-03-17	1073752
10:46:20 2014-03-17	1073752
13:59:20 2014-03-17	1073752
22:17:55 2014-03-17	591325
22:18:07 2014-03-17	591325
22:18:51 2014-03-17	591325
11:37:06 2014-03-17	1073752

شما نمیتوانید در قسمت SELECT تمام ساختار داده ی nested را قرار دهید. شما فقط می توانید ستون های فردی که هر کدام بخشی از این ساختار داده هستند را لیست کنید.

برای INSERT، شما باید تمام ستون های nested را به صورت جدا پاس بدید. در حین درج، سیستم بررسی می کند که تمام آنها طول یکسانی داشته باشند.

برای دستور DESCRIBE، ستون های داخل nested، به صورت جدا ستون می شوند.

دستور ALTER برای عناصر داخل nested بسیار محدود است.

## Special data types

مقادیر نوع داده special، نمیتوانند در در جدول ذخیره و یا در نتایج خروجی قرار بگیرند، اما در نتایج متوسط (intermediate) یک query در حال اجرا استفاده می شوند.

## Expression

برای نشان دادن توابع لامبدا در توابع high-order استفاده می شود.

## Set

برای نصف سمت راست IN استفاده می شود.

## Nothing

The only purpose of this data type is to represent cases where value is not expected. So you can't create a **Nothing** type value

.For example, literal **NULL** has type of **Nullable(Nothing)**. See more about **Nullable**

:The **Nothing** type can also used to denote empty arrays

```
(())SELECT toTypeName(array (:
([])SELECT toTypeName
--((toTypeName(array--r
| (Array(Nothing |
|_____L

.rows in set. Elapsed: 0.062 sec 1
```

## Domains

Domains are special-purpose types, that add some extra features atop of existing base type, leaving on-wire and on-disc format of underlying table intact. At the moment, ClickHouse does not support user-defined domains

:You can use domains anywhere corresponding base type can be used

- Create a column of domain type
- Read/write values from/to domain column
- Use it as index if base type can be used as index
- Call functions with values of domain column
- .etc

## Extra Features of Domains

- Explicit column type name in **SHOW CREATE TABLE** or **DESCRIBE TABLE**
- (...)Input from human-friendly format with **INSERT INTO domain\_table(domain\_column) VALUES**
- Output to human-friendly format for **SELECT domain\_column FROM domain\_table**
- ... Loading data from external source in human-friendly format: **INSERT INTO domain\_table FORMAT CSV**

## Limitations

- .Can't convert index column of base type to domain type via **ALTER TABLE**
- .Can't implicitly convert string values into domain values when inserting data from another column or table
- .Domain adds no constraints on stored values

## IPv4

**IPv4** is a domain based on **UInt32** type and serves as typed replacement for storing IPv4 values. It provides compact storage with human-friendly input-output format, and column type information on inspection

## Basic Usage

```
;CREATE TABLE hits (url String, from IPv4) ENGINE = MergeTree() ORDER BY url
;DESCRIBE TABLE hits
```

name	type	default_type	default_expression	comment	codec_expression
url	String		url		
from	IPv4		from		

:OR you can use IPv4 domain as a key

```
;CREATE TABLE hits (url String, from IPv4) ENGINE = MergeTree() ORDER BY from
```

:**IPv4** domain supports custom input format as IPv4-strings

```
INSERT INTO hits (url, from) VALUES ('https://wikipedia.org', '116.253.40.133')('https://clickhouse.yandex', '183.247.232.58')
;('https://clickhouse.yandex/docs/en/', '116.106.34.242
;SELECT * FROM hits
```

url	from
https://clickhouse.yandex/docs/en/	116.106.34.242
https://wikipedia.org	116.253.40.133
https://clickhouse.yandex	183.247.232.58

:Values are stored in compact binary form

```
;SELECT toTypeName(from), hex(from) FROM hits LIMIT 1
```

toTypeName(from)	hex(from)
IPv4	B7F7E83A

.Domain values are not implicitly convertible to types other than **UInt32**

:If you want to convert **IPv4** value to a string, you have to do that explicitly with **IPv4NumToString()** function

```
;SELECT toTypeName(s), IPv4NumToString(from) as s FROM hits LIMIT 1
```

toTypeName(IPv4NumToString(from))	s
String	183.247.232.58

:Or cast to a **UInt32** value

```
;SELECT toTypeName(i), CAST(from as UInt32) as i FROM hits LIMIT 1
```

toTypeName(CAST(from, 'UInt32'))	i
UInt32	3086477370

## IPv6

**IPv6** is a domain based on **FixedString(16)** type and serves as typed replacement for storing IPv6 values. It provides compact storage with human-friendly input-output format, and column type information on inspection

## Basic Usage

```
;CREATE TABLE hits (url String, from IPv6) ENGINE = MergeTree() ORDER BY url
```

```
;DESCRIBE TABLE hits
```

name	type	default_type	default_expression	comment	codec_expression
			url	String	
			from	IPv6	

:OR you can use **IPv6** domain as a key

```
;CREATE TABLE hits (url String, from IPv6) ENGINE = MergeTree() ORDER BY from
```

:**IPv6** domain supports custom input as IPv6-strings

```
INSERT INTO hits (url, from) VALUES ('https://wikipedia.org', '2a02:aa08:e000:3100::2')('https://clickhouse.yandex',  
;('2001:44c8:129:2632:33:0:252:2')('https://clickhouse.yandex/docs/en/', '2a02:e980:1e::1
```

```
;SELECT * FROM hits
```

url	from
https://clickhouse.yandex	2001:44c8:129:2632:33:0:252:2
https://clickhouse.yandex/docs/en/	2a02:e980:1e::1
https://wikipedia.org	2a02:aa08:e000:3100::2

:Values are stored in compact binary form



```
SELECT toTypeName(from), hex(from) FROM hits LIMIT 1
```

	toTypeName(from)	hex(from)
IPv6		200144C8012926320033000002520002

.Domain values are not implicitly convertible to types other than **FixedString(16)**

.If you want to convert **IPv6** value to a string, you have to do that explicitly with **IPv6NumToString()** function

```
SELECT toTypeName(s), IPv6NumToString(from) as s FROM hits LIMIT 1
```

	toTypeName(IPv6NumToString(from))	s
String		2001:44c8:129:2632:33:0:252:2

.Or cast to a **FixedString(16)** value

```
SELECT toTypeName(i), CAST(from as FixedString(16)) as i FROM hits LIMIT 1
```

	toTypeName(CAST(from, 'FixedString(16)'))	i
FixedString(16)		(FixedString(16

## Table engines

.The table engine (type of table) determines

- .How and where data is stored, where to write it to, and where to read it from
- .Which queries are supported, and how
- .Concurrent data access
- .Use of indexes, if present
- .Whether multithreaded request execution is possible
- .Data replication parameters

When reading, the engine is only required to output the requested columns, but in some cases the engine can partially process data when responding to the request

.For most serious tasks, you should use engines from the **MergeTree** family

## MergeTree

.The **MergeTree** engine and other engines of this family (**\*MergeTree**) are the most robust ClickHouse table engines

The basic idea for **MergeTree** engines family is the following. When you have tremendous amount of a data that should be inserted into the table, you should write them quickly part by part and then merge parts by some rules in background. This method is much more efficient than constantly rewriting data in the storage at the insert

.Main features

- .Stores data sorted by primary key
- .This allows you to create a small sparse index that helps find data faster
- .This allows you to use partitions if the **partitioning key** is specified

ClickHouse supports certain operations with partitions that are more effective than general operations on the same data with the same result. ClickHouse also automatically cuts off the partition data where the partitioning key is specified in the query. This also increases the query performance

- .Data replication support

The family of **ReplicatedMergeTree** tables is used for this. For more information, see the **Data replication** section

.Data sampling support

.If necessary, you can set the data sampling method in the table

Info

.The **Merge** engine does not belong to the **\*MergeTree** family

## Creating a Table

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

.For a description of request parameters, see [request description](#)

### Query clauses

**ENGINE** — Name and parameters of the engine. **ENGINE = MergeTree()**. **MergeTree** engine does not have parameters

.**PARTITION BY** — The [partitioning key](#)

For partitioning by month, use the **toYYYYMM(date\_column)** expression, where **date\_column** is a column with a date of the type **Date**. The partition names here have the "YYYYMM" format

.**ORDER BY** — The sorting key

.(A tuple of columns or arbitrary expressions. Example: **ORDER BY (CounterID, EventDate)**

.**PRIMARY KEY** — The primary key if it differs from the sorting key

By default the primary key is the same as the sorting key (which is specified by the **ORDER BY** clause). Thus in most cases it is unnecessary to specify a separate **PRIMARY KEY** clause

.**SAMPLE BY** — An expression for sampling

If a sampling expression is used, the primary key must contain it. Example: **SAMPLE BY intHash32(UserID) ORDER BY (CounterID, EventDate, intHash32(UserID))**

.**TTL** — An expression for setting storage time for rows

:It must depends on **Date** or **DateTime** column and has one **Date** or **DateTime** column as a result. Example **TTL date + INTERVAL 1 DAY**

For more details, see [TTL for columns and tables](#)

**:SETTINGS** — Additional parameters that control the behavior of the **MergeTree**

- **index\_granularity** — The granularity of an index. The number of data rows between the "marks" of an index. By default, 8192. The list of all available parameters you can see in **MergeTreeSettings.h**
- **use\_minimalistic\_part\_header\_in\_zookeeper** — Storage method of the data parts headers in ZooKeeper. If **use\_minimalistic\_part\_header\_in\_zookeeper=1**, then ZooKeeper stores less data. For more information refer to the **setting description** in the "Server configuration parameters" chapter
- **min\_merge\_bytes\_to\_use\_direct\_io** — The minimum data volume for merge operation required for using of the direct I/O access to the storage disk. During the merging of the data parts, ClickHouse calculates summary storage volume of all the data to be merged. If the volume exceeds **min\_merge\_bytes\_to\_use\_direct\_io** bytes, then ClickHouse reads and writes the data using direct I/O interface (**O\_DIRECT** option) to the storage disk. If **min\_merge\_bytes\_to\_use\_direct\_io = 0**, then the direct I/O is disabled. Default value: **10 \* 1024 \* 1024 \* 1024** bytes
- **merge\_with\_ttl\_timeout** — Minimal time in seconds, when merge with TTL can be repeated. Default value: **86400** (1 day)

### Example of sections setting

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

In the example, we set partitioning by month

We also set an expression for sampling as a hash by the user ID. This allows you to pseudorandomize the data in the table for each **CounterID** and **EventDate**. If, when selecting the data, you define a **SAMPLE** clause, ClickHouse will return an evenly pseudorandom data sample for a subset of users

**index\_granularity** could be omitted because 8192 is the default value

### Deprecated Method for Creating a Table▼

Attention

Do not use this method in new projects and, if possible, switch the old projects to the method described above

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
(ENGINE [=] MergeTree(date-column [, sampling_expression], (primary, key), index_granularity (
```

### MergeTree() parameters

- **date-column** — The name of a column of the type **Date**. ClickHouse automatically creates partitions by month on the basis of this column. The partition names are in the **"YYYYMM"** format
- **sampling\_expression** — an expression for sampling
- **(primary, key)** — primary key. Type — **Tuple**
- **index\_granularity** — The granularity of an index. The number of data rows between the "marks" of an index. The value 8192 is appropriate for most tasks

### Example

```
(MergeTree(EventDate, intHash32(UserID), (CounterID, EventDate, intHash32(UserID))), 8192
```

The **MergeTree** engine is configured in the same way as in the example above for the main engine configuration method

## Data Storage

A table consists of data *parts* sorted by primary key

When data is inserted in a table, separate data parts are created and each of them is lexicographically sorted by primary key. For example, if the primary key is (CounterID, Date), the data in the part is sorted by CounterID, and .within each CounterID, it is ordered by Date

Data belonging to different partitions are separated into different parts. In the background, ClickHouse merges data parts for more efficient storage. Parts belonging to different partitions are not merged. The merge .mechanism does not guarantee that all rows with the same primary key will be in the same data part

For each data part, ClickHouse creates an index file that contains the primary key value for each index row ("mark"). Index row numbers are defined as  $n * index\_granularity$ . The maximum value  $n$  is equal to the integer part of dividing the total number of rows by the index\_granularity. For each column, the "marks" are also written for the .same index rows as the primary key. These "marks" allow you to find the data directly in the columns

You can use a single large table and continually add data to it in small chunks – this is what the MergeTree engine .is intended for

## Primary Keys and Indexes in Queries

:Let's take the (CounterID, Date) primary key. In this case, the sorting and index can be illustrated as follows

[-----] :Whole data										
[CounterID: [aaaaaaaaaaaaaaaaabbbbcdeeeeeeeeeefggggggghhhhhhhhhiiiiiklllllll]										
[Date: [1111111222222333312332111112222223332111111212222223111112223311122333]										
:Marks										
a,1	a,2	a,3	b,3	e,2	e,3	g,1	h,2	i,1	i,3	l,3
Marks numbers: 0 1 2 3 4 5 6 7 8 9 10										

:If the data query specifies

- .(CounterID in ('a', 'h'), the server reads the data in the ranges of marks [0, 3) and [6, 8)
- .(CounterID IN ('a', 'h') AND Date = 3, the server reads the data in the ranges of marks [1, 3) and [7, 8)
- .[Date = 3, the server reads the data in the range of marks [1, 10)

.The examples above show that it is always more effective to use an index than a full scan

A sparse index allows extra data to be read. When reading a single range of the primary key, up to index\_granularity \* 2 extra rows in each data block can be read. In most cases, ClickHouse performance does not degrade when .index\_granularity = 8192

Sparse indexes allow you to work with a very large number of table rows, because such indexes are always stored .in the computer's RAM

.ClickHouse does not require a unique primary key. You can insert multiple rows with the same primary key

## Selecting the Primary Key

The number of columns in the primary key is not explicitly limited. Depending on the data structure, you can :include more or fewer columns in the primary key. This may

- .Improve the performance of an index

If the primary key is (a, b), then adding another column c will improve the performance if the following :conditions are met

- .There are queries with a condition on column c -
- Long data ranges (several times longer than the index\_granularity) with identical values for (a, b) are common. -
- .In other words, when adding another column allows you to skip quite long data ranges

- .Improve data compression

.ClickHouse sorts data by primary key, so the higher the consistency, the better the compression

Provide additional logic when data parts merging in the [CollapsingMergeTree](#) and [SummingMergeTree](#) engines

.In this case it makes sense to specify the *sorting key* that is different from the primary key

A long primary key will negatively affect the insert performance and memory consumption, but extra columns in the primary key do not affect ClickHouse performance during [SELECT](#) queries

## Choosing the Primary Key that differs from the Sorting Key

It is possible to specify the primary key (the expression, values of which are written into the index file (for each mark) that is different from the sorting key (the expression for sorting the rows in data parts). In this case the primary key expression tuple must be a prefix of the sorting key expression tuple

This feature is helpful when using the [SummingMergeTree](#) and [AggregatingMergeTree](#) table engines. In a common case when using these engines the table has two types of columns: *dimensions* and *measures*. Typical queries aggregate values of measure columns with arbitrary [GROUP BY](#) and filtering by dimensions. As [SummingMergeTree](#) and [AggregatingMergeTree](#) aggregate rows with the same value of the sorting key, it is natural to add all dimensions to it. As a result the key expression consists of a long list of columns and this list must be frequently updated with newly added dimensions

In this case it makes sense to leave only a few columns in the primary key that will provide efficient range scans and add the remaining dimension columns to the sorting key tuple

[ALTER of the sorting key](#) is a lightweight operation because when a new column is simultaneously added to the table and to the sorting key, existing data parts don't need to be changed. Since the old sorting key is a prefix of the new sorting key and there is no data in the just added column, the data at the moment of table modification is sorted by both the old and the new sorting key

## Use of Indexes and Partitions in Queries

For [SELECT](#) queries, ClickHouse analyzes whether an index can be used. An index can be used if the [WHERE/PREWHERE](#) clause has an expression (as one of the conjunction elements, or entirely) that represents an equality or inequality comparison operation, or if it has [IN](#) or [LIKE](#) with a fixed prefix on columns or expressions that are in the primary key or partitioning key, or on certain partially repetitive functions of these columns, or logical relationships of these expressions

Thus, it is possible to quickly run queries on one or many ranges of the primary key. In this example, queries will be fast when run for a specific tracking tag; for a specific tag and date range; for a specific tag and date; for multiple tags with a date range, and so on

:Let's look at the engine configured as follows

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

:In this case, in queries

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

ClickHouse will use the primary key index to trim improper data and the monthly partitioning key to trim partitions that are in improper date ranges

The queries above show that the index is used even for complex expressions. Reading from the table is organized so that using the index can't be slower than a full scan

.In the example below, the index can't be used

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

To check whether ClickHouse can use the index when running a query, use the settings `force_index_by_date` and `force_primary_key`

The key for partitioning by month allows reading only those data blocks which contain dates from the proper range. In this case, the data block may contain data for many dates (up to an entire month). Within a block, data is sorted by primary key, which might not contain the date as the first column. Because of this, using a query with only a date condition that does not specify the primary key prefix will cause more data to be read than for a single date

## Use of Index for Partially-Monotonic Primary Keys

Consider, for example, the days of the month. They are the **monotonic sequence** inside one month, but they are not monotonic for a more extended period. This is the partially-monotonic sequence. If a user creates the table with such partially-monotonic primary key, ClickHouse creates a sparse index as usual. When a user selects data from such a table, ClickHouse analyzes query conditions. If the user wants to get data between two marks of the index and both this marks are within one month, ClickHouse can use the index in this particular case because it can calculate the distance between parameters of query and index marks

ClickHouse cannot use an index if the values of the primary key on the query parameters range don't represent the monotonic sequence. In this case, ClickHouse uses full scan method

ClickHouse uses this logic not only for days of month sequences but for any primary key which represents a partially-monotonic sequence

## (Data Skipping Indices (Experimental

You need to set `allow_experimental_data_skipping_indices` to 1 to use indices. (run `SET (allow_experimental_data_skipping_indices = 1`

.Index declaration in the columns section of the `CREATE` query

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

.For tables from the **\*MergeTree** family data skipping indices can be specified

These indices aggregate some information about the specified expression on blocks, which consist of **granularity\_value** granules (size of the granule is specified using `index_granularity` setting in the table engine), then these aggregates are used in `SELECT` queries for reducing the amount of data to read from the disk by skipping big blocks of data where **where** query cannot be satisfied

### Example

```
CREATE TABLE table_name
)
,u64 UInt64
,i32 Int32
,s String
...
,INDEX a (u64 * i32, s) TYPE minmax GRANULARITY 3
INDEX b (u64 * length(s)) TYPE set(1000) GRANULARITY 4
()ENGINE = MergeTree (
...
```

Indices from the example can be used by ClickHouse to reduce the amount of data to read from disk in following queries

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

## Available Types of Indices

minmax

Stores extremes of the specified expression (if the expression is **tuple**, then it stores extremes for each element of **tuple**), uses stored info for skipping blocks of the data like the primary key

(set(max\_rows

Stores unique values of the specified expression (no more than **max\_rows** rows, **max\_rows=0** means "no limits"), use them to check if the **WHERE** expression is not satisfiable on a block of the data

(ngrambf\_v1(n, size\_of\_bloom\_filter\_in\_bytes, number\_of\_hash\_functions, random\_seed

Stores **bloom filter** that contains all ngrams from block of data. Works only with strings. Can be used for optimization of **equals**, **like** and **in** expressions

,n — ngram size

size\_of\_bloom\_filter\_in\_bytes — bloom filter size in bytes (you can use big values here, for example, 256 or 512, because it can be compressed well

,number\_of\_hash\_functions — number of hash functions used in bloom filter

.random\_seed — seed for bloom filter hash functions

(tokenbf\_v1(size\_of\_bloom\_filter\_in\_bytes, number\_of\_hash\_functions, random\_seed

The same as **ngrambf\_v1**, but instead of ngrams stores tokens, which are sequences separated by non-alphanumeric characters

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

## Concurrent Data Access

For concurrent table access, we use multi-versioning. In other words, when a table is simultaneously read and updated, data is read from a set of parts that is current at the time of the query. There are no lengthy locks. Inserts do not get in the way of read operations

.Reading from a table is automatically parallelized

## TTL for columns and tables

.Data with expired TTL is removed while executing merges

If TTL is set for column, when it expires, value will be replaced by default. If all values in columns were zeroed in part, data for this column will be deleted from disk for part. You are not allowed to set TTL for all key columns. If TTL is set for table, when it expires, row will be deleted

When TTL expires on some value or row in part, extra merge will be executed. To control frequency of merges with TTL you can set **merge\_with\_ttl\_timeout**. If it is too low, many extra merges and lack of regular merges can reduce the performance

## Data Replication

:Replication is only supported for tables in the MergeTree family

ReplicatedMergeTree

ReplicatedSummingMergeTree

ReplicatedReplacingMergeTree

ReplicatedAggregatingMergeTree

ReplicatedCollapsingMergeTree

ReplicatedVersionedCollapsingMergeTree

ReplicatedGraphiteMergeTree

Replication works at the level of an individual table, not the entire server. A server can store both replicated and non-replicated tables at the same time

Replication does not depend on sharding. Each shard has its own independent replication

Compressed data for **INSERT** and **ALTER** queries is replicated (for more information, see the documentation for **ALTER**)

**CREATE**, **DROP**, **ATTACH**, **DETACH** and **RENAME** queries are executed on a single server and are not replicated

The **CREATE TABLE** query creates a new replicatable table on the server where the query is run. If this table already exists on other servers, it adds a new replica

The **DROP TABLE** query deletes the replica located on the server where the query is run

The **RENAME** query renames the table on one of the replicas. In other words, replicated tables can have different names on different replicas

To use replication, set the addresses of the ZooKeeper cluster in the config file. Example

```
<zookeeper>
<"node index="1>
<host>example1</host>
<port>2181</port>
<node/>
<"node index="2>
<host>example2</host>
<port>2181</port>
<node/>
<"node index="3>
<host>example3</host>
<port>2181</port>
<node/>
<zookeeper/>
```

Use ZooKeeper version 3.4.5 or later

You can specify any existing ZooKeeper cluster and the system will use a directory on it for its own data (the directory is specified when creating a replicatable table)

If ZooKeeper isn't set in the config file, you can't create replicated tables, and any existing replicated tables will be read-only

ZooKeeper is not used in **SELECT** queries because replication does not affect the performance of **SELECT** and queries run just as fast as they do for non-replicated tables. When querying distributed replicated tables, ClickHouse behavior is controlled by the settings **max\_replica\_delay\_for\_distributed\_queries** and **fallback\_to\_stale\_replicas\_for\_distributed\_queries**

For each **INSERT** query, approximately ten entries are added to ZooKeeper through several transactions. (To be more precise, this is for each inserted block of data; an **INSERT** query contains one block or one block per **max\_insert\_block\_size = 1048576** rows.) This leads to slightly longer latencies for **INSERT** compared to non-replicated tables. But if you follow the recommendations to insert data in batches of no more than one **INSERT** per second, it doesn't create any problems. The entire ClickHouse cluster used for coordinating one ZooKeeper cluster has a total of several hundred **INSERTs** per second. The throughput on data inserts (the number of rows per second) is just as high as for non-replicated data

For very large clusters, you can use different ZooKeeper clusters for different shards. However, this hasn't proven necessary on the Yandex.Metrica cluster (approximately 300 servers)

Replication is asynchronous and multi-master. **INSERT** queries (as well as **ALTER**) can be sent to any available server. Data is inserted on the server where the query is run, and then it is copied to the other servers. Because it is asynchronous, recently inserted data appears on the other replicas with some latency. If part of the replicas are not available, the data is written when they become available. If a replica is available, the latency is the amount of time it takes to transfer the block of compressed data over the network



By default, an INSERT query waits for confirmation of writing the data from only one replica. If the data was successfully written to only one replica and the server with this replica ceases to exist, the stored data will be lost. To enable getting confirmation of data writes from multiple replicas, use the `insert_quorum` option

Each block of data is written atomically. The INSERT query is divided into blocks up to `max_insert_block_size = 1048576` rows. In other words, if the `INSERT` query has less than 1048576 rows, it is made atomically

Data blocks are deduplicated. For multiple writes of the same data block (data blocks of the same size containing the same rows in the same order), the block is only written once. The reason for this is in case of network failures when the client application doesn't know if the data was written to the DB, so the `INSERT` query can simply be repeated. It doesn't matter which replica INSERTs were sent to with identical data. `INSERTs` are idempotent. Deduplication parameters are controlled by `merge_tree` server settings

During replication, only the source data to insert is transferred over the network. Further data transformation (merging) is coordinated and performed on all the replicas in the same way. This minimizes network usage, which means that replication works well when replicas reside in different datacenters. (Note that duplicating data in different datacenters is the main goal of replication)

You can have any number of replicas of the same data. Yandex.Metrica uses double replication in production. Each server uses RAID-5 or RAID-6, and RAID-10 in some cases. This is a relatively reliable and convenient solution

The system monitors data synchronicity on replicas and is able to recover after a failure. Failover is automatic (for small differences in data) or semi-automatic (when data differs too much, which may indicate a configuration error)

## Creating Replicated Tables

The `Replicated` prefix is added to the table engine name. For example: `ReplicatedMergeTree`

### Replicated\*MergeTree parameters

`zoo_path` — The path to the table in ZooKeeper

`replica_name` — The replica name in ZooKeeper

- 
- 

Example

```
CREATE TABLE table_name
)
,EventDate DateTime
,CounterID UInt32
UserID UInt32
(' {ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/table_name', '{replica (
(PARTITION BY toYYYYMM(EventDate
((ORDER BY (CounterID, EventDate, intHash32(UserID
(SAMPLE BY intHash32(UserID
```

Example in deprecated syntax▼

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

As the example shows, these parameters can contain substitutions in curly brackets. The substituted values are taken from the 'macros' section of the configuration file. Example

```
<macros>
<layer>05</layer>
<shard>02</shard>
<replica>example05-02-1.yandex.ru</replica>
</macros>
```

The path to the table in ZooKeeper should be unique for each replicated table. Tables on different shards should have different paths

In this case, the path consists of the following parts

`.clickhouse/tables/` is the common prefix. We recommend using exactly this one/

`layer}-{shard}` is the shard identifier. In this example it consists of two parts, since the Yandex.Metrica cluster uses bi-level sharding. For most tasks, you can leave just the `{shard}` substitution, which will be expanded to the shard identifier

`table_name` is the name of the node for the table in ZooKeeper. It is a good idea to make it the same as the table name. It is defined explicitly, because in contrast to the table name, it doesn't change after a RENAME query

*HINT:* you could add a database name in front of `table_name` as well. E.g. `db_name.table_name`

The replica name identifies different replicas of the same table. You can use the server name for this, as in the example. The name only needs to be unique within each shard

You can define the parameters explicitly instead of using substitutions. This might be convenient for testing and for configuring small clusters. However, you can't use distributed DDL queries (`ON CLUSTER`) in this case

When working with large clusters, we recommend using substitutions because they reduce the probability of error

Run the `CREATE TABLE` query on each replica. This query creates a new replicated table, or adds a new replica to an existing one

If you add a new replica after the table already contains some data on other replicas, the data will be copied from the other replicas to the new one after running the query. In other words, the new replica syncs itself with the others

To delete a replica, run `DROP TABLE`. However, only one replica is deleted – the one that resides on the server where you run the query

## Recovery After Failures

If ZooKeeper is unavailable when a server starts, replicated tables switch to read-only mode. The system periodically attempts to connect to ZooKeeper

If ZooKeeper is unavailable during an `INSERT`, or an error occurs when interacting with ZooKeeper, an exception is thrown

After connecting to ZooKeeper, the system checks whether the set of data in the local file system matches the expected set of data (ZooKeeper stores this information). If there are minor inconsistencies, the system resolves them by syncing data with the replicas

If the system detects broken data parts (with the wrong size of files) or unrecognized parts (parts written to the file system but not recorded in ZooKeeper), it moves them to the `detached` subdirectory (they are not deleted). Any missing parts are copied from the replicas

Note that ClickHouse does not perform any destructive actions such as automatically deleting a large amount of data

When the server starts (or establishes a new session with ZooKeeper), it only checks the quantity and sizes of all files. If the file sizes match but bytes have been changed somewhere in the middle, this is not detected immediately, but only when attempting to read the data for a `SELECT` query. The query throws an exception about a non-matching checksum or size of a compressed block. In this case, data parts are added to the verification queue and copied from the replicas if necessary

If the local set of data differs too much from the expected one, a safety mechanism is triggered. The server enters this in the log and refuses to launch. The reason for this is that this case may indicate a configuration error, such as if a replica on a shard was accidentally configured like a replica on a different shard. However, the thresholds for this mechanism are set fairly low, and this situation might occur during normal failure recovery. In this case, "data is restored semi-automatically - by "pushing a button

To start recovery, create the node `/path_to_table/replica_name/flags/force_restore_data` in ZooKeeper with any content, or run the command to restore all replicated tables

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

.Then restart the server. On start, the server deletes these flags and starts recovery

## Recovery After Complete Data Loss

:If all data and metadata disappeared from one of the servers, follow these steps for recovery

Install ClickHouse on the server. Define substitutions correctly in the config file that contains the shard identifier and replicas, if you use them .1

If you had unreplicated tables that must be manually duplicated on the servers, copy their data from a replica .2  
.(/in the directory `/var/lib/clickhouse/data/db_name/table_name`

Copy table definitions located in `/var/lib/clickhouse/metadata/` from a replica. If a shard or replica identifier is defined explicitly in the table definitions, correct it so that it corresponds to this replica. (Alternatively, start the server and make all the `ATTACH TABLE` queries that should have been in the .sql files in `./var/lib/clickhouse/metadata` .3

To start recovery, create the ZooKeeper node `/path_to_table/replica_name/flags/force_restore_data` with any content, or run the command to restore all replicated tables: `sudo -u clickhouse touch /var/lib/clickhouse/flags/force_restore_data` .4

.Then start the server (restart, if it is already running). Data will be downloaded from replicas

An alternative recovery option is to delete information about the lost replica from ZooKeeper `"/path_to_table/replica_name)`, then create the replica again as described in "[Creating replicated tables](#)

There is no restriction on network bandwidth during recovery. Keep this in mind if you are restoring many replicas at once

## Converting from MergeTree to ReplicatedMergeTree

.We use the term `MergeTree` to refer to all table engines in the `MergeTree` family, the same as for `ReplicatedMergeTree`

If you had a `MergeTree` table that was manually replicated, you can convert it to a replicated table. You might need to do this if you have already collected a large amount of data in a `MergeTree` table and now you want to enable replication

.If the data differs on various replicas, first sync it, or delete this data on all the replicas except one

.Rename the existing `MergeTree` table, then create a `ReplicatedMergeTree` table with the old name  
Move the data from the old table to the `detached` subdirectory inside the directory with the new table data  
.(/var/lib/clickhouse/data/db\_name/table\_name

.Then run `ALTER TABLE ATTACH PARTITION` on one of the replicas to add these data parts to the working set

## Converting from ReplicatedMergeTree to MergeTree

Create a `MergeTree` table with a different name. Move all the data from the directory with the `ReplicatedMergeTree` table data to the new table's data directory. Then delete the `ReplicatedMergeTree` table and restart the server

:If you want to get rid of a `ReplicatedMergeTree` table without launching the server

- .(Delete the corresponding .sql file in the metadata directory `/var/lib/clickhouse/metadata`
- .(Delete the corresponding path in ZooKeeper `/path_to_table/replica_name`

After this, you can launch the server, create a **MergeTree** table, move the data to its directory, and then restart the .server

## Recovery When Metadata in The ZooKeeper Cluster is Lost or Damaged

If the data in ZooKeeper was lost or damaged, you can save data by moving it to an unreplicated table as .described above

## Custom Partitioning Key

Partitioning is available for the **MergeTree** family tables (including **replicated** tables). **Materialized views** based on .MergeTree tables support partitioning, as well

A partition is a logical combination of records in a table by a specified criterion. You can set a partition by an arbitrary criterion, such as by month, by day, or by event type. Each partition is stored separately in order to simplify manipulations of this data. When accessing the data, ClickHouse uses the smallest subset of partitions .possible

The partition is specified in the **PARTITION BY expr** clause when **creating a table**. The partition key can be any expression from the table columns. For example, to specify partitioning by month, use the expression  
:(toYYYYMM(date\_column

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

:The partition key can also be a tuple of expressions (similar to the **primary key**). For example

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

.In this example, we set partitioning by the event types that occurred during the current week

When inserting new data to a table, this data is stored as a separate part (chunk) sorted by the primary key. In 10-15 minutes after inserting, the parts of the same partition are merged into the entire part

Info

A merge only works for data parts that have the same value for the partitioning expression. This means **you shouldn't make overly granular partitions** (more than about a thousand partitions). Otherwise, the **SELECT** query performs poorly because of an unreasonably large number of files in the file system and open file .descriptors

Use the **system.parts** table to view the table parts and partitions. For example, let's assume that we have a **visits** :table with partitioning by month. Let's perform the **SELECT** query for the **system.parts** table

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

partition	name	active
0	1_3_1_201901	201901
1	2_9_1_201901	201901
0	0_8_8_201901	201901
0	0_9_9_201901	201901
1	1_6_4_201902	201902
1	0_10_10_201902	201902
1	0_11_11_201902	201902

The **partition** column contains the names of the partitions. There are two partitions in this example: **201901** and **201902**. You can use this column value to specify the partition name in **ALTER ... PARTITION** queries

The **name** column contains the names of the partition data parts. You can use this column to specify the name of the part in the **ALTER ATTACH PART** query

:Let's break down the name of the first part: **201901\_1\_3\_1**

- .is the partition name **201901**
- .is the minimum number of the data block **1**
- .is the maximum number of the data block **3**
- .(is the chunk level (the depth of the merge tree it is formed from **1**

Info

The parts of old-type tables have the name: **20190117\_20190123\_2\_2\_0** (minimum date - maximum date - minimum (block number - maximum block number - level

The **active** column shows the status of the part. **1** is active; **0** is inactive. The inactive parts are, for example, source parts remaining after merging to a larger part. The corrupted data parts are also indicated as inactive

As you can see in the example, there are several separated parts of the same partition (for example, **201901\_1\_3\_1** and **201901\_1\_9\_2**). This means that these parts are not merged yet. ClickHouse merges the inserted parts of data periodically, approximately 15 minutes after inserting. In addition, you can perform a non-scheduled merge using the **OPTIMIZE** query. Example

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

partition	name	active
0	1_3_1_201901	201901
1	2_9_1_201901	201901
0	0_8_8_201901	201901
0	0_9_9_201901	201901
0	1_6_4_201902	201902
1	2_11_4_201902	201902
0	0_10_10_201902	201902
0	0_11_11_201902	201902

.Inactive parts will be deleted approximately 10 minutes after merging

Another way to view a set of parts and partitions is to go into the directory of the table:

**/var/lib/clickhouse/data/<database>/<table>/. For example**

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

The folders '201901\_1\_1\_0', '201901\_1\_7\_1' and so on are the directories of the parts. Each part relates to a corresponding partition and contains data just for a certain month (the table in this example has partitioning by month).

The **detached** directory contains parts that were detached from the table using the **DETACH** query. The corrupted parts are also moved to this directory, instead of being deleted. The server does not use the parts from the **detached** directory. You can add, delete, or modify the data in this directory at any time – the server will not know about this until you run the **ATTACH** query.

Note that on the operating server, you cannot manually change the set of parts or their data on the file system, since the server will not know about it. For non-replicated tables, you can do this when the server is stopped, but it isn't recommended. For replicated tables, the set of parts cannot be changed in any case.

ClickHouse allows you to perform operations with the partitions: delete them, copy from one table to another, or create a backup. See the list of all operations in the section **Manipulations With Partitions and Parts**.

## ReplacingMergeTree

The engine differs from **MergeTree** in that it removes duplicate entries with the same primary key value (or more accurately, with the same **sorting key** value).

Data deduplication occurs only during a merge. Merging occurs in the background at an unknown time, so you can't plan for it. Some of the data may remain unprocessed. Although you can run an unscheduled merge using the **OPTIMIZE** query, don't count on using it, because the **OPTIMIZE** query will read and write a large amount of data.

Thus, **ReplacingMergeTree** is suitable for clearing out duplicate data in the background in order to save space, but it doesn't guarantee the absence of duplicates.

## Creating a Table

```
(CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
([ENGINE = ReplacingMergeTree([ver (
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[... ,SETTINGS name=value]
```

For a description of request parameters, see **request description**.

### ReplacingMergeTree Parameters

**ver** — column with version. Type **UInt\***, **Date** or **DateTime**. Optional parameter

When merging, **ReplacingMergeTree** from all the rows with the same primary key leaves only one

Last in the selection, if **ver** not set -

With the maximum version, if **ver** specified -

## Query clauses

.When creating a [ReplacingMergeTree](#) table the same [clauses](#) are required, as when creating a [MergeTree](#) table

Deprecated Method for Creating a Table▼

Attention

.Do not use this method in new projects and, if possible, switch the old projects to the method described above

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
([ENGINE [=] ReplacingMergeTree(date-column [, sampling_expression], (primary, key), index_granularity, [ver (
```

.All of the parameters excepting [ver](#) have the same meaning as in [MergeTree](#)

.[ver](#) - column with the version. Optional parameter. For a description, see the text above

•

## SummingMergeTree

The engine inherits from [MergeTree](#). The difference is that when merging data parts for [SummingMergeTree](#) tables ClickHouse replaces all the rows with the same primary key (or more accurately, with the same [sorting key](#)) with one row which contains summarized values for the columns with the numeric data type. If the sorting key is composed in a way that a single key value corresponds to large number of rows, this significantly reduces storage volume and speeds up data selection

We recommend to use the engine together with [MergeTree](#). Store complete data in [MergeTree](#) table, and use [SummingMergeTree](#) for aggregated data storing, for example, when preparing reports. Such an approach will prevent you from losing valuable data due to an incorrectly composed primary key

## Creating a Table

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
([ENGINE = SummingMergeTree([columns (
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[... ,SETTINGS name=value]
```

.For a description of request parameters, see [request description](#)

### Parameters of SummingMergeTree

.[columns](#) - a tuple with the names of columns where values will be summarized. Optional parameter

•

.The columns must be of a numeric type and must not be in the primary key

If [columns](#) not specified, ClickHouse summarizes the values in all columns with a numeric data type that are not in the primary key

## Query clauses

.When creating a [SummingMergeTree](#) table the same [clauses](#) are required, as when creating a [MergeTree](#) table

Deprecated Method for Creating a Table▼

Attention

.Do not use this method in new projects and, if possible, switch the old projects to the method described above

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
([ENGINE [=] SummingMergeTree(date-column [, sampling_expression], (primary, key), index_granularity, [columns (
```

.All of the parameters excepting **columns** have the same meaning as in **MergeTree**

**columns** — tuple with names of columns values of which will be summarized. Optional parameter. For a description, see the text above

## Usage Example

:Consider the following table

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

:Insert data to it

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

ClickHouse may sum all the rows not completely (see below), so we use an aggregate function **sum** and **GROUP BY** clause in the query

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

key	sum(value)
1	2
3	1

## Data Processing

When data are inserted into a table, they are saved as-is. Clickhouse merges the inserted parts of data periodically and this is when rows with the same primary key are summed and replaced with one for each resulting part of data

ClickHouse can merge the data parts so that different resulting parts of data cat consist rows with the same primary key, i.e. the summation will be incomplete. Therefore (**SELECT**) an aggregate function **sum()** and **GROUP BY** clause should be used in a query as described in the example above

## Common rules for summation

The values in the columns with the numeric data type are summarized. The set of columns is defined by the parameter **columns**

- .If the values were 0 in all of the columns for summation, the row is deleted
- .If column is not in the primary key and is not summarized, an arbitrary value is selected from the existing ones
- .The values are not summarized for columns in the primary key

## The Summation in the AggregateFunction Columns

For columns of **AggregateFunction** type ClickHouse behaves as **AggregatingMergeTree** engine aggregating according to the function



## Nested Structures

.Table can have nested data structures that are processed in a special way

:If the name of a nested table ends with **Map** and it contains at least two columns that meet the following criteria

,the first column is numeric (**\*Int\***, **Date**, **DateTime**), let's call it **key**

,(...the other columns are arithmetic (**\*Int\***, **Float32/64**), let's call it (**values**

- 
- 

then this nested table is interpreted as a mapping of **key => (values...)**, and when merging its rows, the elements of

.(...two data sets are merged by **key** with a summation of the corresponding (**values**

:Examples

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

.When requesting data, use the **sumMap(key, value)** function for aggregation of **Map**

.For nested data structure, you do not need to specify its columns in the tuple of columns for summation

## AggregatingMergeTree

The engine inherits from **MergeTree**, altering the logic for data parts merging. ClickHouse replaces all rows with the same primary key (or more accurately, with the same **sorting key**) with a single row (within a one data part) .that stores a combination of states of aggregate functions

You can use **AggregatingMergeTree** tables for incremental data aggregation, including for aggregated materialized .views

.The engine processes all columns with **AggregateFunction** type

.It is appropriate to use **AggregatingMergeTree** if it reduces the number of rows by orders

## Creating a Table

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
,[name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
,[name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
()ENGINE = AggregatingMergeTree (
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[... ,SETTINGS name=value]
```

.For a description of request parameters, see **request description**

### Query clauses

.When creating a **AggregatingMergeTree** table the same **clauses** are required, as when creating a **MergeTree** table

Deprecated Method for Creating a Table▼

Attention

.Do not use this method in new projects and, if possible, switch the old projects to the method described above

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
,[name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
,[name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
(ENGINE [=] AggregatingMergeTree(date-column [, sampling_expression], (primary, key), index_granularity (
```

.All of the parameters have the same meaning as in [MergeTree](#)

## SELECT and INSERT

.To insert data, use [INSERT SELECT](#) query with aggregate -State- functions

When selecting data from [AggregatingMergeTree](#) table, use [GROUP BY](#) clause and the same aggregate functions as when inserting data, but using -Merge suffix

In the results of [SELECT](#) query the values of [AggregateFunction](#) type have implementation-specific binary representation for all of the ClickHouse output formats. If dump data into, for example, [TabSeparated](#) format with [SELECT](#) query then this dump can be loaded back using [INSERT](#) query

## Example of an Aggregated Materialized View

[AggregatingMergeTree](#) materialized view that watches the [test.visits](#) table

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

.Inserting of data into the [test.visits](#) table

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

.The data are inserted in both the table and view [test.basic](#) that will perform the aggregation

:To get the aggregated data, we need to execute a query such as [SELECT ... GROUP BY ...](#) from the view [test.basic](#)

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

## CollapsingMergeTree

.The engine inherits from [MergeTree](#) and adds the logic of rows collapsing to data parts merge algorithm

[CollapsingMergeTree](#) asynchronously deletes (collapses) pairs of rows if all of the fields in a row are equivalent excepting the particular field [Sign](#) which can have [1](#) and [-1](#) values. Rows without a pair are kept. For more details .see the [Collapsing](#) section of the document

The engine may significantly reduce the volume of storage and increase efficiency of [SELECT](#) query as a consequence

## Creating a Table

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster
)
,[name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1
,[name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2
...
(ENGINE = CollapsingMergeTree(sign (
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[... ,SETTINGS name=value]
```

.For a description of query parameters, see [query description](#)

## CollapsingMergeTree Parameters

.**sign** — Name of the column with the type of row: **1** is a "state" row, **-1** is a "cancel" row

.Column data type — **Int8**

## Query clauses

When creating a **CollapsingMergeTree** table, the same **query clauses** are required, as when creating a **MergeTree** table

## Deprecated Method for Creating a Table▼

### Attention

.Do not use this method in new projects and, if possible, switch the old projects to the method described above

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster
)
,[name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1
,[name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2
...
(ENGINE [=] CollapsingMergeTree(date-column [, sampling_expression], (primary, key), index_granularity, sign (
```

.All of the parameters excepting **sign** have the same meaning as in **MergeTree**

.**sign** — Name of the column with the type of row: **1** — "state" row, **-1** — "cancel" row

.Column Data Type — **Int8**

# Collapsing Data

Consider the situation where you need to save continually changing data for some object. It sounds logical to have one row for an object and update it at any change, but update operation is expensive and slow for DBMS because it requires rewriting of the data in the storage. If you need to write data quickly, update not acceptable, but you can write the changes of an object sequentially as follows

Use the particular column **Sign**. If **Sign = 1** it means that the row is a state of an object, let's call it "state" row. If **Sign = -1** it means the cancellation of the state of an object with the same attributes, let's call it "cancel" row

For example, we want to calculate how much pages users checked at some site and how long they were there. At some moment of time we write the following row with the state of user activity

UserID	PageViews	Duration	Sign
1	146	5	4324182021466249494

.At some moment later we register the change of user activity and write it with the following two rows

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

The first row cancels the previous state of the object (user). It should copy all of the fields of the canceled state .excepting **Sign**

.The second row contains the current state

As we need only the last state of user activity, the rows

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

can be deleted collapsing the invalid (old) state of an object. **CollapsingMergeTree** does this while merging of the .data parts

.Why we need 2 rows for each change read in the **Algorithm** paragraph

### Peculiar properties of such approach

- .1 The program that writes the data should remember the state of an object to be able to cancel it. "Cancel" string should be the copy of "state" string with the opposite **Sign**. It increases the initial size of storage but .allows to write the data quickly
- .2 Long growing arrays in columns reduce the efficiency of the engine due to load for writing. The more .straightforward data, the higher efficiency
- .3 The **SELECT** results depend strongly on the consistency of object changes history. Be accurate when preparing data for inserting. You can get unpredictable results in inconsistent data, for example, negative values for .non-negative metrics such as session depth

## Algorithm

When ClickHouse merges data parts, each group of consecutive rows with the same primary key is reduced to not more than two rows, one with **Sign = 1** ("state" row) and another with **Sign = -1** ("cancel" row). In other words, .entries collapse

:For each resulting data part ClickHouse saves

- .1 .The first "cancel" and the last "state" rows, if the number of "state" and "cancel" rows matches
- .2 .The last "state" row, if there is one more "state" row than "cancel" rows
- .3 .The first "cancel" row, if there is one more "cancel" row than "state" rows
- .4 .None of the rows, in all other cases

The merge continues, but ClickHouse treats this situation as a logical error and records it in the server log. .This error can occur if the same data were inserted more than once

- .Thus, collapsing should not change the results of calculating statistics
- .Changes gradually collapsed so that in the end only the last state of almost every object left

The **Sign** is required because the merging algorithm doesn't guarantee that all of the rows with the same primary key will be in the same resulting data part and even on the same physical server. ClickHouse process **SELECT** queries with multiple threads, and it can not predict the order of rows in the result. The aggregation is required if .there is a need to get completely "collapsed" data from **CollapsingMergeTree** table

To finalize collapsing write a query with **GROUP BY** clause and aggregate functions that account for the sign. For example, to calculate quantity, use **sum(Sign)** instead of **count()**. To calculate the sum of something, use **sum(Sign \* x)** .instead of **sum(x)**, and so on, and also add **HAVING sum(Sign) > 0**

The aggregates **count**, **sum** and **avg** could be calculated this way. The aggregate **uniq** could be calculated if an object has at least one state not collapsed. The aggregates **min** and **max** could not be calculated because **.CollapsingMergeTree** does not save values history of the collapsed states

If you need to extract data without aggregation (for example, to check whether rows are present whose newest values match certain conditions), you can use the **FINAL** modifier for the **FROM** clause. This approach is significantly less efficient

## Example of use

:Example data

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

:Creation of the table

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

:Insertion of the data

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

We use two **INSERT** queries to create two different data parts. If we insert the data with one query ClickHouse creates one data part and will not perform any merge ever

:Getting the data

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

UserID	PageViews	Duration	Sign
1	146	5	4324182021466249494

?What do we see and where is collapsing

With two **INSERT** queries, we created 2 data parts. The **SELECT** query was performed in 2 threads, and we got a random order of rows. Collapsing not occurred because there was no merge of the data parts yet. ClickHouse merges data part in an unknown moment of time which we can not predict

:Thus we need aggregation

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

UserID	PageViews	Duration	Sign
185	6	4324182021466249494	1

.If we do not need aggregation and want to force collapsing, we can use **FINAL** modifier for **FROM** clause

```
SELECT * FROM UAct FINAL
```

UserID	PageViews	Duration	Sign
1	185	6	4324182021466249494

.This way of selecting the data is very inefficient. Don't use it for big tables

## VersionedCollapsingMergeTree

:This engine

- .Allows quick writing of object states that are continually changing
- .Deletes old object states in the background. This significantly reduces the volume of storage

.See the section **Collapsing** for details

The engine inherits from **MergeTree** and adds the logic for collapsing rows to the algorithm for merging data parts. **VersionedCollapsingMergeTree** serves the same purpose as **CollapsingMergeTree** but uses a different collapsing algorithm that allows inserting the data in any order with multiple threads. In particular, the **Version** column helps to collapse the rows properly even if they are inserted in the wrong order. In contrast, **CollapsingMergeTree** allows only strictly consecutive insertion

## Creating a Table

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
,[name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
,[name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
(ENGINE = VersionedCollapsingMergeTree(sign, version (
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[... ,SETTINGS name=value]
```

.For a description of query parameters, see the **query description**

### Engine Parameters

```
(VersionedCollapsingMergeTree(sign, version
```

.**sign** — Name of the column with the type of row: **1** is a "state" row, **-1** is a "cancel" row

.The column data type should be **Int8**

.**version** — Name of the column with the version of the object state

.\*The column data type should be **UInt**

### Query Clauses

When creating a **VersionedCollapsingMergeTree** table, the same **clauses** are required as when creating a **MergeTree** table

### Deprecated Method for Creating a Table▼

#### Attention

.Do not use this method in new projects. If possible, switch the old projects to the method described above

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
(ENGINE [=] VersionedCollapsingMergeTree(date-column [, sampling_expression], (primary, key), index_granularity, sign, version (
```

.All of the parameters except **sign** and **version** have the same meaning as in **MergeTree**

.**sign** — Name of the column with the type of row: **1** is a "state" row, **-1** is a "cancel" row •

.Column Data Type — **Int8**

.**version** — Name of the column with the version of the object state •

.\*The column data type should be **UInt**

## Collapsing Data

Consider a situation where you need to save continually changing data for some object. It is reasonable to have one row for an object and update the row whenever there are changes. However, the update operation is expensive and slow for a DBMS because it requires rewriting the data in the storage. Update is not acceptable if you need to write data quickly, but you can write the changes to an object sequentially as follows

Use the **Sign** column when writing the row. If **Sign = 1** it means that the row is a state of an object (let's call it the "state" row). If **Sign = -1** it indicates the cancellation of the state of an object with the same attributes (let's call it the "cancel" row). Also use the **Version** column, which should identify each state of an object with a separate number

For example, we want to calculate how many pages users visited on some site and how long they were there. At some point in time we write the following row with the state of user activity

UserID	PageViews	Duration	Sign	Version
1	1	146	5	4324182021466249494

.At some point later we register the change of user activity and write it with the following two rows

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

The first row cancels the previous state of the object (user). It should copy all of the fields of the canceled state except **Sign**

.The second row contains the current state

Because we need only the last state of user activity, the rows

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

can be deleted, collapsing the invalid (old) state of the object. `VersionedCollapsingMergeTree` does this while merging the data parts

.To find out why we need two rows for each change, see [Algorithm](#)

## Notes on Usage

The program that writes the data should remember the state of an object in order to cancel it. The "cancel" string should be a copy of the "state" string with the opposite `Sign`. This increases the initial size of storage but allows to write the data quickly

Long growing arrays in columns reduce the efficiency of the engine due to the load for writing. The more straightforward the data, the better the efficiency

`SELECT` results depend strongly on the consistency of the history of object changes. Be accurate when preparing data for inserting. You can get unpredictable results with inconsistent data, such as negative values for non-negative metrics like session depth

## Algorithm

When ClickHouse merges data parts, it deletes each pair of rows that have the same primary key and version and different `Sign`. The order of rows does not matter

When ClickHouse inserts data, it orders rows by the primary key. If the `Version` column is not in the primary key, ClickHouse adds it to the primary key implicitly as the last field and uses it for ordering

## Selecting Data

ClickHouse doesn't guarantee that all of the rows with the same primary key will be in the same resulting data part or even on the same physical server. This is true both for writing the data and for subsequent merging of the data parts. In addition, ClickHouse processes `SELECT` queries with multiple threads, and it cannot predict the order of rows in the result. This means that aggregation is required if there is a need to get completely "collapsed" data from a `VersionedCollapsingMergeTree` table

To finalize collapsing, write a query with a `GROUP BY` clause and aggregate functions that account for the sign. For example, to calculate quantity, use `sum(Sign)` instead of `count()`. To calculate the sum of something, use `sum(Sign * x)` instead of `sum(x)`, and add `HAVING sum(Sign) > 0`

The aggregates `count`, `sum` and `avg` can be calculated this way. The aggregate `uniq` can be calculated if an object has at least one non-collapsed state. The aggregates `min` and `max` can't be calculated because `VersionedCollapsingMergeTree` does not save the history of values of collapsed states

If you need to extract the data with "collapsing" but without aggregation (for example, to check whether rows are present whose newest values match certain conditions), you can use the `FINAL` modifier for the `FROM` clause. This approach is inefficient and should not be used with large tables

## Example of Use

:Example data

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

:Creating the table



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

:Inserting the data

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

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

We use two **INSERT** queries to create two different data parts. If we insert the data with a single query, ClickHouse .creates one data part and will never perform any merge

:Getting the data

```
SELECT * FROM UAct
```

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

?What do we see here and where are the collapsed parts

We created two data parts using two **INSERT** queries. The **SELECT** query was performed in two threads, and the .result is a random order of rows

Collapsing did not occur because the data parts have not been merged yet. ClickHouse merges data parts at an .unknown point in time which we cannot predict

:This is why we need aggregation

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

UserID	PageViews	Duration	Version
2	185	6	4324182021466249494

.If we don't need aggregation and want to force collapsing, we can use the **FINAL** modifier for the **FROM** clause

```
SELECT * FROM UAct FINAL
```

UserID	PageViews	Duration	Sign	Version
2	1	185	6	4324182021466249494

.This is a very inefficient way to select data. Don't use it for large tables

## GraphiteMergeTree

This engine is designed for thinning and aggregating/averaging (rollup) **Graphite** data. It may be helpful to developers who want to use ClickHouse as a data store for Graphite

You can use any ClickHouse table engine to store the Graphite data if you don't need rollup, but if you need a rollup use **GraphiteMergeTree**. The engine reduces the volume of storage and increases the efficiency of queries from Graphite

The engine inherits properties from **MergeTree**

## Creating a Table

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

See a detailed description of the **CREATE TABLE** query

A table for the Graphite data should have the following columns for the following data

- Metric name (Graphite sensor). Data type: **String**
- Time of measuring the metric. Data type: **DateTime**
- Value of the metric. Data type: any numeric
- Version of the metric. Data type: any numeric

ClickHouse saves the rows with the highest version or the last written if versions are the same. Other rows are deleted during the merge of data parts

The names of these columns should be set in the rollup configuration

### GraphiteMergeTree parameters

- config\_section** — Name of the section in the configuration file, where are the rules of rollup set

### Query clauses

When creating a **GraphiteMergeTree** table, the same **clauses** are required, as when creating a **MergeTree** table

Deprecated Method for Creating a Table▼

#### Attention

Do not use this method in new projects and, if possible, switch the old projects to the method described above

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
,EventDate Date
,Path String
,Time DateTime
,<Value <Numeric_type
<Version <Numeric_type
...
(ENGINE [=] GraphiteMergeTree(date-column [, sampling_expression], (primary, key), index_granularity, config_section (
```

All of the parameters excepting **config\_section** have the same meaning as in **MergeTree**

.config\_section — Name of the section in the configuration file, where are the rules of rollup set •

## Rollup configuration

The settings for rollup are defined by the graphite\_rollup parameter in the server configuration. The name of the .parameter could be any. You can create several configurations and use them for different tables

:Rollup configuration structure

required-columns
patterns

### Required Columns

- .path\_column\_name — The name of the column storing the metric name (Graphite sensor). Default value: Path •
- .time\_column\_name — The name of the column storing the time of measuring the metric. Default value: Time •
- value\_column\_name — The name of the column storing the value of the metric at the time set in •
- .time\_column\_name. Default value: Value •
- .version\_column\_name — The name of the column storing the version of the metric. Default value: Timestamp •

### Patterns

:Structure of the patterns section

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

#### Attention

:Patterns must be strictly ordered

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

When processing a row, ClickHouse checks the rules in the pattern sections. Each of pattern (including default) sections can contain function parameter for aggregation, retention parameters or both. If the metric name matches the regexp, the rules from the pattern section (or sections) are applied; otherwise, the rules from the default section .are used

:Fields for pattern and default sections

- .regexp- A pattern for the metric name •
- .age – The minimum age of the data in seconds •
- precision- How precisely to define the age of the data in seconds. Should be a divisor for 86400 (seconds in a •
- .(day

**function** - The name of the aggregating function to apply to data whose age falls within the range [age, age + .precision]

## Configuration Example

```
<graphite_rollup>
<version_column_name>Version</version_column_name>
<pattern>
<regex>click_cost</regex>
<function>any</function>
<retention>
<age>0</age>
<precision>5</precision>
<retention/>
<retention>
<age>86400</age>
<precision>60</precision>
<retention/>
<pattern/>
<default>
<function>max</function>
<retention>
<age>0</age>
<precision>60</precision>
<retention/>
<retention>
<age>3600</age>
<precision>300</precision>
<retention/>
<retention>
<age>86400</age>
<precision>3600</precision>
<retention/>
<default/>
</graphite_rollup>
```

## Log Engine Family

These engines were developed for scenarios when you need to write many tables with the small amount of data .(less than 1 million rows

:Engines of the family

StripeLog  
Log  
TinyLog

## Common properties

:Engines

- .Store data on a disk
- .Append data to the end of file when writing
- .Do not support **mutation** operations
- .Do not support indexes
- .This means that **SELECT** queries for ranges of data are not efficient
- .Do not write data atomically

You can get a table with corrupted data if something breaks the write operation, for example, abnormal .server shutdown

# Differences

:The **Log** and **StripeLog** engines support

.Locks for concurrent data access

- 

During **INSERT** query the table is locked, and other queries for reading and writing data both wait for unlocking. If there are no writing data queries, any number of reading data queries can be performed concurrently

.Parallel reading of data

- 

.When reading data ClickHouse uses multiple threads. Each thread processes separated data block

The **Log** engine uses the separate file for each column of the table. The **StripeLog** stores all the data in one file. Thus the **StripeLog** engine uses fewer descriptors in the operating system, but the **Log** engine provides a more efficient reading of the data

The **TinyLog** engine is the simplest in the family and provides the poorest functionality and lowest efficiency. The **TinyLog** engine does not support a parallel reading and concurrent access and stores each column in a separate file. It reads the data slower than both other engines with parallel reading, and it uses almost as many descriptors as the **Log** engine. You can use it in simple low-load scenarios

## StripeLog

This engine belongs to the family of log engines. See the common properties of log engines and their differences in the **Log Engine Family** article

Use this engine in scenarios when you need to write many tables with a small amount of data (less than 1 million rows)

## Creating a Table

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

.See the detailed description of the **CREATE TABLE** query

## Writing the Data

The **StripeLog** engine stores all the columns in one file. For each **INSERT** query, ClickHouse appends the data block to the end of a table file, writing columns one by one

:For each table ClickHouse writes the files

.**data.bin** — Data file

- 

.**index.mrk** — File with marks. Marks contain offsets for each column of each data block inserted

- 

.The **StripeLog** engine does not support the **ALTER UPDATE** and **ALTER DELETE** operations

## Reading the Data

The file with marks allows ClickHouse to parallelize the reading of data. This means that a **SELECT** query returns rows in an unpredictable order. Use the **ORDER BY** clause to sort rows

## Example of Use

:Creating a table

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

:Inserting data

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

.We used two **INSERT** queries to create two data blocks inside the **data.bin** file

ClickHouse uses multiple threads when selecting data. Each thread reads a separate data block and returns resulting rows independently as it finishes. As a result, the order of blocks of rows in the output does not match :the order of the same blocks in the input in most cases. For example

```
SELECT * FROM stripe_log_table
```

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

:(Sorting the results (ascending order by default

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

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

## Log

Engine belongs to the family of log engines. See the common properties of log engines and their differences in the [Log Engine Family](#) article

Log differs from **TinyLog** in that a small file of "marks" resides with the column files. These marks are written on every data block and contain offsets that indicate where to start reading the file in order to skip the specified .number of rows. This makes it possible to read table data in multiple threads

For concurrent data access, the read operations can be performed simultaneously, while write operations block .reads and each other

The Log engine does not support indexes. Similarly, if writing to a table failed, the table is broken, and reading from it returns an error. The Log engine is appropriate for temporary data, write-once tables, and for testing or .demonstration purposes

## TinyLog

Engine belongs to the family of log engines. See the common properties of log engines and their differences in the [Log Engine Family](#) article

- .The simplest table engine, which stores data on a disk
- .Each column is stored in a separate compressed file
- .When writing, data is appended to the end of files

:Concurrent data access is not restricted in any way

- If you are simultaneously reading from a table and writing to it in a different query, the read operation will complete with an error
- If you are writing to a table in multiple queries simultaneously, the data will be broken

The typical way to use this table is write-once: first just write the data one time, then read it as many times as needed

Queries are executed in a single stream. In other words, this engine is intended for relatively small tables ((recommended up to 1,000,000 rows

It makes sense to use this table engine if you have many small tables, since it is simpler than the Log engine ((fewer files need to be opened

The situation when you have a large number of small tables guarantees poor productivity, but may already be used when working with another DBMS, and you may find it easier to switch to using TinyLog types of tables

**.Indexes are not supported**

.In Yandex.Metrica, TinyLog tables are used for intermediary data that is processed in small batches

## Kafka

.This engine works with [Apache Kafka](#)

:Kafka lets you

- .Publish or subscribe to data flows
- .Organize fault-tolerant storage
- .Process streams as they become available

## Creating a Table

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
()ENGINE = Kafka (
SETTINGS
[,kafka_broker_list = 'host:port'
[,...,kafka_topic_list = 'topic1,topic2'
[,kafka_group_name = 'group_name'
[,]kafka_format = 'data_format'
[,kafka_row_delimiter = 'delimiter_symbol']
[, " = kafka_schema]
[,kafka_num_consumers = N]
[<kafka_skip_broken_messages = <0|1]
```

:Required parameters

- .([kafka\\_broker\\_list](#) – A comma-separated list of brokers (for example, [localhost:9092](#)
- .[kafka\\_topic\\_list](#) – A list of Kafka topics
- [kafka\\_group\\_name](#) – A group of Kafka consumers. Reading margins are tracked for each group separately. If you don't want messages to be duplicated in the cluster, use the same group name everywhere
- [kafka\\_format](#) – Message format. Uses the same notation as the SQL [FORMAT](#) function, such as [JSONEachRow](#). For more information, see the [Formats](#) section

:Optional parameters

- .[kafka\\_row\\_delimiter](#) – Delimiter character, which ends the message
- [kafka\\_schema](#) – Parameter that must be used if the format requires a schema definition. For example, [Cap'n Proto](#) requires the path to the schema file and the name of the root [schema.capnp:Message](#) object

- **kafka\_num\_consumers** – The number of consumers per table. Default: 1. Specify more consumers if the throughput of one consumer is insufficient. The total number of consumers should not exceed the number of partitions in the topic, since only one consumer can be assigned per partition
- **kafka\_skip\_broken\_messages** – Kafka message parser mode. If **kafka\_skip\_broken\_messages = 1** then the engine (skips the Kafka messages that can't be parsed (a message equals a row of data

:Examples

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

;SELECT * FROM queue LIMIT 5

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

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

Deprecated Method for Creating a Table▼

Attention

.Do not use this method in new projects. If possible, switch old projects to the method described above

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

## Description

The delivered messages are tracked automatically, so each message in a group is only counted once. If you want to get the data twice, then create a copy of the table with another group name

Groups are flexible and synced on the cluster. For instance, if you have 10 topics and 5 copies of a table in a cluster, then each copy gets 2 topics. If the number of copies changes, the topics are redistributed across the copies automatically. Read more about this at <http://kafka.apache.org/intro>

**SELECT** is not particularly useful for reading messages (except for debugging), because each message can be read only once. It is more practical to create real-time threads using materialized views. To do this

- .1 Use the engine to create a Kafka consumer and consider it a data stream
- .2 Create a table with the desired structure
- .3 Create a materialized view that converts data from the engine and puts it into a previously created table



When the **MATERIALIZED VIEW** joins the engine, it starts collecting data in the background. This allows you to continually receive messages from Kafka and convert them to the required format using **SELECT**. One kafka table can have as many materialized views as you like, they do not read data from the kafka table directly, but receive new records (in blocks), this way you can write to several tables with different detail level. ((with grouping - aggregation and without

:Example

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

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

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

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

To improve performance, received messages are grouped into blocks the size of **max\_insert\_block\_size**. If the block wasn't formed within **stream\_flush\_interval\_ms** milliseconds, the data will be flushed to the table regardless of the completeness of the block

:To stop receiving topic data or to change the conversion logic, detach the materialized view

```
;DETACH TABLE consumer
;ATTACH MATERIALIZED VIEW consumer
```

If you want to change the target table by using **ALTER**, we recommend disabling the material view to avoid discrepancies between the target table and the data from the view

## Configuration

Similar to GraphiteMergeTree, the Kafka engine supports extended configuration using the ClickHouse config file. There are two configuration keys that you can use: global (**kafka**) and topic-level (**kafka\_\***). The global configuration is applied first, and then the topic-level configuration is applied (if it exists)

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

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

For a list of possible configuration options, see the **librdkafka configuration reference**. Use the underscore ( **\_** ) instead of a dot in the ClickHouse configuration. For example, **check\_crcs=true** will be **<check\_crcs>true</check\_crcs>**

## MySQL

.The MySQL engine allows you to perform **SELECT** queries on data that is stored on a remote MySQL server

# Creating a Table

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2]
...
[,INDEX index_name1 expr1 TYPE type1(...) GRANULARITY value1
[,INDEX index_name2 expr2 TYPE type2(...) GRANULARITY value2
;([ENGINE = MySQL('host:port', 'database', 'table', 'user', 'password'[ , replace_query, 'on_duplicate_clause (
```

.See a detailed description of the **CREATE TABLE** query

:The table structure can differ from the original MySQL table structure

Column names should be the same as in the original MySQL table, but you can use just some of these columns and in any order

Column types may differ from those in the original MySQL table. ClickHouse tries to **cast** values to the ClickHouse data types

## Engine Parameters

.**host:port** — MySQL server address

.**database** — Remote database name

.**table** — Remote table name

.**user** — MySQL user

.**password** — User password

**replace\_query** — Flag that converts **INSERT INTO** queries to **REPLACE INTO**. If **replace\_query=1**, the query is substituted

.**on\_duplicate\_clause** — The **ON DUPLICATE KEY on\_duplicate\_clause** expression that is added to the **INSERT** query

Example: **INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1** where **on\_duplicate\_clause** is **UPDATE c2 = c2 + 1**. See the **MySQL documentation** to find which **on\_duplicate\_clause** you can use with the **ON DUPLICATE KEY** clause

To specify **on\_duplicate\_clause** you need to pass **0** to the **replace\_query** parameter. If you simultaneously pass **replace\_query = 1** and **on\_duplicate\_clause**, ClickHouse generates an exception

.Simple **WHERE** clauses such as **=, !=, >, >=, <, <=** are executed on the MySQL server

The rest of the conditions and the **LIMIT** sampling constraint are executed in ClickHouse only after the query to MySQL finishes

## Usage Example

:Table in MySQL

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

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

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

:Table in ClickHouse, retrieving data from the MySQL table

```
CREATE TABLE mysql_table
)
,(float_nullable` Nullable(Float32`
int_id` Int32`
(
('ENGINE = MySQL('localhost:3306', 'test', 'test', 'bayonet', '123

SELECT * FROM mysql_table6

+-----+-----+-----+-----+
| float_nullable | int_id |
+-----+-----+-----+-----+
| NULL | 1 |
+-----+-----+-----+-----+
```

See Also

- [The 'mysql' table function](#)
- [Using MySQL as a source of external dictionary](#)

Distributed

**The Distributed engine does not store data itself** , but allows distributed query processing on multiple .servers  
Reading is automatically parallelized. During a read, the table indexes on remote servers are used, if there are .any  
The Distributed engine accepts parameters: the cluster name in the server's config file, the name of a remote .database, the name of a remote table, and (optionally) a sharding key  
:Example

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

Data will be read from all servers in the 'logs' cluster, from the default.hits table located on every server in the .cluster  
.(Data is not only read, but is partially processed on the remote servers (to the extent that this is possible  
For example, for a query with GROUP BY, data will be aggregated on remote servers, and the intermediate states .of aggregate functions will be sent to the requestor server. Then data will be further aggregated  
  
Instead of the database name, you can use a constant expression that returns a string. For example:  
.(.)currentDatabase

.logs - The cluster name in the server's config file  
:Clusters are set like this

```

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

```

.Here a cluster is defined with the name 'logs' that consists of two shards, each of which contains two replicas  
 Shards refer to the servers that contain different parts of the data (in order to read all the data, you must access  
 .(all the shards  
 .(Replicas are duplicating servers (in order to read all the data, you can access the data on any one of the replicas  
 .Cluster names must not contain dots

**The parameters **host**, **port**, and optionally **user**, **password**, **secure**, **compression** are specified for each :server**

- host** - The address of the remote server. You can use either the domain or the IPv4 or IPv6 address. If you •  
 specify the domain, the server makes a DNS request when it starts, and the result is stored as long as  
 the server is running. If the DNS request fails, the server doesn't start. If you change the DNS record,  
 .restart the server
- port** - The TCP port for messenger activity ('tcp\_port' in the config, usually set to 9000). Do not confuse it •  
 .with http\_port
- user** - Name of the user for connecting to a remote server. Default value: default. This user must have •  
 access to connect to the specified server. Access is configured in the users.xml file. For more  
 ."information, see the section "Access rights
- .password** - The password for connecting to a remote server (not masked). Default value: empty string •
- secure** - Use ssl for connection, usually you also should define **port** = 9440. Server should listen on 9440 •  
 .and have correct certificates
- .compression** - Use data compression. Default value: true •

When specifying replicas, one of the available replicas will be selected for each of the shards when reading. You can configure the algorithm for load balancing (the preference for which replica to access) – see the `.load_balancing'` setting

If the connection with the server is not established, there will be an attempt to connect with a short timeout. If the connection failed, the next replica will be selected, and so on for all the replicas. If the connection attempt failed for all the replicas, the attempt will be repeated the same way, several times

This works in favor of resiliency, but does not provide complete fault tolerance: a remote server might accept the connection, but might not work, or work poorly

You can specify just one of the shards (in this case, query processing should be called remote, rather than distributed) or up to any number of shards. In each shard, you can specify from one to any number of replicas. You can specify a different number of replicas for each shard

You can specify as many clusters as you wish in the configuration

To view your clusters, use the `'system.clusters'` table

The Distributed engine allows working with a cluster like a local server. However, the cluster is inextensible: you must write its configuration in the server config file (even better, for all the cluster's servers)

There is no support for Distributed tables that look at other Distributed tables (except in cases when a Distributed table only has one shard). As an alternative, make the Distributed table look at the "final" tables

The Distributed engine requires writing clusters to the config file. Clusters from the config file are updated on the fly, without restarting the server. If you need to send a query to an unknown set of shards and replicas each time, you don't need to create a Distributed table – use the `'remote'` table function instead. See the section "Table functions"

There are two methods for writing data to a cluster

First, you can define which servers to write which data to, and perform the write directly on each shard. In other words, perform INSERT in the tables that the distributed table "looks at"

This is the most flexible solution – you can use any sharding scheme, which could be non-trivial due to the requirements of the subject area

This is also the most optimal solution, since data can be written to different shards completely independently

Second, you can perform INSERT in a Distributed table. In this case, the table will distribute the inserted data across servers itself

In order to write to a Distributed table, it must have a sharding key set (the last parameter). In addition, if there is only one shard, the write operation works without specifying the sharding key, since it doesn't have any meaning in this case

Each shard can have a weight defined in the config file. By default, the weight is equal to one. Data is distributed across shards in the amount proportional to the shard weight. For example, if there are two shards and the first has a weight of 9 while the second has a weight of 10, the first will be sent 9 / 19 parts of the rows, and the second will be sent 10 / 19

Each shard can have the `'internal_replication'` parameter defined in the config file

If this parameter is set to `'true'`, the write operation selects the first healthy replica and writes data to it. Use this alternative if the Distributed table "looks at" replicated tables. In other words, if the table where data will be written is going to replicate them itself

If it is set to `'false'` (the default), data is written to all replicas. In essence, this means that the Distributed table replicates data itself. This is worse than using replicated tables, because the consistency of replicas is not checked, and over time they will contain slightly different data

To select the shard that a row of data is sent to, the sharding expression is analyzed, and its remainder is taken from dividing it by the total weight of the shards. The row is sent to the shard that corresponds to the half-interval of the remainders from 'prev\_weight' to 'prev\_weights + weight', where 'prev\_weights' is the total weight of the shards with the smallest number, and 'weight' is the weight of this shard. For example, if there are two shards, and the first has a weight of 9 while the second has a weight of 10, the row will be sent to the first shard for the remainders from the range [0, 9), and to the second for the remainders from the range [9, 19).

The sharding expression can be any expression from constants and table columns that returns an integer. For example, you can use the expression 'rand()' for random distribution of data, or 'UserID' for distribution by the remainder from dividing the user's ID (then the data of a single user will reside on a single shard, which simplifies running IN and JOIN by users). If one of the columns is not distributed evenly enough, you can wrap it in a hash function: `intHash64(UserID)`.

A simple remainder from division is a limited solution for sharding and isn't always appropriate. It works for medium and large volumes of data (dozens of servers), but not for very large volumes of data (hundreds of servers or more). In the latter case, use the sharding scheme required by the subject area, rather than using entries in Distributed tables.

SELECT queries are sent to all the shards, and work regardless of how data is distributed across the shards (they can be distributed completely randomly). When you add a new shard, you don't have to transfer the old data to it. You can write new data with a heavier weight – the data will be distributed slightly unevenly, but queries will work correctly and efficiently.

You should be concerned about the sharding scheme in the following cases:

Queries are used that require joining data (IN or JOIN) by a specific key. If data is sharded by this key, you can use local IN or JOIN instead of GLOBAL IN or GLOBAL JOIN, which is much more efficient.

- A large number of servers is used (hundreds or more) with a large number of small queries (queries of individual clients - websites, advertisers, or partners). In order for the small queries to not affect the entire cluster, it makes sense to locate data for a single client on a single shard. Alternatively, as we've done in Yandex.Metrica, you can set up bi-level sharding: divide the entire cluster into "layers", where a layer may consist of multiple shards. Data for a single client is located on a single layer, but shards can be added to a layer as necessary, and data is randomly distributed within them. Distributed tables are created for each layer, and a single shared distributed table is created for global queries.

Data is written asynchronously. For an INSERT to a Distributed table, the data block is just written to the local file system. The data is sent to the remote servers in the background as soon as possible. You should check whether data is sent successfully by checking the list of files (data waiting to be sent) in the table directory: `./var/lib/clickhouse/data/database/table`.

If the server ceased to exist or had a rough restart (for example, after a device failure) after an INSERT to a Distributed table, the inserted data might be lost. If a damaged data part is detected in the table directory, it is transferred to the 'broken' subdirectory and no longer used.

When the `max_parallel_replicas` option is enabled, query processing is parallelized across all replicas within a single shard. For more information, see the section "Settings, max\_parallel\_replicas".

## External Data for Query Processing

ClickHouse allows sending a server the data that is needed for processing a query, together with a SELECT query. This data is put in a temporary table (see the section "Temporary tables") and can be used in the query (for example, in IN operators).

For example, if you have a text file with important user identifiers, you can upload it to the server along with a query that uses filtration by this list.

If you need to run more than one query with a large volume of external data, don't use this feature. It is better to upload the data to the DB ahead of time.

External data can be uploaded using the command-line client (in non-interactive mode), or using the HTTP interface

In the command-line client, you can specify a parameters section in the format

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

.You may have multiple sections like this, for the number of tables being transmitted

.**external** – Marks the beginning of a clause--

.**file** – Path to the file with the table dump, or -, which refers to stdin--

.Only a single table can be retrieved from stdin

.The following parameters are optional: **--name**– Name of the table. If omitted, \_data is used

.**format** – Data format in the file. If omitted, TabSeparated is used--

One of the following parameters is required: **--types** – A list of comma-separated column types. For example:

... ,**UInt64,String**. The columns will be named \_1, \_2

.**structure**– The table structure in the format **UserID UInt64, URL String**. Defines the column names and types--

The files specified in 'file' will be parsed by the format specified in 'format', using the data types specified in 'types' or 'structure'. The table will be uploaded to the server and accessible there as a temporary table with the .name in 'name'

:Examples

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

When using the HTTP interface, external data is passed in the multipart/form-data format. Each table is transmitted as a separate file. The table name is taken from the file name. The 'query\_string' is passed the parameters 'name\_format', 'name\_types', and 'name\_structure', where 'name' is the name of the table that these parameters correspond to. The meaning of the parameters is the same as when using the command-line client

:Example

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

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

.For distributed query processing, the temporary tables are sent to all the remote servers

## Dictionary

.The **Dictionary** engine displays the **dictionary** data as a ClickHouse table

:As an example, consider a dictionary of **products** with the following configuration

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

:Query the dictionary data

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

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

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

.You can use the dictGet\* function to get the dictionary data in this format

This view isn't helpful when you need to get raw data, or when performing a JOIN operation. For these cases, you .can use the Dictionary engine, which displays the dictionary data in a table

:Syntax

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

:Usage example



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

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

.Ok

.rows in set. Elapsed: 0.004 sec 0

.Take a look at what's in the table

```
;select * from products limit 1
```

```
* SELECT
FROM products
LIMIT 1
```

product_id	title
Some item   152689	

.rows in set. Elapsed: 0.006 sec 1

## Merge

The **Merge** engine (not to be confused with **MergeTree**) does not store data itself, but allows reading from any number of other tables simultaneously. Reading is automatically parallelized. Writing to a table is not supported. When reading, the indexes of tables that are actually being read are used, if they exist. The **Merge** engine accepts parameters: the database name and a regular expression for tables.

.Example

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

Data will be read from the tables in the **hits** database that have names that match the regular expression **."^WatchLog**

Instead of the database name, you can use a constant expression that returns a string. For example, **.(currentDatabase**

.Regular expressions — **re2** (supports a subset of PCRE), case-sensitive  
.See the notes about escaping symbols in regular expressions in the "match" section

When selecting tables to read, the **Merge** table itself will not be selected, even if it matches the regex. This is to avoid loops. It is possible to create two **Merge** tables that will endlessly try to read each others' data, but this is not a good idea.

The typical way to use the **Merge** engine is for working with a large number of **TinyLog** tables as if with a single table.

.Example 2

Let's say you have a old table (WatchLog\_old) and decided to change partitioning without moving data to a new table (WatchLog\_new) and you need to see data from both tables

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

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

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

* SELECT
FROM WatchLog
```

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

## Virtual Columns

Virtual columns are columns that are provided by the table engine, regardless of the table definition. In other words, these columns are not specified in **CREATE TABLE**, but they are accessible for **SELECT**

Virtual columns differ from normal columns in the following ways

- They are not specified in table definitions
- Data can't be added to them with **INSERT**
- When using **INSERT** without specifying the list of columns, virtual columns are ignored
- (\* They are not selected when using the asterisk (**SELECT**
- Virtual columns are not shown in **SHOW CREATE TABLE** and **DESC TABLE** queries

The **Merge** type table contains a virtual **\_table** column of the **String** type. (If the table already has a **\_table** column, the virtual column is called **\_table1**; if you already have **\_table1**, it's called **\_table2**, and so on.) It contains the name of the table that data was read from

If the **WHERE/PREWHERE** clause contains conditions for the **\_table** column that do not depend on other table columns (as one of the conjunction elements, or as an entire expression), these conditions are used as an index. The conditions are performed on a data set of table names to read data from, and the read operation will be performed from only those tables that the condition was triggered on

## (File(InputFormat

(.The data source is a file that stores data in one of the supported input formats (TabSeparated, Native, etc

Usage examples

- Data export from ClickHouse to file
- Convert data from one format to another
- Updating data in ClickHouse via editing a file on a disk

## Usage in ClickHouse Server

(File(Format

**Format** should be supported for either **INSERT** and **SELECT**. For the full list of supported formats see **Formats**

ClickHouse does not allow to specify filesystem path for **File**. It will use folder defined by **path** setting in server configuration

When creating table using **File(Format)** it creates empty subdirectory in that folder. When data is written to that table, it's put into **data.Format** file in that subdirectory

You may manually create this subfolder and file in server filesystem and then **ATTACH** it to table information with .matching name, so you can query data from that file

Warning

Be careful with this functionality, because ClickHouse does not keep track of external changes to such files. The .result of simultaneous writes via ClickHouse and outside of ClickHouse is undefined

:Example

:Set up the **file\_engine\_table** table **.1**

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

.By default ClickHouse will create folder **/var/lib/clickhouse/data/default/file\_engine\_table**

:Manually create **/var/lib/clickhouse/data/default/file\_engine\_table/data.TabSeparated** containing **.2**

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

:Query the data **.3**

```
SELECT * FROM file_engine_table
```

name	value
one	1
two	2

# Usage in Clickhouse-local

In **clickhouse-local** File engine accepts file path in addition to **Format**. Default input/output streams can be specified .using numeric or human-readable names like **0** or **stdin**, **1** or **stdout**

:Example

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

# Details of Implementation

Reads can be parallel, but not writes

:Not supported

**ALTER**  
**SELECT ... SAMPLE**

Indices

Replication

- 
- 
- 
- 
- 
- 

# Null

.When writing to a Null table, data is ignored. When reading from a Null table, the response is empty

However, you can create a materialized view on a Null table. So the data written to the table will end up in the .view

# Set

A data set that is always in RAM. It is intended for use on the right side of the IN operator (see the section "IN .("operators

You can use INSERT to insert data in the table. New elements will be added to the data set, while duplicates will be ignored

But you can't perform SELECT from the table. The only way to retrieve data is by using it in the right half of the IN operator

Data is always located in RAM. For INSERT, the blocks of inserted data are also written to the directory of tables on the disk. When starting the server, this data is loaded to RAM. In other words, after restarting, the data remains in place

For a rough server restart, the block of data on the disk might be lost or damaged. In the latter case, you may need to manually delete the file with damaged data

## Join

A prepared data structure for JOIN that is always located in RAM

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

Engine parameters: **ANY|ALL** – strictness; **LEFT|INNER** – type. For more information, see the **JOIN Clause** section  
These parameters are set without quotes and must match the JOIN that the table will be used for. k1, k2, ... are the key columns from the USING clause that the join will be made on

The table can't be used for GLOBAL JOINS

You can use INSERT to add data to the table, similar to the Set engine. For ANY, data for duplicated keys will be ignored. For ALL, it will be counted

You can't perform SELECT directly from the table. There are two ways to retrieve data from table with Join engine  
\* use it as the "right-hand" table for JOIN \* use **joinGet** function, which allows to extract data from Join table with dictionary-like syntax

Storing data on the disk is the same as for the Set engine

You can customize several settings when creating JOIN table with the following syntax

```
;CREATE TABLE join_any_left_null ( ... ) ENGINE = Join(ANY, LEFT, ...) SETTINGS join_use_nulls = 1
```

The following settings are supported by JOIN engine

**join\_use\_nulls**  
**max\_rows\_in\_join**  
**max\_bytes\_in\_join**  
**join\_overflow\_mode**  
**join\_any\_take\_last\_row**

- 
- 
- 
- 
- 

## (URL(URL, Format

Manages data on a remote HTTP/HTTPS server. This engine is similar to the **File** engine

## Using the engine in the ClickHouse server

The **format** must be one that ClickHouse can use in **SELECT** queries and, if necessary, in **INSERTs**. For the full list of supported formats, see **Formats**

The **URL** must conform to the structure of a Uniform Resource Locator. The specified URL must point to a server that uses HTTP or HTTPS. This does not require any additional headers for getting a response from the server

,**INSERT** and **SELECT** queries are transformed to **POST** and **GET** requests respectively. For processing **POST** requests, the remote server must support **Chunked transfer encoding**

## :Example

: Create a **url\_engine\_table** table on the server **.1**

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

Create a basic HTTP server using the standard Python 3 tools and **.2**  
:start it

```
from http.server import BaseHTTPRequestHandler, HTTPServer

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

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

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

```
python3 server.py
```

:Request data **.3**

```
SELECT * FROM url_engine_table
```

```
|_word_|_value_|
| Hello |    1 |
| World |    2 |
|_____|_
```

## Details of Implementation

Reads and writes can be parallel

:Not supported

.**ALTER** and **SELECT...SAMPLE** operations

.Indexes

.Replication

- 
- 
- 
- 
- 

## View

Used for implementing views (for more information, see the **CREATE VIEW query**). It does not store data, but only stores the specified **SELECT** query. When reading from a table, it runs this query (and deletes all unnecessary .(columns from the query

## MaterializedView

Used for implementing materialized views (for more information, see **CREATE TABLE**). For storing data, it uses a .different engine that was specified when creating the view. When reading from a table, it just uses this engine

## Memory

The Memory engine stores data in RAM, in uncompressed form. Data is stored in exactly the same form as it is received when read. In other words, reading from this table is completely free  
.Concurrent data access is synchronized. Locks are short: read and write operations don't block each other  
.Indexes are not supported. Reading is parallelized  
Maximal productivity (over 10 GB/sec) is reached on simple queries, because there is no reading from the disk, decompressing, or deserializing data. (We should note that in many cases, the productivity of the MergeTree (.engine is almost as high  
.When restarting a server, data disappears from the table and the table becomes empty  
Normally, using this table engine is not justified. However, it can be used for tests, and for tasks where maximum (.speed is required on a relatively small number of rows (up to approximately 100,000,000

The Memory engine is used by the system for temporary tables with external query data (see the section .("External data for processing a query"), and for implementing GLOBAL IN (see the section "IN operators

## Buffer

Buffers the data to write in RAM, periodically flushing it to another table. During the read operation, data is read .from the buffer and the other table simultaneously

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

Engine parameters:database, table – The table to flush data to. Instead of the database name, you can use a constant expression that returns a string. num\_layers – Parallelism layer. Physically, the table will be represented as 'num\_layers' of independent buffers. Recommended value: 16. min\_time, max\_time, min\_rows, max\_rows, .min\_bytes, and max\_bytes are conditions for flushing data from the buffer

Data is flushed from the buffer and written to the destination table if all the 'min' conditions or at least one 'max' condition are met.min\_time, max\_time – Condition for the time in seconds from the moment of the first write to the buffer. min\_rows, max\_rows – Condition for the number of rows in the buffer. min\_bytes, max\_bytes – Condition for .the number of bytes in the buffer

During the write operation, data is inserted to a 'num\_layers' number of random buffers. Or, if the data part to insert is large enough (greater than 'max\_rows' or 'max\_bytes'), it is written directly to the destination table, .omitting the buffer

The conditions for flushing the data are calculated separately for each of the 'num\_layers' buffers. For example, if .num\_layers = 16 and max\_bytes = 100000000, the maximum RAM consumption is 1.6 GB

:Example

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

Creating a 'merge.hits\_buffer' table with the same structure as 'merge.hits' and using the Buffer engine. When writing to this table, data is buffered in RAM and later written to the 'merge.hits' table. 16 buffers are created. The data in each of them is flushed if either 100 seconds have passed, or one million rows have been written, or 100 MB of data have been written; or if simultaneously 10 seconds have passed and 10,000 rows and 10 MB of data have been written. For example, if just one row has been written, after 100 seconds it will be flushed, no matter .what. But if many rows have been written, the data will be flushed sooner

When the server is stopped, with DROP TABLE or DETACH TABLE, buffer data is also flushed to the destination .table

You can set empty strings in single quotation marks for the database and table name. This indicates the absence of a destination table. In this case, when the data flush conditions are reached, the buffer is simply cleared. This .may be useful for keeping a window of data in memory

When reading from a Buffer table, data is processed both from the buffer and from the destination table (if there .(is one

Note that the Buffer tables does not support an index. In other words, data in the buffer is fully scanned, which (.might be slow for large buffers. (For data in a subordinate table, the index that it supports will be used

If the set of columns in the Buffer table doesn't match the set of columns in a subordinate table, a subset of columns that exist in both tables is inserted

If the types don't match for one of the columns in the Buffer table and a subordinate table, an error message is entered in the server log and the buffer is cleared  
.The same thing happens if the subordinate table doesn't exist when the buffer is flushed

If you need to run ALTER for a subordinate table and the Buffer table, we recommend first deleting the Buffer table, running ALTER for the subordinate table, then creating the Buffer table again

.If the server is restarted abnormally, the data in the buffer is lost

FINAL and SAMPLE do not work correctly for Buffer tables. These conditions are passed to the destination table, but are not used for processing data in the buffer. If these features are required we recommend only using the Buffer table for writing, while reading from the destination table

When adding data to a Buffer, one of the buffers is locked. This causes delays if a read operation is simultaneously being performed from the table

Data that is inserted to a Buffer table may end up in the subordinate table in a different order and in different blocks. Because of this, a Buffer table is difficult to use for writing to a CollapsingMergeTree correctly. To avoid problems, you can set 'num\_layers' to 1

If the destination table is replicated, some expected characteristics of replicated tables are lost when writing to a Buffer table. The random changes to the order of rows and sizes of data parts cause data deduplication to quit working, which means it is not possible to have a reliable 'exactly once' write to replicated tables

.Due to these disadvantages, we can only recommend using a Buffer table in rare cases

A Buffer table is used when too many INSERTs are received from a large number of servers over a unit of time and data can't be buffered before insertion, which means the INSERTs can't run fast enough

Note that it doesn't make sense to insert data one row at a time, even for Buffer tables. This will only produce a speed of a few thousand rows per second, while inserting larger blocks of data can produce over a million rows per second (see the section "Performance

## JDBC

.Allows ClickHouse to connect to external databases via **JDBC**

To implement the JDBC connection, ClickHouse uses the separate program **clickhouse-jdbc-bridge** that should run as a daemon

.This engine supports the **Nullable** data type

## Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(ENGINE = JDBC(dbms_uri, external_database, external_table
```

### Engine Parameters

.**dbms\_uri** — URI of an external DBMS

.<Format: **jdbc:<driver\_name>://<host\_name>:<port>/?user=<username>&password=<password>**

.Example for MySQL: **jdbc:mysql://localhost:3306/?user=root&password=root**

.**external\_database** — Database in an external DBMS

.**external\_table** — Name of the table in **external\_database**

## Usage Example

:Creating a table in MySQL server by connecting directly with it's console client

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

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

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

:Creating a table in ClickHouse server and selecting data from it

```
('CREATE TABLE jdbc_table ENGINE JDBC('jdbc:mysql://localhost:3306/?user=root&password=root', 'test', 'test
.Ok

DESCRIBE TABLE jdbc_table

+-----+-----+-----+-----+
| name | type | default_type | default_expression |
+-----+-----+-----+-----+
| int_id | Int32 |
| (int_nullable | Nullable(Int32 |
| float | Float32 |
| (float_nullable | Nullable(Float32 |
+-----+-----+-----+-----+

.rows in set. Elapsed: 0.031 sec 10

* SELECT
FROM jdbc_table

+-----+-----+-----+-----+
| int_id | int_nullable | float | float_nullable |
+-----+-----+-----+-----+
| NULL | 2 | NULL | 1 |
+-----+-----+-----+-----+

.rows in set. Elapsed: 0.055 sec 1
```

## See Also

[JDBC table function](#)



## ODBC

.Allows ClickHouse to connect to external databases via [ODBC](#)

To safely implement ODBC connections, ClickHouse uses a separate program [clickhouse-odbc-bridge](#). If the ODBC driver is loaded directly from [clickhouse-server](#), driver problems can crash the ClickHouse server. ClickHouse automatically starts [clickhouse-odbc-bridge](#) when it is required. The ODBC bridge program is installed from the same .package as the [clickhouse-server](#)

.This engine supports the [Nullable](#) data type

## Creating a Table





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

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

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

:Table in ClickHouse, retrieving data from the MySQL table

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

SELECT * FROM odbc_t

┌─int_id─┴─float_nullable─┐
| NULL | 1 |
└──────────┴──────────┘
```

See Also

- ODBC external dictionaries
- ODBC table function
- 
- 

SQL Reference

- SELECT
- INSERT INTO
- CREATE
- ALTER
- Other types of queries
- 
- 
- 
- 
- 

SELECT Queries Syntax

.SELECT performs data retrieval

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

.All the clauses are optional, except for the required list of expressions immediately after SELECT  
.The clauses below are described in almost the same order as in the query execution conveyor

If the query omits the **DISTINCT**, **GROUP BY** and **ORDER BY** clauses and the **IN** and **JOIN** subqueries, the query will be completely stream processed, using O(1) amount of RAM  
Otherwise, the query might consume a lot of RAM if the appropriate restrictions are not specified:  
**max\_memory\_usage**, **max\_rows\_to\_group\_by**, **max\_rows\_to\_sort**, **max\_rows\_in\_distinct**, **max\_bytes\_in\_distinct**, **max\_rows\_in\_set**, **max\_bytes\_in\_set**, **max\_rows\_in\_join**, **max\_bytes\_in\_join**, **max\_bytes\_before\_external\_sort**, **max\_bytes\_before\_external\_group\_by**.  
For more information, see the section "Settings". It is possible to use external sorting (saving temporary tables to a disk) and external aggregation. **The system does not have "merge join"**

## FROM Clause

.If the FROM clause is omitted, data will be read from the **system.one** table  
The 'system.one' table contains exactly one row (this table fulfills the same purpose as the DUAL table found in other DBMSs)

The FROM clause specifies the table to read data from, or a subquery, or a table function; ARRAY JOIN and the (regular JOIN may also be included (see below)

.Instead of a table, the SELECT subquery may be specified in brackets  
.In this case, the subquery processing pipeline will be built into the processing pipeline of an external query  
In contrast to standard SQL, a synonym does not need to be specified after a subquery. For compatibility, it is possible to write 'AS name' after a subquery, but the specified name isn't used anywhere

.A table function may be specified instead of a table. For more information, see the section "Table functions"

To execute a query, all the columns listed in the query are extracted from the appropriate table. Any columns not needed for the external query are thrown out of the subqueries  
If a query does not list any columns (for example, SELECT count() FROM t), some column is extracted from the table anyway (the smallest one is preferred), in order to calculate the number of rows

The FINAL modifier can be used only for a SELECT from a CollapsingMergeTree table. When you specify FINAL, data is selected fully "collapsed". Keep in mind that using FINAL leads to a selection that includes columns related to the primary key, in addition to the columns specified in the SELECT. Additionally, the query will be executed in a single stream, and data will be merged during query execution. This means that when using FINAL, the query is processed more slowly. In most cases, you should avoid using FINAL. For more information, see the section ""CollapsingMergeTree engine"

## SAMPLE Clause

.The **SAMPLE** clause allows for approximated query processing

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

:Approximated query processing can be useful in the following cases

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

Note

You can only use sampling with the tables in the MergeTree family, and only if the sampling expression was specified during table creation (see MergeTree engine

:The features of data sampling are listed below

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

:For the SAMPLE clause the following syntax is supported

Description	SAMPLE Clause Syntax
Here <b>k</b> is the number from 0 to 1	<b>SAMPLE k</b>
The query is executed on <b>k</b> fraction of data. For example, <b>SAMPLE 0.1</b> runs the query on 10% of data. <a href="#">Read more</a>	
<b>SAMPLE n</b> Here <b>n</b> is a sufficiently large integer. The query is executed on a sample of at least <b>n</b> rows (but not significantly more than this). For example, <b>SAMPLE 10000000</b> runs the query on a minimum of 10,000,000 rows. <a href="#">Read more</a>	
<b>SAMPLE k OFFSET m</b> Here <b>k</b> and <b>m</b> are the numbers from 0 to 1. The query is executed on a sample of <b>k</b> fraction of the data. The data used for the sample is offset by <b>m</b> fraction. <a href="#">Read more</a>	
<b>SAMPLE k</b>	
Here <b>k</b> is the number from 0 to 1 (both fractional and decimal notations are supported). For example, <b>SAMPLE 1/2</b> or <b>SAMPLE 0.5</b>	

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

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

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

SAMPLE n

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

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

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

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

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

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

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

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

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

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

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

## SAMPLE k OFFSET m

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

### Example 1

```
SAMPLE 1/10
```

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

[-----++]

### Example 2

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

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

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

## ARRAY JOIN Clause

Allows executing `JOIN` with an array or nested data structure. The intent is similar to the `arrayJoin` function, but its functionality is broader

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

You can specify only a single `ARRAY JOIN` clause in a query

The query execution order is optimized when running **ARRAY JOIN**. Although **ARRAY JOIN** must always be specified before the **WHERE/PREWHERE** clause, it can be performed either before **WHERE/PREWHERE** (if the result is needed in this clause), or after completing it (to reduce the volume of calculations). The processing order is controlled by the .query optimizer

:Supported types of **ARRAY JOIN** are listed below

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

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

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

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

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

:The example below uses the **ARRAY JOIN** clause

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

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

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

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

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

### Using Aliases

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

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

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

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

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

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

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

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

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

:The example below uses the **arrayEnumerate** function

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

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

## ARRAY JOIN With Nested Data Structure

:**ARRAY JOIN**`` also works with **nested data structures**. Example

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

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

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

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

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

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

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

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

:This variation also makes sense

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

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

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

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



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

:Example of using the `arrayEnumerate` function

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

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

## JOIN Clause

.Joins the data in the normal `SQL JOIN` sense

Note

.Not related to `ARRAY JOIN`

```
<SELECT <expr_list>
<FROM <left_subquery>
<GLOBAL> [<ANY>|<ALL>] [<INNER>|<LEFT>|<RIGHT>|<FULL>|<CROSS>] [<OUTER>] JOIN <right_subquery>]
... (<ON <expr_list>)|(<USING <column_list>)
```

The table names can be specified instead of `<left_subquery>` and `<right_subquery>`. This is equivalent to the `SELECT * .FROM table` subquery, except in a special case when the table has the `Join` engine – an array prepared for joining

## Supported Types of JOIN

(`INNER JOIN` (or `JOIN`

(`LEFT JOIN` (or `LEFT OUTER JOIN`

(`RIGHT JOIN` (or `RIGHT OUTER JOIN`

(`FULL JOIN` (or `FULL OUTER JOIN`

( , `CROSS JOIN` (or

- 
- 
- 
- 
- 

.See the standard `SQL JOIN` description

## Multiple JOIN

Performing queries, ClickHouse rewrites multiple joins into the sequence of two-table joins. If there are four tables for join ClickHouse joins the first and the second, then joins the result with the third table, and at the last step, it joins the fourth one

If a query contains `WHERE` clause, ClickHouse tries to push down filters from this clause into the intermediate join. .If it cannot apply the filter to each intermediate join, ClickHouse applies the filters after all joins are completed

:We recommend the `JOIN ON` or `JOIN USING` syntax for creating a query. For example

```
SELECT * FROM t1 JOIN t2 ON t1.a = t2.a JOIN t3 ON t1.a = t3.a
```

Also, you can use comma separated list of tables for join. Works only with the `.allow_experimental_cross_to_join_conversion = 1` setting

```
`For example, `SELECT * FROM t1, t2, t3 WHERE t1.a = t2.a AND t1.a = t3.a
```

.Don't mix these syntaxes

ClickHouse doesn't support the syntax with commas directly, so we don't recommend to use it. The algorithm tries to rewrite the query in terms of **CROSS** and **INNER JOIN** clauses and then proceeds the query processing. When rewriting the query, ClickHouse tries to optimize performance and memory consumption. By default, ClickHouse treats comma as an **INNER JOIN** clause and converts it to **CROSS JOIN** when the algorithm cannot guaranty that **INNER JOIN** returns required data

## Strictness

- **ALL** — If the right table has several matching rows, ClickHouse creates a **Cartesian product** from matching rows. This is the normal **JOIN** behavior for standard SQL
- **ANY** — If the right table has several matching rows, only the first one found is joined. If the right table has only one matching row, the results of queries with **ANY** and **ALL** keywords are the same
- **ASOF** — For joining sequences with a non-exact match. Usage of **ASOF JOIN** is described below

## ASOF JOIN Usage

**ASOF JOIN** is useful when you need to join records that have no exact match. For example, consider the following tables

```

table_1      table_2
event | ev_time | user_id    event | ev_time | user_id
a name="../../../query_language/insert_into/"></a><a name="../../../query_language/insert_into/"></a><a name="../../../query_language/insert_into/"></a><a name="../../../query_language/insert_into/"></a><a name="../../../query_language/insert_into/"></a><a name="query_language/insert_into/"></a><a name="../../../query_language/insert_into/"></a><a name="insert_into/"></a>
INSERT ###
.Adding data
:Basic query format
sql `` `
... ,(INSERT INTO [db.]table [(c1, c2, c3)] VALUES (v11, v12, v13), (v21, v22, v23
```

The query can specify a list of columns to insert [(c1, c2, c3)]. In this case, the rest of the columns are filled with

- The values calculated from the **DEFAULT** expressions specified in the table definition
- Zeros and empty strings, if **DEFAULT** expressions are not defined

.If **strict insert defaults=1**, columns that do not have **DEFAULT** defined must be listed in the query

Data can be passed to the INSERT in any **format** supported by ClickHouse. The format must be specified explicitly in the query

```
INSERT INTO [db.]table [(c1, c2, c3)] FORMAT format name data set
```

:For example, the following query format is identical to the basic version of INSERT ... VALUES

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

ClickHouse removes all spaces and one line feed (if there is one) before the data. When forming a query, we recommend putting the data on a new line after the query operators (this is important if the data begins with .(spaces

### :Example

```
INSERT INTO t FORMAT TabSeparated
!Hello, world 11
Qwerty 22
```

You can insert data separately from the query by using the command-line client or the HTTP interface. For more information, see the section "[Interfaces](#)"

## Inserting The Results of **SELECT**

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

Columns are mapped according to their position in the **SELECT** clause. However, their names in the **SELECT** expression and the table for **INSERT** may differ. If necessary, type casting is performed

None of the data formats except Values allow setting values to expressions such as **now()**, **1 + 2**, and so on. The Values format allows limited use of expressions, but this is not recommended, because in this case inefficient code is used for their execution

Other queries for modifying data parts are not supported: **UPDATE**, **DELETE**, **REPLACE**, **MERGE**, **UPSERT**, **INSERT UPDATE**. However, you can delete old data using **ALTER TABLE ... DROP PARTITION**

## Performance Considerations

**INSERT** sorts the input data by primary key and splits them into partitions by month. If you insert data for mixed months, it can significantly reduce the performance of the **INSERT** query. To avoid this

Add data in fairly large batches, such as 100,000 rows at a time

Group data by month before uploading it to ClickHouse

Performance will not decrease if

Data is added in real time

You upload data that is usually sorted by time

## CREATE DATABASE

Creating db\_name databases

```
[CREATE DATABASE [IF NOT EXISTS] db_name [ON CLUSTER cluster
```

A database is just a directory for tables

If **IF NOT EXISTS** is included, the query won't return an error if the database already exists

## CREATE TABLE

The **CREATE TABLE** query can have several forms

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [compression_codec] [TTL expr1
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [compression_codec] [TTL expr2
...
ENGINE = engine (
```

Creates a table named 'name' in the 'db' database or the current database if 'db' is not set, with the structure specified in brackets and the 'engine' engine

The structure of the table is a list of column descriptions. If indexes are supported by the engine, they are indicated as parameters for the table engine

A column description is **name type** in the simplest case. Example: **RegionID UInt32**

(Expressions can also be defined for default values (see below

```
[CREATE TABLE [IF NOT EXISTS] [db.]table_name AS [db2.]name2 [ENGINE = engine
```

Creates a table with the same structure as another table. You can specify a different engine for the table. If the engine is not specified, the same engine will be used as for the **db2.name2** table

```
... CREATE TABLE [IF NOT EXISTS] [db.]table_name ENGINE = engine AS SELECT
```

Creates a table with a structure like the result of the `SELECT` query, with the 'engine' engine, and fills it with data.  
.from `SELECT`

In all cases, if `IF NOT EXISTS` is specified, the query won't return an error if the table already exists. In this case, the query won't do anything

There can be other clauses after the `ENGINE` clause in the query. See detailed documentation on how to create tables in the descriptions of [table engines](#)

## Default Values

The column description can specify an expression for a default value, in one of the following ways: `DEFAULT expr`, `MATERIALIZED expr`, `ALIAS expr`  
(Example: `URLDomain String DEFAULT domain(URL`

If an expression for the default value is not defined, the default values will be set to zeros for numbers, empty strings for strings, empty arrays for arrays, and `0000-00-00` for dates or `0000-00-00 00:00:00` for dates with time.  
.NULLs are not supported

If the default expression is defined, the column type is optional. If there isn't an explicitly defined type, the default expression type is used. Example: `EventDate DEFAULT toDate(EventTime)` – the 'Date' type will be used for the 'EventDate' column

If the data type and default expression are defined explicitly, this expression will be cast to the specified type (using type casting functions. Example: `Hits UInt32 DEFAULT 0` means the same thing as `Hits UInt32 DEFAULT toUInt32(0`

Default expressions may be defined as an arbitrary expression from table constants and columns. When creating and changing the table structure, it checks that expressions don't contain loops. For `INSERT`, it checks that expressions are resolvable – that all columns they can be calculated from have been passed

### `DEFAULT expr`

Normal default value. If the `INSERT` query doesn't specify the corresponding column, it will be filled in by computing the corresponding expression

### `MATERIALIZED expr`

Materialized expression. Such a column can't be specified for `INSERT`, because it is always calculated  
.For an `INSERT` without a list of columns, these columns are not considered

In addition, this column is not substituted when using an asterisk in a `SELECT` query. This is to preserve the invariant that the dump obtained using `SELECT *` can be inserted back into the table using `INSERT` without specifying the list of columns

### `ALIAS expr`

Synonym. Such a column isn't stored in the table at all  
.Its values can't be inserted in a table, and it is not substituted when using an asterisk in a `SELECT` query  
.It can be used in `SELECT`s if the alias is expanded during query parsing

When using the `ALTER` query to add new columns, old data for these columns is not written. Instead, when reading old data that does not have values for the new columns, expressions are computed on the fly by default. However, if running the expressions requires different columns that are not indicated in the query, these columns will additionally be read, but only for the blocks of data that need it

If you add a new column to a table but later change its default expression, the values used for old data will change (for data where values were not stored on the disk). Note that when running background merges, data for columns that are missing in one of the merging parts is written to the merged part

It is not possible to set default values for elements in nested data structures

## TTL expression

Can be specified only for MergeTree-family tables. An expression for setting storage time for values. It must depends on **Date** or **DateTime** column and has one **Date** or **DateTime** column as a result. Example

**TTL date + INTERVAL 1 DAY**

You are not allowed to set TTL for key columns. For more details, see [TTL for columns and tables](#)

## Column Compression Codecs

Besides default data compression, defined in [server settings](#), per-column specification is also available

Supported compression algorithms

**NONE** - no compression for data applied

**LZ4**

**LZ4HC(level)** - (level) - LZ4\_HC compression algorithm with defined level

Possible **level** range: [3, 12]. Default value: 9. Greater values stands for better compression and higher CPU

usage. Recommended value range: [4,9

**ZSTD(level)** - ZSTD compression algorithm with defined **level**. Possible **level** value range: [1, 22]. Default value:

1

Greater values stands for better compression and higher CPU usage

**Delta(delta\_bytes)** - compression approach when raw values are replace with difference of two neighbour

values. Up to **delta\_bytes** are used for storing delta value

Possible **delta\_bytes** values: 1, 2, 4, 8. Default value for delta bytes is **sizeof(type)**, if it is equals to 1, 2, 4, 8 and

equals to 1 otherwise

Syntax example

```
CREATE TABLE codec_example
)
/* dt Date CODEC(ZSTD), /* используется уровень сжатия по-умолчанию
,(ts DateTime CODEC(LZ4HC
,(float_value Float32 CODEC(NONE
((double_value Float64 CODEC(LZ4HC(9
(
ENGINE = MergeTree
()PARTITION BY tuple
ORDER BY dt
```

Codecs can be combined in a pipeline. Default table codec is not included into pipeline (if it should be applied to a column, you have to specify it explicitly in pipeline). Example below shows an optimization approach for storing timeseries metrics

Usually, values for particular metric, stored in **path** does not differ significantly from point to point. Using delta-encoding allows to reduce disk space usage significantly

```
CREATE TABLE timeseries_example
)
,dt Date
,ts DateTime
,path String
(value Float32 CODEC(Delta, ZSTD
(
ENGINE = MergeTree
PARTITION BY dt
(ORDER BY (path, ts
```

## Temporary Tables

ClickHouse supports temporary tables which have the following characteristics

Temporary tables disappear when the session ends, including if the connection is lost

A temporary table use the Memory engine only

The DB can't be specified for a temporary table. It is created outside of databases

- If a temporary table has the same name as another one and a query specifies the table name without specifying the DB, the temporary table will be used
- For distributed query processing, temporary tables used in a query are passed to remote servers

:To create a temporary table, use the following syntax

```
[CREATE TEMPORARY TABLE [IF NOT EXISTS] table_name [ON CLUSTER cluster]
)
[,name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1]
[,name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2]
...
(
```

In most cases, temporary tables are not created manually, but when using external data for a query, or for distributed **(GLOBAL) IN**. For more information, see the appropriate sections

## (Distributed DDL queries (ON CLUSTER clause

.The **CREATE**, **DROP**, **ALTER**, and **RENAME** queries support distributed execution on a cluster

:For example, the following query creates the **all\_hits Distributed** table on each host in **cluster**

```
(CREATE TABLE IF NOT EXISTS all_hits ON CLUSTER cluster (p Date, i Int32) ENGINE = Distributed(cluster, default, hits
```

In order to run these queries correctly, each host must have the same cluster definition (to simplify syncing configs, you can use substitutions from ZooKeeper). They must also connect to the ZooKeeper servers. The local version of the query will eventually be implemented on each host in the cluster, even if some hosts are currently not available. The order for executing queries within a single host is guaranteed. **ALTER** queries are not yet supported for replicated tables

## CREATE VIEW

```
... CREATE [MATERIALIZED] VIEW [IF NOT EXISTS] [db.]table_name [TO[db.]name] [ENGINE = engine] [POPULATE] AS SELECT
```

.Creates a view. There are two types of views: normal and MATERIALIZED

Normal views don't store any data, but just perform a read from another table. In other words, a normal view is nothing more than a saved query. When reading from a view, this saved query is used as a subquery in the FROM clause

:As an example, assume you've created a view

```
... CREATE VIEW view AS SELECT
```

:and written a query

```
SELECT a, b, c FROM view
```

:This query is fully equivalent to using the subquery

```
(... SELECT a, b, c FROM (SELECT
```

.Materialized views store data transformed by the corresponding SELECT query

.When creating a materialized view, you must specify ENGINE – the table engine for storing data

A materialized view is arranged as follows: when inserting data to the table specified in SELECT, part of the inserted data is converted by this SELECT query, and the result is inserted in the view

If you specify POPULATE, the existing table data is inserted in the view when creating it, as if making a **CREATE TABLE ... AS SELECT ...**. Otherwise, the query contains only the data inserted in the table after creating the view. We don't recommend using POPULATE, since data inserted in the table during the view creation will not be inserted in it.

A **SELECT** query can contain **DISTINCT**, **GROUP BY**, **ORDER BY**, **LIMIT**... Note that the corresponding conversions are performed independently on each block of inserted data. For example, if **GROUP BY** is set, data is aggregated during insertion, but only within a single packet of inserted data. The data won't be further aggregated. The exception is when using an **ENGINE** that independently performs data aggregation, such as **SummingMergeTree**

The execution of **ALTER** queries on materialized views has not been fully developed, so they might be inconvenient. If the materialized view uses the construction **TO [db.]name**, you can **DETACH** the view, run **ALTER** for the target table, and then **ATTACH** the previously detached (**DETACH**) view

Views look the same as normal tables. For example, they are listed in the result of the **SHOW TABLES** query

There isn't a separate query for deleting views. To delete a view, use **DROP TABLE**

## ALTER

The **ALTER** query is only supported for **\*MergeTree** tables, as well as **Merge** and **Distributed**. The query has several variations

## Column Manipulations

Changing the table structure

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

In the query, specify a list of one or more comma-separated actions

Each action is an operation on a column

The following actions are supported

- ADD COLUMN** — Adds a new column to the table
- DROP COLUMN** — Deletes the column
- CLEAR COLUMN** — Resets column values
- COMMENT COLUMN** — Adds a text comment to the column
- MODIFY COLUMN** — Changes column's type and/or default expression

These actions are described in detail below

### ADD COLUMN

```
[ADD COLUMN [IF NOT EXISTS] name [type] [default_expr] [AFTER name_after]
```

Adds a new column to the table with the specified **name**, **type**, and **default\_expr** (see the section **Default expressions**)

If the **IF NOT EXISTS** clause is included, the query won't return an error if the column already exists. If you specify **AFTER name\_after** (the name of another column), the column is added after the specified one in the list of table columns. Otherwise, the column is added to the end of the table. Note that there is no way to add a column to the beginning of a table. For a chain of actions, **name\_after** can be the name of a column that is added in one of the previous actions

Adding a column just changes the table structure, without performing any actions with data. The data doesn't appear on the disk after **ALTER**. If the data is missing for a column when reading from the table, it is filled in with default values (by performing the default expression if there is one, or using zeros or empty strings). The column appears on the disk after merging data parts (see **MergeTree**)

This approach allows us to complete the **ALTER** query instantly, without increasing the volume of old data

Example

```
ALTER TABLE visits ADD COLUMN browser String AFTER user_id
```

### DROP COLUMN

```
DROP COLUMN [IF EXISTS] name
```

Deletes the column with the name `name`. If the `IF EXISTS` clause is specified, the query won't return an error if the column doesn't exist

.Deletes data from the file system. Since this deletes entire files, the query is completed almost instantly

:Example

```
ALTER TABLE visits DROP COLUMN browser
```

## CLEAR COLUMN

```
CLEAR COLUMN [IF EXISTS] name IN PARTITION partition_name
```

Resets all data in a column for a specified partition. Read more about setting the partition name in the section

.[How to specify the partition expression](#)

.If the `IF EXISTS` clause is specified, the query won't return an error if the column doesn't exist

:Example

```
()ALTER TABLE visits CLEAR COLUMN browser IN PARTITION tuple
```

## COMMENT COLUMN

```
'COMMENT COLUMN [IF EXISTS] name 'comment
```

Adds a comment to the column. If the `IF EXISTS` clause is specified, the query won't return an error if the column doesn't exist

Each column can have one comment. If a comment already exists for the column, a new comment overwrites the previous comment

.Comments are stored in the `comment_expression` column returned by the `DESCRIBE TABLE` query

:Example

```
'.ALTER TABLE visits COMMENT COLUMN browser 'The table shows the browser used for accessing the site
```

## MODIFY COLUMN

```
[MODIFY COLUMN [IF EXISTS] name [type] [default_expr
```

This query changes the `name` column's type to `type` and/or the default expression to `default_expr`. If the `IF EXISTS` clause is specified, the query won't return an error if the column doesn't exist

When changing the type, values are converted as if the `toType` functions were applied to them. If only the default expression is changed, the query doesn't do anything complex, and is completed almost instantly

:Example

```
(ALTER TABLE visits MODIFY COLUMN browser Array(String
```

Changing the column type is the only complex action – it changes the contents of files with data. For large tables, this may take a long time

:There are several processing stages

- .Preparing temporary (new) files with modified data
- .Renaming old files
- .Renaming the temporary (new) files to the old names
- .Deleting the old files

.Only the first stage takes time. If there is a failure at this stage, the data is not changed

If there is a failure during one of the successive stages, data can be restored manually. The exception is if the old files were deleted from the file system but the data for the new files did not get written to the disk and was lost



The **ALTER** query for changing columns is replicated. The instructions are saved in ZooKeeper, then each replica applies them. All **ALTER** queries are run in the same order. The query waits for the appropriate actions to be completed on the other replicas. However, a query to change columns in a replicated table can be interrupted, and all actions will be performed asynchronously

## ALTER Query Limitations

The **ALTER** query lets you create and delete separate elements (columns) in nested data structures, but not whole nested data structures. To add a nested data structure, you can add columns with a name like **name.nested\_name** and the type **Array(T)**. A nested data structure is equivalent to multiple array columns with a name that has the same prefix before the dot

There is no support for deleting columns in the primary key or the sampling key (columns that are used in the **ENGINE** expression). Changing the type for columns that are included in the primary key is only possible if this change does not cause the data to be modified (for example, you are allowed to add values to an Enum or to change a type from **DateTime** to **UInt32**)

If the **ALTER** query is not sufficient to make the table changes you need, you can create a new table, copy the data to it using the **INSERT SELECT** query, then switch the tables using the **RENAME** query and delete the old table. You can use the **clickhouse-copier** as an alternative to the **INSERT SELECT** query

The **ALTER** query blocks all reads and writes for the table. In other words, if a long **SELECT** is running at the time of the **ALTER** query, the **ALTER** query will wait for it to complete. At the same time, all new queries to the same table will wait while this **ALTER** is running

For tables that don't store data themselves (such as **Merge** and **Distributed**), **ALTER** just changes the table structure, and does not change the structure of subordinate tables. For example, when running **ALTER** for a **Distributed** table, you will also need to run **ALTER** for the tables on all remote servers

## Manipulations With Key Expressions

The following command is supported

```
MODIFY ORDER BY new_expression
```

It only works for tables in the **MergeTree** family (including replicated tables). The command changes the sorting key of the table to **new\_expression** (an expression or a tuple of expressions). Primary key remains the same

The command is lightweight in a sense that it only changes metadata. To keep the property that data part rows are ordered by the sorting key expression you cannot add expressions containing existing columns to the sorting key (only columns added by the **ADD COLUMN** command in the same **ALTER** query)

## Manipulations With Data Skipping Indices

It only works for tables in the **\*MergeTree** family (including replicated tables). The following operations are available

**ALTER TABLE [db].name ADD INDEX name expression TYPE type GRANULARITY value AFTER name [AFTER name2]** Adds index description to tables metadata •

**ALTER TABLE [db].name DROP INDEX name** - Removes index description from tables metadata and deletes index files from disk •

These commands are lightweight in a sense that they only change metadata or remove files. (Also, they are replicated (syncing indices metadata through ZooKeeper)

## Manipulations With Partitions and Parts

The following operations with **partitions** are available

- **DETACH PARTITION** - Moves a partition to the **detached** directory and forget it
- **DROP PARTITION** - Deletes a partition
- **ATTACH PART|PARTITION** - Adds a part or partition from the **detached** directory to the table
- **REPLACE PARTITION** - Copies the data partition from one table to another
- **CLEAR COLUMN IN PARTITION** - Resets the value of a specified column in a partition
- **FREEZE PARTITION** - Creates a backup of a partition
- **FETCH PARTITION** - Downloads a partition from another server

## DETACH PARTITION

```
ALTER TABLE table_name DETACH PARTITION partition_expr
```

Moves all data for the specified partition to the **detached** directory. The server forgets about the detached data partition as if it does not exist. The server will not know about this data until you make the **ATTACH** query

:Example

```
ALTER TABLE visits DETACH PARTITION 201901
```

.Read about setting the partition expression in a section [How to specify the partition expression](#)

After the query is executed, you can do whatever you want with the data in the **detached** directory — delete it from the file system, or just leave it

This query is replicated - it moves the data to the **detached** directory on all replicas. Note that you can execute this query only on a leader replica. To find out if a replica is a leader, perform the **SELECT** query to the **system.replicas** table. Alternatively, it is easier to make a **DETACH** query on all replicas - all the replicas throw an exception, except the leader replica

## DROP PARTITION

```
ALTER TABLE table_name DROP PARTITION partition_expr
```

Deletes the specified partition from the table. This query tags the partition as inactive and deletes data completely, approximately in 10 minutes

.Read about setting the partition expression in a section [How to specify the partition expression](#)

.The query is replicated - it deletes data on all replicas

## ATTACH PARTITION|PART

```
ALTER TABLE table_name ATTACH PARTITION|PART partition_expr
```

Adds data to the table from the **detached** directory. It is possible to add data for an entire partition or for a separate part. Examples

```
;ALTER TABLE visits ATTACH PARTITION 201901
;ALTER TABLE visits ATTACH PART 201901_2_2_0
```

.Read more about setting the partition expression in a section [How to specify the partition expression](#)

This query is replicated. Each replica checks whether there is data in the **detached** directory. If the data is in this directory, the query checks the integrity, verifies that it matches the data on the server that initiated the query. If everything is correct, the query adds data to the replica. If not, it downloads data from the query requestor replica, or from another replica where the data has already been added

So you can put data to the **detached** directory on one replica, and use the **ALTER ... ATTACH** query to add it to the table on all replicas

## REPLACE PARTITION

```
ALTER TABLE table2 REPLACE PARTITION partition_expr FROM table1
```

.This query copies the data partition from the **table1** to **table2**. Note that data won't be deleted from **table1**

:For the query to run successfully, the following conditions must be met

- .Both tables must have the same structure
- .Both tables must have the same partition key

## CLEAR COLUMN IN PARTITION

```
ALTER TABLE table_name CLEAR COLUMN column_name IN PARTITION partition_expr
```

Resets all values in the specified column in a partition. If the **DEFAULT** clause was determined when creating a table, this query sets the column value to a specified default value

:Example

```
ALTER TABLE visits CLEAR COLUMN hour in PARTITION 201902
```

## FREEZE PARTITION

```
[ALTER TABLE table_name FREEZE [PARTITION partition_expr
```

This query creates a local backup of a specified partition. If the **PARTITION** clause is omitted, the query creates the backup of all partitions at once

Note that for old-styled tables you can specify the prefix of the partition name (for example, '2019') - then the query creates the backup for all the corresponding partitions. Read about setting the partition expression in a section [How to specify the partition expression](#)

Note

.The entire backup process is performed without stopping the server

At the time of execution, for a data snapshot, the query creates hardlinks to a table data. Hardlinks are placed in the directory `/var/lib/clickhouse/shadow/N/...`, where

- `/var/lib/clickhouse/` is the working ClickHouse directory specified in the config/
- `N` is the incremental number of the backup

The same structure of directories is created inside the backup as inside `/var/lib/clickhouse/`. The query performs `'chmod'` for all files, forbidding writing into them

After creating the backup, you can copy the data from `/var/lib/clickhouse/shadow/` to the remote server and then delete it from the local server. Note that the **ALTER t FREEZE PARTITION** query is not replicated. It creates a local backup only on the local server

The query creates backup almost instantly (but first it waits for the current queries to the corresponding table to finish running)

**ALTER TABLE t FREEZE PARTITION** copies only the data, not table metadata. To make a backup of table metadata, copy the file `/var/lib/clickhouse/metadata/database/table.sql`

:To restore data from a backup, do the following

1. (Create the table if it does not exist. To view the query, use the .sql file (replace **ATTACH** in it with **CREATE**)
  2. Copy the data from the `data/database/table/` directory inside the backup to the `/var/lib/clickhouse/data/database/table/detached/` directory
  3. Run **ALTER TABLE t ATTACH PARTITION** queries to add the data to a table
- .Restoring from a backup doesn't require stopping the server

.For more information about backups and restoring data, see the [Data Backup](#) section

## FETCH PARTITION

```
'ALTER TABLE table_name FETCH PARTITION partition_expr FROM 'path-in-zookeeper
```

.Downloads a partition from another server. This query only works for the replicated tables

:The query does the following

Downloads the partition from the specified shard. In 'path-in-zookeeper' you must specify a path to the shard .1  
.in ZooKeeper

Then the query puts the downloaded data to the **detached** directory of the **table\_name** table. Use the **ATTACH** .2  
**.PARTITION|PART** query to add the data to the table

:For example

```
;ALTER TABLE users FETCH PARTITION 201902 FROM '/clickhouse/tables/01-01/visits  
;ALTER TABLE users ATTACH PARTITION 201902
```

:Note that

The **ALTER ... FETCH PARTITION** query isn't replicated. It places the partition to the **detached** directory only on the •  
.local server

The **ALTER TABLE ... ATTACH** query is replicated. It adds the data to all replicas. The data is added to one of the •  
.replicas from the **detached** directory, and to the others - from neighboring replicas

Before downloading, the system checks if the partition exists and the table structure matches. The most  
.appropriate replica is selected automatically from the healthy replicas

Although the query is called **ALTER TABLE**, it does not change the table structure and does not immediately change  
.the data available in the table

## How To Set Partition Expression

:You can specify the partition expression in **ALTER ... PARTITION** queries in different ways

As a value from the **partition** column of the **system.parts** table. For example, **ALTER TABLE visits DETACH PARTITION** •  
**.201901**

As the expression from the table column. Constants and constant expressions are supported. For example, •  
**.('ALTER TABLE visits DETACH PARTITION toYYYYMM(toDate('2019-01-25**

Using the partition ID. Partition ID is a string identifier of the partition (human-readable, if possible) that is •  
used as the names of partitions in the file system and in ZooKeeper. The partition ID must be specified in the  
**.PARTITION ID** clause, in a single quotes. For example, **ALTER TABLE visits DETACH PARTITION ID '201901**

In the **ALTER ATTACH PART** query, to specify the name of a part, use a value from the **name** column of the •  
**.system.parts** table. For example, **ALTER TABLE visits ATTACH PART 201901\_1\_1\_0**

Usage of quotes when specifying the partition depends on the type of partition expression. For example, for the  
**.String** type, you have to specify its name in quotes ('). For the **Date** and **Int\*** types no quotes are needed

For old-style tables, you can specify the partition either as a number **201901** or a string **'201901'**. The syntax for the  
. (new-style tables is stricter with types (similar to the parser for the VALUES input format

All the rules above are also true for the **OPTIMIZE** query. If you need to specify the only partition when optimizing a  
:non-partitioned table, set the expression **PARTITION tuple()**. For example

```
;OPTIMIZE TABLE table_not_partitioned PARTITION tuple() FINAL
```

The examples of **ALTER ... PARTITION** queries are demonstrated in the tests **00502\_custom\_partitioning\_local** and  
**.00502\_custom\_partitioning\_replicated\_zookeeper**

## Synchronicity of ALTER Queries

For non-replicable tables, all **ALTER** queries are performed synchronously. For replicatable tables, the query just  
adds instructions for the appropriate actions to **ZooKeeper**, and the actions themselves are performed as soon as  
.possible. However, the query can wait for these actions to be completed on all the replicas

.For **ALTER ... ATTACH|DETACH|DROP** queries, you can use the **replication\_alter\_partitions\_sync** setting to set up waiting  
.Possible values: **0** – do not wait; **1** – only wait for own execution (default); **2** – wait for all

## Mutations

Mutations are an ALTER query variant that allows changing or deleting rows in a table. In contrast to standard **UPDATE** and **DELETE** queries that are intended for point data changes, mutations are intended for heavy operations that change a lot of rows in a table

The functionality is in beta stage and is available starting with the 1.1.54388 version. Currently **\*MergeTree** table engines are supported (both replicated and unreplicated)

Existing tables are ready for mutations as-is (no conversion necessary), but after the first mutation is applied to a table, its metadata format becomes incompatible with previous server versions and falling back to a previous version becomes impossible

:Currently available commands

```
ALTER TABLE [db.]table DELETE WHERE filter_expr
```

The **filter\_expr** must be of type UInt8. The query deletes rows in the table for which this expression takes a non-zero value

```
ALTER TABLE [db.]table UPDATE column1 = expr1 [, ...] WHERE filter_expr
```

The command is available starting with the 18.12.14 version. The **filter\_expr** must be of type UInt8. This query updates values of specified columns to the values of corresponding expressions in rows for which the **filter\_expr** takes a non-zero value. Values are casted to the column type using the **CAST** operator. Updating columns that are used in the calculation of the primary or the partition key is not supported

.One query can contain several commands separated by commas

For **\*MergeTree** tables mutations execute by rewriting whole data parts. There is no atomicity - parts are substituted for mutated parts as soon as they are ready and a **SELECT** query that started executing during a mutation will see data from parts that have already been mutated along with data from parts that have not been mutated yet

Mutations are totally ordered by their creation order and are applied to each part in that order. Mutations are also partially ordered with INSERTs - data that was inserted into the table before the mutation was submitted will be mutated and data that was inserted after that will not be mutated. Note that mutations do not block INSERTs in any way

A mutation query returns immediately after the mutation entry is added (in case of replicated tables to ZooKeeper, for nonreplicated tables - to the filesystem). The mutation itself executes asynchronously using the system profile settings. To track the progress of mutations you can use the **system.mutations** table. A mutation that was successfully submitted will continue to execute even if ClickHouse servers are restarted. There is no way to roll back the mutation once it is submitted, but if the mutation is stuck for some reason it can be cancelled with the **KILL MUTATION** query

Entries for finished mutations are not deleted right away (the number of preserved entries is determined by the **finished\_mutations\_to\_keep** storage engine parameter). Older mutation entries are deleted

## Miscellaneous Queries

### ATTACH

This query is exactly the same as **CREATE**, but

.Instead of the word **CREATE** it uses the word **ATTACH**

The query does not create data on the disk, but assumes that data is already in the appropriate places, and just adds information about the table to the server

.After executing an ATTACH query, the server will know about the existence of the table

If the table was previously detached (**DETACH**), meaning that its structure is known, you can use shorthand without defining the structure

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

This query is used when starting the server. The server stores table metadata as files with **ATTACH** queries, which .(it simply runs at launch (with the exception of system tables, which are explicitly created on the server

## CHECK TABLE

.Checks if the data in the table is corrupted

```
CHECK TABLE [db.]name
```

The **CHECK TABLE** query compares actual file sizes with the expected values which are stored on the server. If the file sizes do not match the stored values, it means the data is corrupted. This can be caused, for example, by a .system crash during query execution

The query response contains the **result** column with a single row. The row has a value of **:Boolean** type

- .The data in the table is corrupted - 0
- .The data maintains integrity - 1

:The **CHECK TABLE** query is only supported for the following table engines

- Log
- TinyLog
- StripeLog

These engines do not provide automatic data recovery on failure. Use the **CHECK TABLE** query to track data loss in a .timely manner

.To avoid data loss use the **MergeTree** family tables

### If the data is corrupted

:If the table is corrupted, you can copy the non-corrupted data to another table. To do this

- Create a new table with the same structure as damaged table. To do this execute the query **CREATE TABLE** .<new\_table\_name> AS <damaged\_table\_name> .1
- Set the **max\_threads** value to 1 to process the next query in a single thread. To do this run the query **SET** .max\_threads = 1 .2
- Execute the query **INSERT INTO <new\_table\_name> SELECT \* FROM <damaged\_table\_name>**. This request copies the .3 non-corrupted data from the damaged table to another table. Only the data before the corrupted part will be .copied
- .Restart the **clickhouse-client** to reset the **max\_threads** value .4

## DESCRIBE TABLE

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

:Returns the following **String** type columns

- .name — Column name
- .type — Column type
- default\_type — Clause that is used in **default expression** (**DEFAULT**, **MATERIALIZED** or **ALIAS**). Column contains an .empty string, if the default expression isn't specified
- .default\_expression — Value specified in the **DEFAULT** clause
- .comment\_expression — Comment text

Nested data structures are output in "expanded" format. Each column is shown separately, with the name after a .dot

## DETACH

.Deletes information about the 'name' table from the server. The server stops knowing about the table's existence

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

This does not delete the table's data or metadata. On the next server launch, the server will read the metadata and find out about the table again

Similarly, a "detached" table can be re-attached using the **ATTACH** query (with the exception of system tables, which do not have metadata stored for them)

.There is no **DETACH DATABASE** query

## DROP

.This query has two types: **DROP DATABASE** and **DROP TABLE**

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

.Deletes all tables inside the 'db' database, then deletes the 'db' database itself

.If **IF EXISTS** is specified, it doesn't return an error if the database doesn't exist

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

.Deletes the table

.If **IF EXISTS** is specified, it doesn't return an error if the table doesn't exist or the database doesn't exist

## EXISTS

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

Returns a single **UInt8**-type column, which contains the single value **0** if the table or database doesn't exist, or **1** if the table exists in the specified database

## KILL QUERY

```
[KILL QUERY [ON CLUSTER cluster  
<WHERE <where expression to SELECT FROM system.processes query  
[SYNC|ASYNC|TEST]  
[FORMAT format]
```

.Attempts to forcibly terminate the currently running queries

The queries to terminate are selected from the system.processes table using the criteria defined in the **WHERE** clause of the **KILL** query

:Examples

```
:Forcibly terminates all queries with the specified query_id --  
'KILL QUERY WHERE query_id='2-857d-4a57-9ee0-327da5d60a90  
  
:'Synchronously terminates all queries run by 'username --  
KILL QUERY WHERE user='username' SYNC
```

.Read-only users can only stop their own queries

By default, the asynchronous version of queries is used (**ASYNC**), which doesn't wait for confirmation that queries have stopped

The synchronous version (**SYNC**) waits for all queries to stop and displays information about each process as it stops

:The response contains the **kill\_status** column, which can take the following values

- .finished' - The query was terminated successfully' .1
- .waiting' - Waiting for the query to end after sending it a signal to terminate' .2
- .The other values explain why the query can't be stopped .3

.A test query (**TEST**) only checks the user's rights and displays a list of queries to stop

## KILL MUTATION

```
[KILL MUTATION [ON CLUSTER cluster  
<WHERE <where expression to SELECT FROM system.mutations query  
[TEST]  
[FORMAT format]
```

Tries to cancel and remove **mutations** that are currently executing. Mutations to cancel are selected from the **.system.mutations** table using the filter specified by the **WHERE** clause of the **KILL** query

.A test query (**TEST**) only checks the user's rights and displays a list of queries to stop

:Examples

```
:Cancel and remove all mutations of the single table --  
'KILL MUTATION WHERE database = 'default' AND table = 'table'  
  
:Cancel the specific mutation --  
'KILL MUTATION WHERE database = 'default' AND table = 'table' AND mutation_id = 'mutation_3.txt'
```

The query is useful when a mutation is stuck and cannot finish (e.g. if some function in the mutation query throws  
(an exception when applied to the data contained in the table

.Changes already made by the mutation are not rolled back

## OPTIMIZE

```
[OPTIMIZE TABLE [db.]name [ON CLUSTER cluster] [PARTITION partition] [FINAL
```

.Asks the table engine to do something for optimization

.Supported only by **\*MergeTree** engines, in which this query initializes a non-scheduled merge of data parts

.If you specify a **PARTITION**, only the specified partition will be optimized

.If you specify **FINAL**, optimization will be performed even when all the data is already in one part

Warning

.OPTIMIZE can't fix the "Too many parts" error

## RENAME

.Renames one or more tables

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

All tables are renamed under global locking. Renaming tables is a light operation. If you indicated another database after TO, the table will be moved to this database. However, the directories with databases must reside  
(in the same file system (otherwise, an error is returned

## SET

```
SET param = value
```

Allows you to set **param** to **value**. You can also make all the settings from the specified settings profile in a single

."query. To do this, specify 'profile' as the setting name. For more information, see the section "Settings

.The setting is made for the session, or for the server (globally) if **GLOBAL** is specified

When making a global setting, the setting is not applied to sessions already running, including the current session.

.It will only be used for new sessions

.When the server is restarted, global settings made using **SET** are lost

.To make settings that persist after a server restart, you can only use the server's config file

## SHOW CREATE TABLE



```
[SHOW CREATE [TEMPORARY] TABLE [db.]table [INTO OUTFILE filename] [FORMAT format]
```

Returns a single **String**-type 'statement' column, which contains a single value – the **CREATE** query used for creating the specified table

## SHOW DATABASES

```
[SHOW DATABASES [INTO OUTFILE filename] [FORMAT format]
```

.Prints a list of all databases

.**[This query is identical to `SELECT name FROM system.databases [INTO OUTFILE filename] [FORMAT format`**

."See also the section "Formats

## SHOW PROCESSLIST

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

.Outputs a list of queries currently being processed, other than **SHOW PROCESSLIST** queries

:Prints a table containing the columns

**user** – The user who made the query. Keep in mind that for distributed processing, queries are sent to remote servers under the 'default' user. SHOW PROCESSLIST shows the username for a specific query, not for a query that this query initiated

**address** – The name of the host that the query was sent from. For distributed processing, on remote servers, this is the name of the query requestor host. To track where a distributed query was originally made from, look at .SHOW PROCESSLIST on the query requestor server

.**elapsed** – The execution time, in seconds. Queries are output in order of decreasing execution time

**rows\_read, bytes\_read** – How many rows and bytes of uncompressed data were read when processing the query. For distributed processing, data is totaled from all the remote servers. This is the data used for restrictions and quotas

.**memory\_usage** – Current RAM usage in bytes. See the setting 'max\_memory\_usage

.**query** – The query itself. In INSERT queries, the data for insertion is not output

**query\_id** – The query identifier. Non-empty only if it was explicitly defined by the user. For distributed processing, the query ID is not passed to remote servers

This query is nearly identical to: **SELECT \* FROM system.processes**. The difference is that the **SHOW PROCESSLIST** query does not show itself in a list, when the **SELECT .. FROM system.processes** query does

:(Tip (execute in the console

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

## SHOW TABLES

```
[SHOW [TEMPORARY] TABLES [FROM db] [LIKE 'pattern'] [INTO OUTFILE filename] [FORMAT format]
```

Displays a list of tables

.Tables from the current database, or from the 'db' database if "FROM db" is specified

.All tables, or tables whose name matches the pattern, if "LIKE 'pattern'" is specified

This query is identical to: **SELECT name FROM system.tables WHERE database = 'db' [AND name LIKE 'pattern'] [INTO OUTFILE .filename] [FORMAT format**

."See also the section "LIKE operator

# TRUNCATE

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

Removes all data from a table. When the clause **IF EXISTS** is omitted, the query returns an error if the table does not exist

The **TRUNCATE** query is not supported for **View**, **File**, **URL** and **Null** table engines

## USE

```
USE db
```

Lets you set the current database for the session

The current database is used for searching for tables if the database is not explicitly defined in the query with a dot before the table name

This query can't be made when using the HTTP protocol, since there is no concept of a session

## Functions

There are at least\* two types of functions - regular functions (they are just called "functions") and aggregate functions. These are completely different concepts. Regular functions work as if they are applied to each row separately (for each row, the result of the function doesn't depend on the other rows). Aggregate functions (accumulate a set of values from various rows (i.e. they depend on the entire set of rows

"In this section we discuss regular functions. For aggregate functions, see the section "Aggregate functions

There is a third type of function that the 'arrayJoin' function belongs to; table functions can also be mentioned - \*separately

## Strong typing

In contrast to standard SQL, ClickHouse has strong typing. In other words, it doesn't make implicit conversions between types. Each function works for a specific set of types. This means that sometimes you need to use type conversion functions

## Common subexpression elimination

All expressions in a query that have the same AST (the same record or same result of syntactic parsing) are considered to have identical values. Such expressions are concatenated and executed once. Identical subqueries are also eliminated this way

## Types of results

All functions return a single return as the result (not several values, and not zero values). The type of result is usually defined only by the types of arguments, not by the values. Exceptions are the tupleElement function (the a.N operator), and the toFixedString function

## Constants

For simplicity, certain functions can only work with constants for some arguments. For example, the right argument of the LIKE operator must be a constant

Almost all functions return a constant for constant arguments. The exception is functions that generate random numbers

The 'now' function returns different values for queries that were run at different times, but the result is considered a constant, since constancy is only important within a single query

A constant expression is also considered a constant (for example, the right half of the LIKE operator can be (constructed from multiple constants

Functions can be implemented in different ways for constant and non-constant arguments (different code is executed). But the results for a constant and for a true column containing only the same value should match each other

# NULL processing

Functions have the following behaviors

- If at least one of the arguments of the function is **NULL**, the function result is also **NULL**
- Special behavior that is specified individually in the description of each function. In the ClickHouse source code, these functions have **UseDefaultImplementationForNulls=false**

## Constancy

Functions can't change the values of their arguments – any changes are returned as the result. Thus, the result of calculating separate functions does not depend on the order in which the functions are written in the query

## Error handling

Some functions might throw an exception if the data is invalid. In this case, the query is canceled and an error text is returned to the client. For distributed processing, when an exception occurs on one of the servers, the other servers also attempt to abort the query

## Evaluation of argument expressions

In almost all programming languages, one of the arguments might not be evaluated for certain operators. This is usually the operators **&&**, **||**, and

But in ClickHouse, arguments of functions (operators) are always evaluated. This is because entire parts of columns are evaluated at once, instead of calculating each row separately

## Performing functions for distributed query processing

For distributed query processing, as many stages of query processing as possible are performed on remote servers, and the rest of the stages (merging intermediate results and everything after that) are performed on the requestor server

This means that functions can be performed on different servers

(For example, in the query **SELECT f(sum(g(x))) FROM distributed\_table GROUP BY h(y)**

- if a **distributed\_table** has at least two shards, the functions 'g' and 'h' are performed on remote servers, and the function 'f' is performed on the requestor server

- if a **distributed\_table** has only one shard, all the 'f', 'g', and 'h' functions are performed on this shard's server

The result of a function usually doesn't depend on which server it is performed on. However, sometimes this is important

For example, functions that work with dictionaries use the dictionary that exists on the server they are running on. Another example is the **hostName** function, which returns the name of the server it is running on in order to make **GROUP BY** by servers in a **SELECT** query

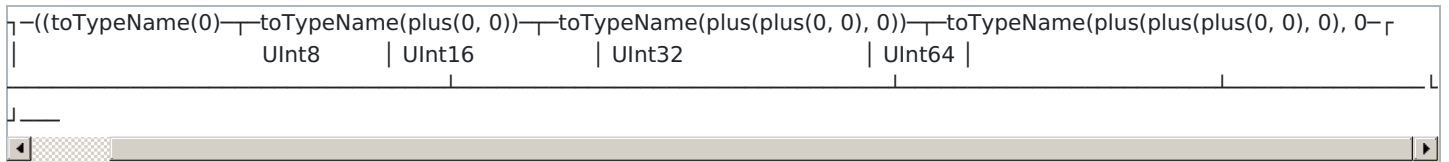
If a function in a query is performed on the requestor server, but you need to perform it on remote servers, you can wrap it in an 'any' aggregate function or add it to a key in **GROUP BY**

## Arithmetic functions

For all arithmetic functions, the result type is calculated as the smallest number type that the result fits in, if there is such a type. The minimum is taken simultaneously based on the number of bits, whether it is signed, and whether it floats. If there are not enough bits, the highest bit type is taken

Example

```
(SELECT toTypeName(0), toTypeName(0 + 0), toTypeName(0 + 0 + 0), toTypeName(0 + 0 + 0 + 0
```



Arithmetic functions work for any pair of types from UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64, Float32, or Float64

.++Overflow is produced the same way as in C

## plus(a, b), a + b operator

.Calculates the sum of the numbers

You can also add integer numbers with a date or date and time. In the case of a date, adding an integer means adding the corresponding number of days. For a date with time, it means adding the corresponding number of seconds

## minus(a, b), a - b operator

.Calculates the difference. The result is always signed

.You can also calculate integer numbers from a date or date with time. The idea is the same – see above for 'plus

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

.Calculates the product of the numbers

## divide(a, b), a / b operator

.Calculates the quotient of the numbers. The result type is always a floating-point type

.It is not integer division. For integer division, use the 'intDiv' function

.When dividing by zero you get 'inf', '-inf', or 'nan

## (intDiv(a, b

.Calculates the quotient of the numbers. Divides into integers, rounding down (by the absolute value

.An exception is thrown when dividing by zero or when dividing a minimal negative number by minus one

## (intDivOrZero(a, b

Differs from 'intDiv' in that it returns zero when dividing by zero or when dividing a minimal negative number by minus one

## modulo(a, b), a % b operator

.Calculates the remainder after division

.If arguments are floating-point numbers, they are pre-converted to integers by dropping the decimal portion

.The remainder is taken in the same sense as in C++. Truncated division is used for negative numbers

.An exception is thrown when dividing by zero or when dividing a minimal negative number by minus one

## negate(a), -a operator

.Calculates a number with the reverse sign. The result is always signed

## (abs(a

Calculates the absolute value of the number (a). That is, if a < 0, it returns -a. For unsigned types it doesn't do anything. For signed integer types, it returns an unsigned number

## (gcd(a, b

.Returns the greatest common divisor of the numbers

.An exception is thrown when dividing by zero or when dividing a minimal negative number by minus one

## lcm(a, b

.Returns the least common multiple of the numbers

.An exception is thrown when dividing by zero or when dividing a minimal negative number by minus one

## Comparison functions

.(Comparison functions always return 0 or 1 (UInt8

:The following types can be compared

numbers

strings and fixed strings

dates

dates with times

- 
- 
- 
- 

.within each group, but not between different groups

For example, you can't compare a date with a string. You have to use a function to convert the string to a date, or .vice versa

Strings are compared by bytes. A shorter string is smaller than all strings that start with it and that contain at least one more character

Note. Up until version 1.1.54134, signed and unsigned numbers were compared the same way as in C++. In other words, you could get an incorrect result in cases like `SELECT 9223372036854775807 > -1`. This behavior changed in version 1.1.54134 and is now mathematically correct

equals, `a = b` and `a == b` operator

notEquals, `a != b` and `a <> b`

less, `< operator`

greater, `> operator`

lessOrEquals, `<= operator`

greaterOrEquals, `>= operator`

## Logical functions

.Logical functions accept any numeric types, but return a UInt8 number equal to 0 or 1

."Zero as an argument is considered "false," while any non-zero value is considered "true"

and, AND operator

or, OR operator

not, NOT operator

xor

## Type conversion functions

toUInt8, toUInt16, toUInt32, toUInt64

toInt8, toInt16, toInt32, toInt64

toFloat32, toFloat64

toDate, toDateTime

toUInt8OrZero, toUInt16OrZero, toUInt32OrZero, toUInt64OrZero,  
toInt8OrZero, toInt16OrZero, toInt32OrZero, toInt64OrZero,  
toFloat32OrZero, toFloat64OrZero, toDateOrZero,  
toDateTimeOrZero

toUInt8OrNull, toUInt16OrNull, toUInt32OrNull, toUInt64OrNull,  
toInt8OrNull, toInt16OrNull, toInt32OrNull, toInt64OrNull,  
toFloat32OrNull, toFloat64OrNull, toDateOrNull,  
toDateTimeOrNull

toString

.Functions for converting between numbers, strings (but not fixed strings), dates, and dates with times  
.All these functions accept one argument

When converting to or from a string, the value is formatted or parsed using the same rules as for the  
TabSeparated format (and almost all other text formats). If the string can't be parsed, an exception is thrown and  
.the request is canceled

When converting dates to numbers or vice versa, the date corresponds to the number of days since the beginning  
.of the Unix epoch

When converting dates with times to numbers or vice versa, the date with time corresponds to the number of  
.seconds since the beginning of the Unix epoch

:The date and date-with-time formats for the toDate/toDateTime functions are defined as follows

```
YYYY-MM-DD  
YYYY-MM-DD hh:mm:ss
```

As an exception, if converting from UInt32, Int32, UInt64, or Int64 numeric types to Date, and if the number is  
greater than or equal to 65536, the number is interpreted as a Unix timestamp (and not as the number of days)  
and is rounded to the date. This allows support for the common occurrence of writing 'toDate(unix\_timestamp)',  
which otherwise would be an error and would require writing the more cumbersome  
.('toDate(toDateTime(unix\_timestamp

Conversion between a date and date with time is performed the natural way: by adding a null time or dropping the  
.time

.++Conversion between numeric types uses the same rules as assignments between different numeric types in C

Additionally, the toString function of the DateTime argument can take a second String argument containing the  
name of the time zone. Example: [Asia/Yekaterinburg](#) In this case, the time is formatted according to the specified  
.time zone

```
SELECT  
,now() AS now_local  
toString(now(), 'Asia/Yekaterinburg') AS now_yekat
```

```
┌──────────now_local──┴──────────now_yekat──────────┐  
| 02:11:21 2016-06-15 | 00:11:21 2016-06-15 |  
└──────────┴──────────┘
```

.Also see the [toUnixTimestamp](#) function

toDecimal32(value, S), toDecimal64(value, S),  
(toDecimal128(value, S

Converts **value** to **Decimal** of precision **S**. The **value** can be a number or a string. The **S** (scale) parameter specifies the number of decimal places

## (toFixedString(s, N

.Converts a String type argument to a FixedString(N) type (a string with fixed length N). N must be a constant  
If the string has fewer bytes than N, it is passed with null bytes to the right. If the string has more bytes than N, an exception is thrown

## (toStringCutToZero(s

Accepts a String or FixedString argument. Returns the String with the content truncated at the first zero byte  
.found

:Example

```
SELECT toFixedString('foo', 8) AS s, toStringCutToZero(s) AS s_cut
```

```
┌-s-┐┌-s_cut-┐  
│foo\0\0\0\0\0│foo│  
└-┘└-┘
```

```
SELECT toFixedString('foo\0bar', 8) AS s, toStringCutToZero(s) AS s_cut
```

```
┌-s-┐┌-s_cut-┐  
│foo\0bar\0│foo│  
└-┘└-┘
```

reinterpretAsUInt8, reinterpretAsUInt16, reinterpretAsUInt32,  
reinterpretAsUInt64

reinterpretAsInt8, reinterpretAsInt16, reinterpretAsInt32,  
reinterpretAsInt64

reinterpretAsFloat32, reinterpretAsFloat64

reinterpretAsDate, reinterpretAsDateTime

These functions accept a string and interpret the bytes placed at the beginning of the string as a number in host order (little endian). If the string isn't long enough, the functions work as if the string is padded with the necessary number of null bytes. If the string is longer than needed, the extra bytes are ignored. A date is interpreted as the number of days since the beginning of the Unix Epoch, and a date with time is interpreted as the number of seconds since the beginning of the Unix Epoch

## reinterpretAsString

This function accepts a number or date or date with time, and returns a string containing bytes representing the corresponding value in host order (little endian). Null bytes are dropped from the end. For example, a UInt32 type value of 255 is a string that is one byte long

## reinterpretAsFixedString

This function accepts a number or date or date with time, and returns a FixedString containing bytes representing the corresponding value in host order (little endian). Null bytes are dropped from the end. For example, a UInt32 type value of 255 is a FixedString that is one byte long

## (CAST(x, t

.Converts 'x' to the 't' data type. The syntax CAST(x AS t) is also supported

:Example

```
SELECT
,AS timestamp '23:00:00 2016-06-15'
,CAST(timestamp AS DateTime) AS datetime
,CAST(timestamp AS Date) AS date
,CAST(timestamp, 'String') AS string
CAST(timestamp, 'FixedString(22)') AS fixed_string
```

timestamp	datetime	date	string	fixed_string
0\0\0\23:00:00 2016-06-15	23:00:00 2016-06-15	2016-06-15	23:00:00 2016-06-15	23:00:00 2016-06-15

.(Conversion to FixedString(N) only works for arguments of type String or FixedString(N)

:Type conversion to **Nullable** and back is supported. Example

```
SELECT toTypeName(x) FROM t_null

--(toTypeName(x--r
|      Int8 |
|      Int8 |
|_____|L

SELECT toTypeName(CAST(x, 'Nullable(UInt16)')) FROM t_null

--(('toTypeName(CAST(x, 'Nullable(UInt16--r
|      (Nullable(UInt16 |
|      (Nullable(UInt16 |
|_____|L
```

## toIntervalYear, toIntervalQuarter, toIntervalMonth, toIntervalWeek, toIntervalDay, toIntervalHour, toIntervalMinute, toIntervalSecond

.(Converts a Number type argument to a Interval type (duration

The interval type is actually very useful, you can use this type of data to perform arithmetic operations directly with Date or DateTime. At the same time, ClickHouse provides a more convenient syntax for declaring Interval :type data. For example

```
WITH
,date('2019-01-01') AS date
,INTERVAL 1 WEEK AS interval_week
toIntervalWeek(1) AS interval_to_week
SELECT
,date + interval_week
date + interval_to_week
```

+(plus(date, interval_week)--plus(date, interval_to_week--r
2019-01-08   2019-01-08

## parseDateTimeBestEffort

.Parse a number type argument to a Date or DateTime type

.different from toDate and toDateTime, parseDateTimeBestEffort can progress more complex date format

For more information, see the link: [Complex Date Format](#)

## parseDateTimeBestEffortOrNull

Same as for [parseDateTimeBestEffort](#) except that it returns null when it encounters a date format that cannot be .processed

## parseDateTimeBestEffortOrZero



Same as for `parseDateTimeBestEffort` except that it returns zero date or zero date time when it encounters a date .format that cannot be processed

## Functions for working with dates and times

Support for time zones

All functions for working with the date and time that have a logical use for the time zone can accept a second optional time zone argument. Example: Asia/Yekaterinburg. In this case, they use the specified time zone instead .of the local (default) one

```
SELECT
,toDateTime('2016-06-15 23:00:00') AS time
,toDate(time) AS date_local
,toDate(time, 'Asia/Yekaterinburg') AS date_yekat
toString(time, 'US/Samoa') AS time_samoa
```

time	date_local	date_yekat	time_samoa
09:00:00	2016-06-15	2016-06-16	2016-06-15   23:00:00

.Only time zones that differ from UTC by a whole number of hours are supported

### toTimeZone

.Convert time or date and time to the specified time zone

### toYear

.(Converts a date or date with time to a UInt16 number containing the year number (AD

### toQuarter

.Converts a date or date with time to a UInt8 number containing the quarter number

### toMonth

.(Converts a date or date with time to a UInt8 number containing the month number (1-12

### toDayOfYear

.(Converts a date or date with time to a UInt16 number containing the number of the day of the year (1-366

### toDayOfMonth

.(Converts a date or date with time to a UInt8 number containing the number of the day of the month (1-31

### toDayOfWeek

Converts a date or date with time to a UInt8 number containing the number of the day of the week (Monday is 1, .(and Sunday is 7

### toHour

.(Converts a date with time to a UInt8 number containing the number of the hour in 24-hour time (0-23  
This function assumes that if clocks are moved ahead, it is by one hour and occurs at 2 a.m., and if clocks are moved back, it is by one hour and occurs at 3 a.m. (which is not always true – even in Moscow the clocks were .(twice changed at a different time

### toMinute

.(Converts a date with time to a UInt8 number containing the number of the minute of the hour (0-59

### toSecond

.(Converts a date with time to a UInt8 number containing the number of the second in the minute (0-59)  
.Leap seconds are not accounted for

## toUnixTimestamp

.Converts a date with time to a unix timestamp

## toStartOfYear

.Rounds down a date or date with time to the first day of the year  
.Returns the date

## toStartOfISOYear

.Rounds down a date or date with time to the first day of ISO year  
.Returns the date

## toStartOfQuarter

.Rounds down a date or date with time to the first day of the quarter  
.The first day of the quarter is either 1 January, 1 April, 1 July, or 1 October  
.Returns the date

## toStartOfMonth

.Rounds down a date or date with time to the first day of the month  
.Returns the date

### Attention

The behavior of parsing incorrect dates is implementation specific. ClickHouse may return zero date, throw an exception or do "natural" overflow

## toMonday

.Rounds down a date or date with time to the nearest Monday  
.Returns the date

## toStartOfDay

.Rounds down a date with time to the start of the day

## toStartOfHour

.Rounds down a date with time to the start of the hour

## toStartOfMinute

.Rounds down a date with time to the start of the minute

## toStartOfFiveMinute

.Rounds down a date with time to the start of the five-minute interval

## toStartOfTenMinutes

.Rounds down a date with time to the start of the ten-minute interval

## toStartOfFifteenMinutes

.Rounds down the date with time to the start of the fifteen-minute interval

([toStartOfInterval(time\_or\_data, INTERVAL x unit [, time\_zone

,This is a generalization of other functions named `toStartOf*`. For example  
,`(toStartOfInterval(t, INTERVAL 1 year)` returns the same as `toStartOfYear(t`  
,`(toStartOfInterval(t, INTERVAL 1 month)` returns the same as `toStartOfMonth(t`  
,`(toStartOfInterval(t, INTERVAL 1 day)` returns the same as `toStartOfDay(t`  
,`toStartOfInterval(t, INTERVAL 15 minute)` returns the same as `toStartOfFifteenMinutes(t)` etc

## toTime

.Converts a date with time to a certain fixed date, while preserving the time

## toRelativeYearNum

.Converts a date with time or date to the number of the year, starting from a certain fixed point in the past

## toRelativeQuarterNum

.Converts a date with time or date to the number of the quarter, starting from a certain fixed point in the past

## toRelativeMonthNum

.Converts a date with time or date to the number of the month, starting from a certain fixed point in the past

## toRelativeWeekNum

.Converts a date with time or date to the number of the week, starting from a certain fixed point in the past

## toRelativeDayNum

.Converts a date with time or date to the number of the day, starting from a certain fixed point in the past

## toRelativeHourNum

.Converts a date with time or date to the number of the hour, starting from a certain fixed point in the past

## toRelativeMinuteNum

.Converts a date with time or date to the number of the minute, starting from a certain fixed point in the past

## toRelativeSecondNum

.Converts a date with time or date to the number of the second, starting from a certain fixed point in the past

## toISOYear

.Converts a date or date with time to a UInt16 number containing the ISO Year number

## toISOWeek

.Converts a date or date with time to a UInt8 number containing the ISO Week number

## now

.Accepts zero arguments and returns the current time at one of the moments of request execution  
.This function returns a constant, even if the request took a long time to complete

## today

.Accepts zero arguments and returns the current date at one of the moments of request execution  
'`((()`The same as `'toDate(now`

## yesterday

.Accepts zero arguments and returns yesterday's date at one of the moments of request execution  
'The same as `'today() - 1`

## timeSlot

.Rounds the time to the half hour

This function is specific to Yandex.Metrica, since half an hour is the minimum amount of time for breaking a session into two sessions if a tracking tag shows a single user's consecutive pageviews that differ in time by strictly more than this amount. This means that tuples (the tag ID, user ID, and time slot) can be used to search .for pageviews that are included in the corresponding session

## toYYYYMM

.(Converts a date or date with time to a UInt32 number containing the year and month number ( $YYYY * 100 + MM$

## toYYYYMMDD

Converts a date or date with time to a UInt32 number containing the year and month number ( $YYYY * 10000 + MM * 100 + DD$

## toYYYYMMDDhhmmss

Converts a date or date with time to a UInt64 number containing the year and month number ( $YYYY * (10000000000 + MM * 100000000 + DD * 100000 + hh * 10000 + mm * 100 + ss$

## addYears, addMonths, addWeeks, addDays, addHours, addMinutes, addSeconds, addQuarters

:Function adds a Date/DateTime interval to a Date/DateTime and then return the Date/DateTime. For example

```
WITH
,toDate('2018-01-01') AS date
toDateTime('2018-01-01 00:00:00') AS date_time
SELECT
,addYears(date, 1) AS add_years_with_date
addYears(date_time, 1) AS add_years_with_date_time
```

```
|--add_years_with_date--|add_years_with_date_time--|
| 00:00:00 2019-01-01 | 2019-01-01 |
|-----|-----|
```

## subtractYears, subtractMonths, subtractWeeks, subtractDays, subtractHours, subtractMinutes, subtractSeconds, subtractQuarters

:Function subtract a Date/DateTime interval to a Date/DateTime and then return the Date/DateTime. For example

```
WITH
,toDate('2019-01-01') AS date
toDateTime('2019-01-01 00:00:00') AS date_time
SELECT
,subtractYears(date, 1) AS subtract_years_with_date
subtractYears(date_time, 1) AS subtract_years_with_date_time
```

```
|--subtract_years_with_date--|subtract_years_with_date_time--|
| 00:00:00 2018-01-01 | 2018-01-01 |
|-----|-----|
```

## [dateDiff('unit', t1, t2, [timezone

Return the difference between two times expressed in 'unit' e.g. 'hours'. 't1' and 't2' can be Date or DateTime, If 'timezone' is specified, it applied to both arguments. If not, timezones from datatypes 't1' and 't2' are used. If that .timezones are not the same, the result is unspecified

:Supported unit values

**unit**  
second  
minute  
hour  
day  
week  
month  
quarter  
year

## ([timeSlots(StartTime, Duration[, Size

For a time interval starting at 'StartTime' and continuing for 'Duration' seconds, it returns an array of moments in time, consisting of points from this interval rounded down to the 'Size' in seconds. 'Size' is an optional parameter: .a constant UInt32, set to 1800 by default  
For example, `timeSlots(toDateTime('2012-01-01 12:20:00'), 600) = [toDateTime('2012-01-01 12:00:00'), toDateTime('2012-01-01 12:30:00')`  
.This is necessary for searching for pageviews in the corresponding session

## ([formatDateTime(Time, Format[, Timezone

Function formats a Time according given Format string. N.B.: Format is a constant expression, e.g. you can not .have multiple formats for single result column  
:Supported modifiers for Format  
(Example" column shows formatting result for time `2018-01-02 22:33:44"`)

Example	Description	Modifier
20	(year divided by 100 and truncated to integer (00-99	C%
02	(day of the month, zero-padded (01-31	d%
01/02/2018	Short MM/DD/YY date, equivalent to %m/%d/%y	D%
2	(day of the month, space-padded ( 1-31	e%
2018-01-02	short YYYY-MM-DD date, equivalent to %Y-%m-%d	F%
22	(hour in 24h format (00-23	H%
10	(hour in 12h format (01-12	I%
002	(day of the year (001-366	j%
01	(month as a decimal number (01-12	m%
33	(minute (00-59	M%
	('new-line character ('\n	n%
PM	AM or PM designation	p%
22:33	hour HH:MM time, equivalent to %H:%M-24	R%
44	(second (00-59	S%
	('horizontal-tab character ('\t	t%
22:33:44	ISO 8601 time format (HH:MM:SS), equivalent to %H:%M:%S	T%
2	(ISO 8601 weekday as number with Monday as 1 (1-7	u%
01	(ISO 8601 week number (01-53	V%
2	(weekday as a decimal number with Sunday as 0 (0-6	w%
18	(Year, last two digits (00-99	y%
2018	Year	Y%
%	a % sign	%%

## Functions for working with strings

### empty

- .Returns 1 for an empty string or 0 for a non-empty string
- .The result type is UInt8
- .A string is considered non-empty if it contains at least one byte, even if this is a space or a null byte
- .The function also works for arrays

## notEmpty

- .Returns 0 for an empty string or 1 for a non-empty string
- .The result type is UInt8
- .The function also works for arrays

## length

- .(Returns the length of a string in bytes (not in characters, and not in code points)
- .The result type is UInt64
- .The function also works for arrays

## lengthUTF8

Returns the length of a string in Unicode code points (not in characters), assuming that the string contains a set of bytes that make up UTF-8 encoded text. If this assumption is not met, it returns some result (it doesn't throw an exception)

- .The result type is UInt64

## char\_length, CHAR\_LENGTH

Returns the length of a string in Unicode code points (not in characters), assuming that the string contains a set of bytes that make up UTF-8 encoded text. If this assumption is not met, it returns some result (it doesn't throw an exception)

- .The result type is UInt64

## character\_length, CHARACTER\_LENGTH

Returns the length of a string in Unicode code points (not in characters), assuming that the string contains a set of bytes that make up UTF-8 encoded text. If this assumption is not met, it returns some result (it doesn't throw an exception)

- .The result type is UInt64

## lower, lcase

- .Converts ASCII Latin symbols in a string to lowercase

## upper, ucase

- .Converts ASCII Latin symbols in a string to uppercase

## lowerUTF8

- .Converts a string to lowercase, assuming the string contains a set of bytes that make up a UTF-8 encoded text
- .It doesn't detect the language. So for Turkish the result might not be exactly correct
- If the length of the UTF-8 byte sequence is different for upper and lower case of a code point, the result may be incorrect for this code point
- .If the string contains a set of bytes that is not UTF-8, then the behavior is undefined

## upperUTF8

- .Converts a string to uppercase, assuming the string contains a set of bytes that make up a UTF-8 encoded text
- .It doesn't detect the language. So for Turkish the result might not be exactly correct
- If the length of the UTF-8 byte sequence is different for upper and lower case of a code point, the result may be incorrect for this code point
- .If the string contains a set of bytes that is not UTF-8, then the behavior is undefined

## isValidUTF8

.Returns 1, if the set of bytes is valid UTF-8 encoded, otherwise 0

## toValidUTF8

Replaces invalid UTF-8 characters by the 💎 (U+FFFD) character. All running in a row invalid characters are collapsed into the one replacement character

```
( toValidUTF8( input_string
```

:Parameters

.input\_string — Any set of bytes represented as the **String** data type object

.Returned value: Valid UTF-8 string

## Example

```
('SELECT toValidUTF8('\x61\xF0\x80\x80\x80b
```

```
└─('toValidUTF8('a💎💎💎b─┐
|           a💎b |
└──────────────────┘ L
```

## reverse

.(Reverses the string (as a sequence of bytes

## reverseUTF8

Reverses a sequence of Unicode code points, assuming that the string contains a set of bytes representing a UTF-8 text. Otherwise, it does something else (it doesn't throw an exception

## (... ,format(pattern, s0, s1

Formatting constant pattern with the string listed in the arguments. **pattern** is a simplified Python format pattern. Format string contains "replacement fields" surrounded by curly braces **{}**. Anything that is not contained in braces is considered literal text, which is copied unchanged to the output. If you need to include a brace character in the literal text, it can be escaped by doubling: **{{** and **}}**. Field names can be numbers (starting from zero) or .(empty (then they are treated as consequence numbers

```
('SELECT format('{1} {0} {1}', 'World', 'Hello
```

```
└─('format('{1} {0} {1}', 'World', 'Hello─┐
|           Hello World Hello |
└──────────────────┘ L
```

```
('SELECT format('{} {}', 'Hello', 'World
```

```
└─('format('{} {}', 'Hello', 'World─┐
|           Hello World |
└──────────────────┘ L
```

## (... ,concat(s1, s2

.Concatenates the strings listed in the arguments, without a separator

## (... ,concatAssumeInjective(s1, s2

Same as **concat**, the difference is that you need to ensure that `concat(s1, s2, s3) -> s4` is injective, it will be used for optimization of GROUP BY

## substring(s, offset, length), mid(s, offset, length), substr(s, offset, length)

Returns a substring starting with the byte from the 'offset' index that is 'length' bytes long. Character indexing starts from one (as in standard SQL). The 'offset' and 'length' arguments must be constants

## (substringUTF8(s, offset, length

The same as 'substring', but for Unicode code points. Works under the assumption that the string contains a set of bytes representing a UTF-8 encoded text. If this assumption is not met, it returns some result (it doesn't throw an exception)

## (appendTrailingCharIfAbsent(s, c

If the 's' string is non-empty and does not contain the 'c' character at the end, it appends the 'c' character to the end

## (convertCharset(s, from, to

.'Returns the string 's' that was converted from the encoding in 'from' to the encoding in 'to

## (base64Encode(s

Encodes 's' string into base64

## (base64Decode(s

.Decode base64-encoded string 's' into original string. In case of failure raises an exception

## (tryBase64Decode(s

.Similar to base64Decode, but in case of error an empty string would be returned

## (endsWith(s, suffix

Returns whether to end with the specified suffix. Returns 1 if the string ends with the specified suffix, otherwise it returns 0

## (startsWith(s, prefix

Returns whether to start with the specified prefix. Returns 1 if the string starts with the specified prefix, otherwise it returns 0

## (trimLeft(s

.Returns a string that removes the whitespace characters on left side

## (trimRight(s

.Returns a string that removes the whitespace characters on right side

## (trimBoth(s

.Returns a string that removes the whitespace characters on either side

## Functions for Searching Strings

The search is case-sensitive by default in all these functions. There are separate variants for case insensitive search

## (position(haystack, needle), locate(haystack, needle

.Search for the substring **needle** in the string **haystack**

.Returns the position (in bytes) of the found substring, starting from 1, or returns 0 if the substring was not found



.For a case-insensitive search, use the function `positionCaseInsensitive`

## `(positionUTF8(haystack, needle`

The same as `position`, but the position is returned in Unicode code points. Works under the assumption that the string contains a set of bytes representing a UTF-8 encoded text. If this assumption is not met, it returns some .(result (it doesn't throw an exception

.For a case-insensitive search, use the function `positionCaseInsensitiveUTF8`

## `([multiSearchAllPositions(haystack, [needle1, needle2, ..., needlen`

.The same as `position`, but returns `Array` of the `positions` for all `needlei`

For a case-insensitive search or/and in UTF-8 format use functions `multiSearchAllPositionsCaseInsensitive`,  
`.multiSearchAllPositionsUTF8`, `multiSearchAllPositionsCaseInsensitiveUTF8`

## `multiSearchFirstPosition(haystack, [needle1, needle2, ...,` `([needlen`

.The same as `position` but returns the leftmost offset of the string `haystack` that is matched to some of the needles

For a case-insensitive search or/and in UTF-8 format use functions `multiSearchFirstPositionCaseInsensitive`,  
`.multiSearchFirstPositionUTF8`, `multiSearchFirstPositionCaseInsensitiveUTF8`

## `([multiSearchFirstIndex(haystack, [needle1, needle2, ..., needlen`

.Returns the index `i` (starting from 1) of the leftmost found `needlei` in the string `haystack` and 0 otherwise

For a case-insensitive search or/and in UTF-8 format use functions `multiSearchFirstIndexCaseInsensitive`,  
`.multiSearchFirstIndexUTF8`, `multiSearchFirstIndexCaseInsensitiveUTF8`

## `([multiSearchAny(haystack, [needle1, needle2, ..., needlen`

.Returns 1, if at least one string `needlei` matches the string `haystack` and 0 otherwise

For a case-insensitive search or/and in UTF-8 format use functions `multiSearchAnyCaseInsensitive`, `multiSearchAnyUTF8`,  
`.multiSearchAnyCaseInsensitiveUTF8`

**Note: in all `multiSearch*` functions the number of needles should be less than 2<sup>8</sup> because of .implementation specification**

## `(match(haystack, pattern`

Checks whether the string matches the `pattern` regular expression. A `re2` regular expression. The `syntax` of the `re2` .regular expressions is more limited than the syntax of the Perl regular expressions

.Returns 0 if it doesn't match, or 1 if it matches

Note that the backslash symbol (`\`) is used for escaping in the regular expression. The same symbol is used for escaping in string literals. So in order to escape the symbol in a regular expression, you must write two .backslashes (`\\`) in a string literal

The regular expression works with the string as if it is a set of bytes. The regular expression can't contain null .bytes

.For patterns to search for substrings in a string, it is better to use `LIKE` or `'position'`, since they work much faster

## `([multiMatchAny(haystack, [pattern1, pattern2, ..., patternn`

The same as `match`, but returns 0 if none of the regular expressions are matched and 1 if any of the patterns matches. It uses `hyperscan` library. For patterns to search substrings in a string, it is better to use `multiSearchAny` .since it works much faster

**Note:** the length of any of the **haystack** string must be less than  $2^{32}$  bytes otherwise the exception is **.thrown**. This restriction takes place because of hyperscan API

**(multiMatchAnyIndex(haystack, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>**

.The same as **multiMatchAny**, but returns any index that matches the haystack

**multiFuzzyMatchAny(haystack, distance, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>**

The same as **multiMatchAny**, but returns 1 if any pattern matches the haystack within a constant **edit distance**. This function is also in an experimental mode and can be extremely slow. For more information see **hyperscan .documentation**

**multiFuzzyMatchAnyIndex(haystack, distance, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>**

.The same as **multiFuzzyMatchAny**, but returns any index that matches the haystack within a constant edit distance

**Note:** **multiFuzzyMatch\*** functions do not support UTF-8 regular expressions, and such expressions are **.treated as bytes because of hyperscan restriction**

**.;Note:** to turn off all functions that use hyperscan, use setting **SET allow\_hyperscan = 0**

**(extract(haystack, pattern**

Extracts a fragment of a string using a regular expression. If 'haystack' doesn't match the 'pattern' regex, an empty string is returned. If the regex doesn't contain subpatterns, it takes the fragment that matches the entire .regex. Otherwise, it takes the fragment that matches the first subpattern

**(extractAll(haystack, pattern**

Extracts all the fragments of a string using a regular expression. If 'haystack' doesn't match the 'pattern' regex, an empty string is returned. Returns an array of strings consisting of all matches to the regex. In general, the behavior is the same as the 'extract' function (it takes the first subpattern, or the entire expression if there isn't a .(subpattern

**like(haystack, pattern), haystack LIKE pattern operator**

.Checks whether a string matches a simple regular expression

**\_** The regular expression can contain the metasympols **%** and

**.(**indicates any quantity of any bytes (including zero characters **%**

**.**indicates any one byte **\_**

Use the backslash (**\**) for escaping metasympols. See the note on escaping in the description of the 'match' .function

.For regular expressions like **%needle%**, the code is more optimal and works as fast as the **position** function

.For other regular expressions, the code is the same as for the 'match' function

**notLike(haystack, pattern), haystack NOT LIKE pattern operator**

.The same thing as 'like', but negative

**(ngramDistance(haystack, needle**

Calculates the 4-gram distance between **haystack** and **needle**: counts the symmetric difference between two multisets of 4-grams and normalizes it by the sum of their cardinalities. Returns float number from 0 to 1 -- the closer to zero, the more strings are similar to each other. If the constant **needle** or **haystack** is more than 32Kb, throws an exception. If some of the non-constant **haystack** or **needle** strings are more than 32Kb, the distance is always one

For case-insensitive search or/and in UTF-8 format use functions **ngramDistanceCaseInsensitive**, **ngramDistanceUTF8**, **.ngramDistanceCaseInsensitiveUTF8**

## (ngramSearch(haystack, needle

Same as **ngramDistance** but calculates the non-symmetric difference between **needle** and **haystack** -- the number of n-grams from needle minus the common number of n-grams normalized by the number of **needle** n-grams. Can be useful for fuzzy string search

For case-insensitive search or/and in UTF-8 format use functions **ngramSearchCaseInsensitive**, **ngramSearchUTF8**, **.ngramSearchCaseInsensitiveUTF8**

**Note: For UTF-8 case we use 3-gram distance. All these are not perfectly fair n-gram distances. We use 2-byte hashes to hash n-grams and then calculate the (non-)symmetric difference between these hash tables -- collisions may occur. With UTF-8 case-insensitive format we do not use fair **tolower** function -- we zero the 5-th bit (starting from zero) of each codepoint byte and first bit of zeroth byte .if bytes more than one -- this works for Latin and mostly for all Cyrillic letters**

## Functions for searching and replacing in strings

### (replaceOne(haystack, pattern, replacement

.Replaces the first occurrence, if it exists, of the 'pattern' substring in 'haystack' with the 'replacement' substring  
.Hereafter, 'pattern' and 'replacement' must be constants

### replaceAll(haystack, pattern, replacement), replace(haystack, (pattern, replacement

.Replaces all occurrences of the 'pattern' substring in 'haystack' with the 'replacement' substring

### (replaceRegexpOne(haystack, pattern, replacement

.Replacement using the 'pattern' regular expression. A re2 regular expression

.Replaces only the first occurrence, if it exists

.A pattern can be specified as 'replacement'. This pattern can include substitutions **\0-\9**

The substitution **\0** includes the entire regular expression. Substitutions **\1-\9** correspond to the subpattern

**\** numbers.To use the **\** character in a template, escape it using

.Also keep in mind that a string literal requires an extra escape

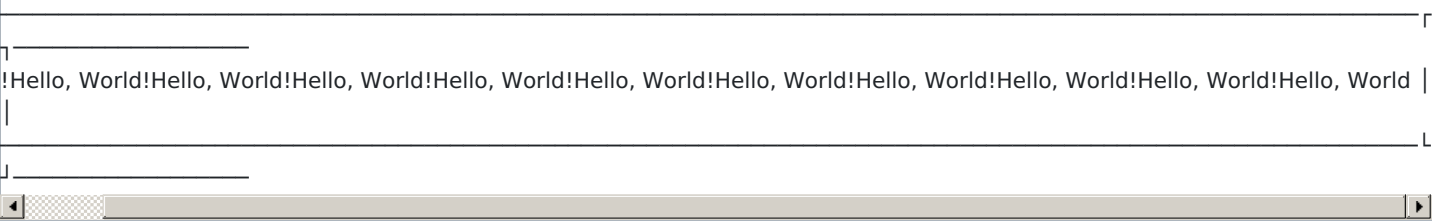
:Example 1. Converting the date to American format

```
SELECT DISTINCT
,EventDate
replaceRegexpOne(toString(EventDate), '\\d{4}-\\d{2}-\\d{2}', '\\2/\\3/\\1') AS res
FROM test.hits
LIMIT 7
FORMAT TabSeparated
```

03/17/2014	2014-03-17
03/18/2014	2014-03-18
03/19/2014	2014-03-19
03/20/2014	2014-03-20
03/21/2014	2014-03-21
03/22/2014	2014-03-22
03/23/2014	2014-03-23

:Example 2. Copying a string ten times

```
SELECT replaceRegexpOne('Hello, World!', '.', '\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0') AS res
```



(replaceRegexpAll(haystack, pattern, replacement

:This does the same thing, but replaces all the occurrences. Example

```
SELECT replaceRegexpAll('Hello, World!', '.', '\\0\\0') AS res
```



As an exception, if a regular expression worked on an empty substring, the replacement is not made more than once  
:Example

```
SELECT replaceRegexpAll('Hello, World!', '^', 'here: ') AS res
```



(regexpQuoteMeta(s

.The function adds a backslash before some predefined characters in the string  
'.' , ':' , '}' , '+' , '\*' , '?' , '[' , ']' , ':' , '\$' , '^' , '(' , ')' , '|' , '\' , 'Predefined characters: '0  
This implementation slightly differs from re2::RE2::QuoteMeta. It escapes zero byte as \0 instead of \x00 and it  
.escapes only required characters  
For more information, see the link: [RE2](#)

Conditional functions

if(cond, then, else), cond ? operator then : else

.Returns then if cond != 0, or else if cond = 0  
.cond must be of type UInt8, and then and else must have the lowest common type  
then and else can be NULL

multif

.Allows you to write the CASE operator more compactly in the query

```
(multif(cond_1, then_1, cond_2, then_2...else
```

:Parameters

- .cond\_N — The condition for the function to return then\_N
- .then\_N — The result of the function when executed
- .else — The result of the function if none of the conditions is met

.The function accepts 2N+1 parameters

Returned values

.The function returns one of the values `then_N` or `else`, depending on the conditions `cond_N`

### Example

Take the table

x		y	
NULL	1		
3	2		

.Run the query `SELECT multilf(isNull(y) x, y < 3, y, NULL) FROM t_null`. Result

(multilf(isNull(y), x, less(y, 3), y, NULL)	
1	
NULL	

## Mathematical functions

All the functions return a Float64 number. The accuracy of the result is close to the maximum precision possible, but the result might not coincide with the machine representable number nearest to the corresponding real number

### (`e`)

.Returns a Float64 number that is close to the number  $e$

### (`pi`)

.Returns a Float64 number that is close to the number  $\pi$

### (`exp(x`

.Accepts a numeric argument and returns a Float64 number close to the exponent of the argument

### (`log(x)`, `ln(x`

.Accepts a numeric argument and returns a Float64 number close to the natural logarithm of the argument

### (`exp2(x`

.Accepts a numeric argument and returns a Float64 number close to 2 to the power of  $x$

### (`log2(x`

.Accepts a numeric argument and returns a Float64 number close to the binary logarithm of the argument

### (`exp10(x`

.Accepts a numeric argument and returns a Float64 number close to 10 to the power of  $x$

### (`log10(x`

.Accepts a numeric argument and returns a Float64 number close to the decimal logarithm of the argument

### (`sqrt(x`

.Accepts a numeric argument and returns a Float64 number close to the square root of the argument

### (`cbrt(x`

.Accepts a numeric argument and returns a Float64 number close to the cubic root of the argument

### (`erf(x`

If 'x' is non-negative, then  $\text{erf}(x / \sigma\sqrt{2})$  is the probability that a random variable having a normal distribution with standard deviation ' $\sigma$ ' takes the value that is separated from the expected value by more than 'x'

:(Example (three sigma rule

```
((SELECT erf(3 / sqrt(2))
--(((erf(divide(3, sqrt(2))
| 0.9973002039367398 |
|_____|
|_____|
```

## (erfc(x

Accepts a numeric argument and returns a Float64 number close to  $1 - \text{erf}(x)$ , but without loss of precision for large 'x' values

## (lgamma(x

.The logarithm of the gamma function

## (tgamma(x

.Gamma function

## (sin(x

.The sine

## (cos(x

.The cosine

## (tan(x

.The tangent

## (asin(x

.The arc sine

## (acos(x

.The arc cosine

## (atan(x

.The arc tangent

## (pow(x, y), power(x, y

.Takes two numeric arguments x and y. Returns a Float64 number close to x to the power of y

## intExp2

.Accepts a numeric argument and returns a UInt64 number close to 2 to the power of x

## intExp10

.Accepts a numeric argument and returns a UInt64 number close to 10 to the power of x

## Rounding functions

### ([floor(x[, N

Returns the largest round number that is less than or equal to x. A round number is a multiple of 1/10N, or the .nearest number of the appropriate data type if 1 / 10N isn't exact  
.N' is an integer constant, optional parameter. By default it is zero, which means to round to an integer'  
.N' may be negative'

.Examples: `floor(123.45, 1) = 123.4`, `floor(123.45, -1) = 120`

.x is any numeric type. The result is a number of the same type

For integer arguments, it makes sense to round with a negative 'N' value (for non-negative 'N', the function .(doesn't do anything

.If rounding causes overflow (for example, `floor(-128, -1)`), an implementation-specific result is returned

## ([ceil(x[, N]), ceiling(x[, N

Returns the smallest round number that is greater than or equal to 'x'. In every other way, it is the same as the .('floor' function (see above

## ([round(x[, N

.Rounds a value to a specified number of decimal places

The function returns the nearest number of the specified order. In case when given number has equal distance to .(surrounding numbers the function returns the number having the nearest even digit (banker's rounding

```
[round(expression [, decimal_places
```

### :Parameters

.**expression** — A number to be rounded. Can be any **expression** returning the numeric **data type**

.**decimal\_places** — An integer value

.If **decimal\_places** > 0 then the function rounds the value to the right of the decimal point

.If **decimal\_places** < 0 then the function rounds the value to the left of the decimal point

If **decimal\_places** = 0 then the function rounds the value to integer. In this case the argument can be

.omitted

### :Returned value

.The rounded number of the same type as the input number

## Examples

### Example of use

```
SELECT number / 2 AS x, round(x) FROM system.numbers LIMIT 3
```

```
--(x--round(divide(number, 2--r
| 0 | 0 | |
| 0 | 0.5 |
| 1 | 1 |
|_ _ _ _ _|_ _ _ _ _|_ _ _ _ _|
```

### Examples of rounding

.Rounding to the nearest number

```
round(3.2, 0) = 3
round(4.1267, 2) = 4.13
round(22,-1) = 20
round(467,-2) = 500
round(-467,-2) = -500
```

.Banker's rounding

```
round(3.5) = 4  
round(4.5) = 4  
round(3.55, 1) = 3.6  
round(3.65, 1) = 3.6
```

## (roundToExp2(num

Accepts a number. If the number is less than one, it returns 0. Otherwise, it rounds the number down to the .nearest (whole non-negative) degree of two

## (roundDuration(num

Accepts a number. If the number is less than one, it returns 0. Otherwise, it rounds the number down to numbers from the set: 1, 10, 30, 60, 120, 180, 240, 300, 600, 1200, 1800, 3600, 7200, 18000, 36000. This function is specific to Yandex.Metrica and used for implementing the report on session length

## (roundAge(num

Accepts a number. If the number is less than 18, it returns 0. Otherwise, it rounds the number down to a number from the set: 18, 25, 35, 45, 55. This function is specific to Yandex.Metrica and used for implementing the report .on user age

## (roundDown(num, arr

Accept a number, round it down to an element in the specified array. If the value is less than the lowest bound, .the lowest bound is returned

# Functions for working with arrays

## empty

.Returns 1 for an empty array, or 0 for a non-empty array  
.The result type is UInt8  
.The function also works for strings

## notEmpty

.Returns 0 for an empty array, or 1 for a non-empty array  
.The result type is UInt8  
.The function also works for strings

## length

.Returns the number of items in the array  
.The result type is UInt64  
.The function also works for strings

emptyArrayUInt8, emptyArrayUInt16, emptyArrayUInt32,  
emptyArrayUInt64

emptyArrayInt8, emptyArrayInt16, emptyArrayInt32,  
emptyArrayInt64

emptyArrayFloat32, emptyArrayFloat64

emptyArrayDate, emptyArrayDateTime

emptyArrayString

.Accepts zero arguments and returns an empty array of the appropriate type

emptyArrayToSingle



.Accepts an empty array and returns a one-element array that is equal to the default value

## (range(N

.Returns an array of numbers from 0 to N-1

Just in case, an exception is thrown if arrays with a total length of more than 100,000,000 elements are created in a data block

## [... ,array(x1, ...), operator [x1

.Creates an array from the function arguments

The arguments must be constants and have types that have the smallest common type. At least one argument must be passed, because otherwise it isn't clear which type of array to create. That is, you can't use this function .(to create an empty array (to do that, use the 'emptyArray\*' function described above

.Returns an 'Array(T)' type result, where 'T' is the smallest common type out of the passed arguments

## arrayConcat

.Combines arrays passed as arguments

```
(arrayConcat(arrays
```

### Parameters

.arrays - Arbitrary number of arguments of **Array** type

•

### Example

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

```
┌─── res ──┐
│ [1,2,3,4,5,6] │
└──────────┘
```

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

.Get the element with the index **n** from the array **arr**. **n** must be any integer type

.Indexes in an array begin from one

Negative indexes are supported. In this case, it selects the corresponding element numbered from the end. For example, **arr[-1]** is the last item in the array

If the index falls outside of the bounds of an array, it returns some default value (0 for numbers, an empty string .(.for strings, etc

## (has(arr, elem

.Checks whether the 'arr' array has the 'elem' element

.Returns 0 if the the element is not in the array, or 1 if it is

.**NULL** is processed as a value

```
(SELECT has([1, 2, NULL], NULL
```

```
┌──(has([1, 2, NULL], NULL)──┐
│ 1                          │
└──────────────────────────┘
```

## hasAll

.Checks whether one array is a subset of another

```
(hasAll(set, subset
```

### Parameters

**.set** – Array of any type with a set of elements

**.subset** – Array of any type with elements that should be tested to be a subset of **set**

### Return values

.if **set** contains all of the elements from **subset** ,1

.otherwise ,0

### Peculiar properties

.An empty array is a subset of any array

.Null processed as a value

.Order of values in both of arrays doesn't matter

### Examples

**.SELECT hasAll([], [])** returns 1

**.SELECT hasAll([1, Null], [Null])** returns 1

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

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

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

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

## hasAny

.Checks whether two arrays have intersection by some elements

```
(hasAny(array1, array2
```

### Parameters

**.array1** – Array of any type with a set of elements

**.array2** – Array of any type with a set of elements

### Return values

.if **array1** and **array2** have one similar element at least ,1

.otherwise ,0

### Peculiar properties

.Null processed as a value

.Order of values in both of arrays doesn't matter

### Examples

**.SELECT hasAny([1], [])** returns 0

**.SELECT hasAny([Null], [Null, 1])** returns 1

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

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

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

## (indexOf(arr, x

.Returns the index of the first 'x' element (starting from 1) if it is in the array, or 0 if it is not

:Example

```
(SELECT indexof([1,3,NULL,NULL],NULL (:  
  
(SELECT indexof([1, 3, NULL, NULL], NULL  
  
┌──(indexof([1, 3, NULL, NULL], NULL──┐  
| 3 |  
└──────────────────────────────────┘
```

.Elements set to **NULL** are handled as normal values

## (countEqual(arr, x

.(Returns the number of elements in the array equal to x. Equivalent to arrayCount (elem -> elem = x, arr

.**NULL** elements are handled as separate values

:Example

```
(SELECT countEqual([1, 2, NULL, NULL], NULL  
  
┌──(countEqual([1, 2, NULL, NULL], NULL──┐  
| 2 |  
└──────────────────────────────────┘
```

## (arrayEnumerate(arr

[ (Returns the array [1, 2, 3, ..., length (arr

This function is normally used with ARRAY JOIN. It allows counting something just once for each array after  
:applying ARRAY JOIN. Example

```
SELECT  
,count() AS Reaches  
countIf(num = 1) AS Hits  
FROM test.hits  
ARRAY JOIN  
,GoalsReached  
arrayEnumerate(GoalsReached) AS num  
WHERE CounterID = 160656  
LIMIT 10
```

```
┌──Reaches──┐──Hits──┐  
| 31406 | 95606 |  
└────────┘────────┘
```

In this example, Reaches is the number of conversions (the strings received after applying ARRAY JOIN), and Hits is the number of pageviews (strings before ARRAY JOIN). In this particular case, you can get the same result in an  
:easier way

```
SELECT  
,sum(length(GoalsReached)) AS Reaches  
count() AS Hits  
FROM test.hits  
(WHERE (CounterID = 160656) AND notEmpty(GoalsReached
```

```
┌──Reaches──┐──Hits──┐  
| 31406 | 95606 |  
└────────┘────────┘
```

This function can also be used in higher-order functions. For example, you can use it to get array indexes for  
:elements that match a condition

# (... ,arrayEnumerateUniq(arr

Returns an array the same size as the source array, indicating for each element what its position is among .elements with the same value

. [For example: arrayEnumerateUniq([10, 20, 10, 30]) = [1, 1, 2, 1

. This function is useful when using ARRAY JOIN and aggregation of array elements

. Example

```
SELECT
,Goals.ID AS GoalID
,sum(Sign) AS Reaches
,sumIf(Sign, num = 1) AS Visits
FROM test.visits
ARRAY JOIN
,Goals
arrayEnumerateUniq(Goals.ID) AS num
WHERE CounterID = 160656
GROUP BY GoalID
ORDER BY Reaches DESC
LIMIT 10
```

GoalID	Reaches	Visits
1097	3214	53225
1097	3188	2825062
488	2803	56600
365	2401	1989037
910	2396	2830064
373	2372	1113562
812	2262	3270895
345	2262	1084657
799	2260	56599
812	2256	3271094

In this example, each goal ID has a calculation of the number of conversions (each element in the Goals nested data structure is a goal that was reached, which we refer to as a conversion) and the number of sessions. Without ARRAY JOIN, we would have counted the number of sessions as sum(Sign). But in this particular case, the rows were multiplied by the nested Goals structure, so in order to count each session one time after this, we apply a .condition to the value of the arrayEnumerateUniq(Goals.ID) function

The arrayEnumerateUniq function can take multiple arrays of the same size as arguments. In this case, .uniqueness is considered for tuples of elements in the same positions in all the arrays

```
SELECT arrayEnumerateUniq([1, 1, 1, 2, 2, 2], [1, 1, 2, 1, 1, 2]) AS res
```

res
[1,2,1,1,2,1]

This is necessary when using ARRAY JOIN with a nested data structure and further aggregation across multiple .elements in this structure

## arrayPopBack

. Removes the last item from the array

```
(arrayPopBack(array
```

### Parameters

. array – Array

### Example

```
SELECT arrayPopBack([1, 2, 3]) AS res
```

```
┌──res─┐
│ [1,2] │
└──────┘
```

## arrayPopFront

.Removes the first item from the array

```
(arrayPopFront(array
```

### Parameters

**.array** – Array

- 

### Example

```
SELECT arrayPopFront([1, 2, 3]) AS res
```

```
┌──res─┐
│ [2,3] │
└──────┘
```

## arrayPushBack

.Adds one item to the end of the array

```
(arrayPushBack(array, single_value
```

### Parameters

**.array** – Array

**single\_value** – A single value. Only numbers can be added to an array with numbers, and only strings can be added to an array of strings. When adding numbers, ClickHouse automatically sets the **single\_value** type for the data type of the array. For more information about the types of data in ClickHouse, see "[Data types](#)". Can be **NULL**. The function adds a **NULL** element to an array, and the type of array elements converts to **Nullable**

- 

- 

### Example

```
SELECT arrayPushBack(['a'], 'b') AS res
```

```
┌──res─┐
│ ['a','b'] │
└──────┘
```

## arrayPushFront

.Adds one element to the beginning of the array

```
(arrayPushFront(array, single_value
```

### Parameters

**.array** – Array

**single\_value** – A single value. Only numbers can be added to an array with numbers, and only strings can be added to an array of strings. When adding numbers, ClickHouse automatically sets the **single\_value** type for the data type of the array. For more information about the types of data in ClickHouse, see "[Data types](#)". Can be **NULL**. The function adds a **NULL** element to an array, and the type of array elements converts to **Nullable**

- 

- 

### Example

```
SELECT arrayPushBack(['b'], 'a') AS res
```

res
['a','b']

# arrayResize

.Changes the length of the array

```
([arrayResize(array, size[, extender
```

## :Parameters

- .array — Array
- .size — Required length of the array
  - If size is less than the original size of the array, the array is truncated from the right
  - If size is larger than the initial size of the array, the array is extended to the right with extender values or default values for the data type of the array items
- .extender — Value for extending an array. Can be NULL

## :Returned value

.An array of length size

## Examples of calls

```
(SELECT arrayResize([1], 3
--(arrayResize([1], 3--
| [1,0,0] |
|_____|
```

```
(SELECT arrayResize([1], 3, NULL
--(arrayResize([1], 3, NULL--
| [NULL,NULL,1] |
|_____|
```

# arraySlice

.Returns a slice of the array

```
([arraySlice(array, offset[, length
```

## Parameters

- .array - Array of data
- offset - Indent from the edge of the array. A positive value indicates an offset on the left, and a negative value is an indent on the right. Numbering of the array items begins with 1
- length - The length of the required slice. If you specify a negative value, the function returns an open slice .[offset, array\_length - length). If you omit the value, the function returns the slice [offset, the\_end\_of\_array

## Example

```
SELECT arraySlice([1, 2, NULL, 4, 5], 2, 3) AS res
```

res
[NULL,4,2]

.Array elements set to NULL are handled as normal values

(... ,arraySort([func,] arr

Sorts the elements of the **arr** array in ascending order. If the **func** function is specified, sorting order is determined by the result of the **func** function applied to the elements of the array. If **func** accepts multiple arguments, the **arraySort** function is passed several arrays that the arguments of **func** will correspond to. Detailed examples are shown at the end of **arraySort** description

:Example of integer values sorting

```
;([SELECT arraySort([1, 3, 3, 0
```

```
]-([arraySort([1, 3, 3, 0]-r
|      [0,1,3,3] |
|_____L
```

:Example of string values sorting

```
;(['!', 'SELECT arraySort(['hello', 'world
```

```
]-(['!', 'arraySort(['hello', 'world'-r
|      ['hello', 'world', '!'] |
|_____L
```

:Consider the following sorting order for the **NULL**, **NaN** and **Inf** values

```
;([SELECT arraySort([1, nan, 2, NULL, 3, nan, -4, NULL, inf, -inf
```

```
]-([arraySort([1, nan, 2, NULL, 3, nan, -4, NULL, inf, -inf]-r
|      [inf,-4,1,2,3,inf,nan,nan,NULL,NULL-] |
|_____L
```

.**Inf** values are first in the array-

.**NULL** values are last in the array

.**NaN** values are right before **NULL**

.**Inf** values are right before **NaN**

- 
- 
- 
- 

Note that **arraySort** is a **higher-order function**. You can pass a lambda function to it as the first argument. In this case, sorting order is determined by the result of the lambda function applied to the elements of the array

:Let's consider the following example

```
;SELECT arraySort((x) -> -x, [1, 2, 3]) as res
```

```
|-res-r
| [3,2,1] |
|_____L
```

For each element of the source array, the lambda function returns the sorting key, that is, [1 -> -1, 2 -> -2, 3 -> -3]. Since the **arraySort** function sorts the keys in ascending order, the result is [3, 2, 1]. Thus, the **(x) -> -x** lambda function sets the **descending order** in a sorting

The lambda function can accept multiple arguments. In this case, you need to pass the **arraySort** function several arrays of identical length that the arguments of lambda function will correspond to. The resulting array will consist of elements from the first input array; elements from the next input array(s) specify the sorting keys. For example

```
;SELECT arraySort((x, y) -> y, ['hello', 'world'], [2, 1]) as res
```

```
|-res-r
| ['world', 'hello'] |
|_____L
```

Here, the elements that are passed in the second array ([2, 1]) define a sorting key for the corresponding element from the source array (['hello', 'world']), that is, ['hello' -> 2, 'world' -> 1]. Since the lambda function doesn't use **x**, actual values of the source array don't affect the order in the result. So, 'hello' will be the second element in the result, and 'world' will be the first

.Other examples are shown below

```
;SELECT arraySort((x, y) -> y, [0, 1, 2], ['c', 'b', 'a']) as res
```

res
[2,1,0]

```
;SELECT arraySort((x, y) -> -y, [0, 1, 2], [1, 2, 3]) as res
```

res
[2,1,0]

Note

.To improve sorting efficiency, the **Schwartzian transform** is used

## (... ,arrayReverseSort([func,] arr

Sorts the elements of the **arr** array in descending order. If the **func** function is specified, **arr** is sorted according to the result of the **func** function applied to the elements of the array, and then the sorted array is reversed. If **func** accepts multiple arguments, the **arrayReverseSort** function is passed several arrays that the arguments of **func** will correspond to. Detailed examples are shown at the end of **arrayReverseSort** description

:Example of integer values sorting

```
;([SELECT arrayReverseSort([1, 3, 3, 0
```

arrayReverseSort([1, 3, 3, 0
[3,3,1,0]

:Example of string values sorting

```
;(['!', 'SELECT arrayReverseSort(['hello', 'world
```

arrayReverseSort(['hello', 'world
['!', 'world', 'hello']

:Consider the following sorting order for the **NULL**, **NaN** and **Inf** values

```
;SELECT arrayReverseSort([1, nan, 2, NULL, 3, nan, -4, NULL, inf, -inf]) as res
```

res
[inf,3,2,1,-4,-inf,nan,nan,NULL,NULL]

- .**Inf** values are first in the array
- .**NULL** values are last in the array
- .**NaN** values are right before **NULL**
- .**Inf** values are right before **NaN**

Note that the **arrayReverseSort** is a **higher-order function**. You can pass a lambda function to it as the first argument. Example is shown below

```
;SELECT arrayReverseSort((x) -> -x, [1, 2, 3]) as res
```

res
[1,2,3]

:The array is sorted in the following way



- At first, the source array ([1, 2, 3]) is sorted according to the result of the lambda function applied to the elements of the array. The result is an array [3, 2, 1].  
 Array that is obtained on the previous step, is reversed. So, the final result is [1, 2, 3].

The lambda function can accept multiple arguments. In this case, you need to pass the `arrayReverseSort` function several arrays of identical length that the arguments of lambda function will correspond to. The resulting array will consist of elements from the first input array; elements from the next input array(s) specify the sorting keys. For example

```
;SELECT arrayReverseSort((x, y) -> y, ['hello', 'world'], [2, 1]) as res
```

```
┌─── res ──┐
│ ['hello','world'] │
└──────────┘
```

In this example, the array is sorted in the following way

- At first, the source array (['hello', 'world']) is sorted according to the result of the lambda function applied to the elements of the arrays. The elements that are passed in the second array ([2, 1]), define the sorting keys for corresponding elements from the source array. The result is an array ['world', 'hello'].  
 Array that was sorted on the previous step, is reversed. So, the final result is ['hello', 'world'].

Other examples are shown below

```
;SELECT arrayReverseSort((x, y) -> y, [4, 3, 5], ['a', 'b', 'c']) AS res
```

```
┌─── res ──┐
│ [5,3,4] │
└──────────┘
```

```
;SELECT arrayReverseSort((x, y) -> -y, [4, 3, 5], [1, 2, 3]) AS res
```

```
┌─── res ──┐
│ [4,3,5] │
└──────────┘
```

## (... ,arrayUniq(arr

If one argument is passed, it counts the number of different elements in the array.  
 If multiple arguments are passed, it counts the number of different tuples of elements at corresponding positions in multiple arrays.

(If you want to get a list of unique items in an array, you can use `arrayReduce('groupUniqArray', arr`

## (arrayJoin(arr

"A special function. See the section ["ArrayJoin function"](#)

## (arrayDifference(arr

Takes an array, returns an array with the difference between all pairs of neighboring elements. For example

```
([SELECT arrayDifference([1, 2, 3, 4
```

```
┌──([arrayDifference([1, 2, 3, 4─┐
│ [0,1,1,1] │
└──────────┘
```

## (arrayDistinct(arr

Takes an array, returns an array containing the different elements in all the arrays. For example

```
([SELECT arrayDistinct([1, 2, 2, 3, 1
```

```

┌--(arrayDistinct([1, 2, 2, 3, 1],
|                  [1,2,3])
└-----┘

```

## (arrayEnumerateDense(arr

Returns an array of the same size as the source array, indicating where each element first appears in the source array. For example: arrayEnumerateDense([10,20,10,30]) = [1,2,1,3]

## (arrayIntersect(arr

:Takes an array, returns the intersection of all array elements. For example

```

SELECT
,arrayIntersect([1, 2], [1, 3], [2, 3]) AS no_intersect
,arrayIntersect([1, 2], [1, 3], [1, 4]) AS intersect

```

```

┌--no_intersect└--intersect┐
|      [1] |      [] |
└-----┘

```

## (... ,arrayReduce(agg\_func, arr1

Applies an aggregate function to array and returns its result.If aggregate function has multiple arguments, then this function can be applied to multiple arrays of the same size

arrayReduce('agg\_func', arr1, ...) - apply the aggregate function **agg\_func** to arrays **arr1....** If multiple arrays passed, then elements on corresponding positions are passed as multiple arguments to the aggregate function. For example: SELECT arrayReduce('max', [1,2,3]) = 3

## (arrayReverse(arr

Returns an array of the same size as the source array, containing the result of inverting all elements of the source array

# Functions for splitting and merging strings and arrays

## (splitByChar(separator, s

Splits a string into substrings separated by 'separator'.'separator' must be a string constant consisting of exactly one character

Returns an array of selected substrings. Empty substrings may be selected if the separator occurs at the beginning or end of the string, or if there are multiple consecutive separators

## (splitByString(separator, s

.The same as above, but it uses a string of multiple characters as the separator. The string must be non-empty

## ([arrayStringConcat(arr[, separator

Concatenates the strings listed in the array with the separator.'separator' is an optional parameter: a constant string, set to an empty string by default  
.Returns the string

## (alphaTokens(s

.Selects substrings of consecutive bytes from the ranges a-z and A-Z.Returns an array of substrings

**:Example**

```
('SELECT alphaTokens('abca1abc
--('alphaTokens('abca1abc--r
|   ['abca','abc'] |
|_____L
```

## Bit functions

Bit functions work for any pair of types from UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64, Float32, or Float64

The result type is an integer with bits equal to the maximum bits of its arguments. If at least one of the arguments is signed, the result is a signed number. If an argument is a floating-point number, it is cast to Int64

(bitAnd(a, b

(bitOr(a, b

(bitXor(a, b

(bitNot(a

(bitShiftLeft(a, b

(bitShiftRight(a, b

(bitRotateLeft(a, b

(bitRotateRight(a, b

(bitTest(a, b

(bitTestAll(a, b

(bitTestAny(a, b

## Bitmap functions

Bitmap functions work for two bitmaps Object value calculation, it is to return new bitmap or cardinality while using formula calculation, such as and, or, xor, and not, etc

There are 2 kinds of construction methods for Bitmap Object. One is to be constructed by aggregation function groupBitmap with -State, the other is to be constructed by Array Object. It is also to convert Bitmap Object to Array Object

RoaringBitmap is wrapped into a data structure while actual storage of Bitmap objects. When the cardinality is less than or equal to 32, it uses Set objet. When the cardinality is greater than 32, it uses RoaringBitmap object. That is why storage of low cardinality set is faster

.For more information on RoaringBitmap, see: [CRoaring](#)

## bitmapBuild

.Build a bitmap from unsigned integer array

```
(bitmapBuild(array
```

### Parameters

.array - unsigned integer array

### Example

```
(SELECT bitmapBuild([1, 2, 3, 4, 5]) AS res, toTypeName(res
```

```
(([res—toTypeName(bitmapBuild([1, 2, 3, 4, 5—r
| (AggregateFunction(groupBitmap, UInt8 | |
| ) L
])
```

## bitmapToArray

.Convert bitmap to integer array

```
(bitmapToArray(bitmap
```

### Parameters

.**bitmap** – bitmap object

•

### Example

```
SELECT bitmapToArray(bitmapBuild([1, 2, 3, 4, 5])) AS res
```

```
res—r
| [1,2,3,4,5] |
|_____L
```

## bitmapHasAny

.Checks whether two bitmaps have intersection by some elements

```
(bitmapHasAny(bitmap1, bitmap2
```

### Parameters

.**bitmap\*** – bitmap object

•

### Return values

.if **bitmap1** and **bitmap2** have one similar element at least ,**1**

•

.otherwise ,**0**

•

### Example

```
SELECT bitmapHasAny(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res
```

```
res—r
| 1 |
|_____L
```

## bitmapHasAll

Analogous to **hasAll(array, array)** returns 1 if the first bitmap contains all the elements of the second one, 0  
.otherwise

.If the second argument is an empty bitmap then returns 1

```
(bitmapHasAll(bitmap,bitmap
```

### Parameters

.**bitmap** – bitmap object

•

### Example

```
SELECT bitmapHasAll(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res
```

```
res—r
| 0 |
|_____L
```

## bitmapAnd

.Two bitmap and calculation, the result is a new bitmap

```
(bitmapAnd(bitmap,bitmap
```

Parameters

.**bitmap** - bitmap object •

Example

```
SELECT bitmapToArray(bitmapAnd(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res
```

```
┌──res──┐
│ [3] │
└──────┘
```

# bitmapOr

.Two bitmap or calculation, the result is a new bitmap

```
(bitmapOr(bitmap,bitmap
```

Parameters

.**bitmap** - bitmap object •

Example

```
SELECT bitmapToArray(bitmapOr(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res
```

```
┌────────res──┐
│ [1,2,3,4,5] │
└──────────────┘
```

# bitmapXor

.Two bitmap xor calculation, the result is a new bitmap

```
(bitmapXor(bitmap,bitmap
```

Parameters

.**bitmap** - bitmap object •

Example

```
SELECT bitmapToArray(bitmapXor(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res
```

```
┌────────res──┐
│ [1,2,4,5] │
└──────────────┘
```

# bitmapAndnot

.Two bitmap andnot calculation, the result is a new bitmap

```
(bitmapAndnot(bitmap,bitmap
```

Parameters

.**bitmap** - bitmap object •

Example

```
SELECT bitmapToArray(bitmapAndnot(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res
```

res
[ 1,2 ]

## bitmapCardinality

.Retrun bitmap cardinality of type UInt64

```
(bitmapCardinality(bitmap
```

### Parameters

.**bitmap** – bitmap object

### Example

```
SELECT bitmapCardinality(bitmapBuild([1, 2, 3, 4, 5])) AS res
```

res
5

## bitmapAndCardinality

.Two bitmap and calculation, return cardinality of type UInt64

```
(bitmapAndCardinality(bitmap,bitmap
```

### Parameters

.**bitmap** – bitmap object

### Example

```
;SELECT bitmapAndCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res
```

res
1

## bitmapOrCardinality

.Two bitmap or calculation, return cardinality of type UInt64

```
(bitmapOrCardinality(bitmap,bitmap
```

### Parameters

.**bitmap** – bitmap object

### Example

```
;SELECT bitmapOrCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res
```

res
5

## bitmapXorCardinality

.Two bitmap xor calculation, return cardinality of type UInt64

```
(bitmapXorCardinality(bitmap,bitmap
```

### Parameters

.**bitmap** – bitmap object

## Example

```
;SELECT bitmapXorCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res
```

```
┌--res--┐
│ 4 │
└-----┘
```

## bitmapAndnotCardinality

.Two bitmap andnot calculation, return cardinality of type UInt64

```
(bitmapAndnotCardinality(bitmap,bitmap
```

## Parameters

.**bitmap** – bitmap object

•

## Example

```
;SELECT bitmapAndnotCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res
```

```
┌--res--┐
│ 2 │
└-----┘
```

## Hash functions

.Hash functions can be used for deterministic pseudo-random shuffling of elements

## halfMD5

**Interprets** all the input parameters as strings and calculates the MD5 hash value for each of them. Then combines hashes. Then from the resulting string, takes the first 8 bytes of the hash and interprets them as **UInt64** in big-endian byte order

```
(... ,halfMD5(par1
```

.(The function works relatively slow (5 million short strings per second per processor core

.Consider using the **sipHash64** function instead

## Parameters

.The function takes a variable number of input parameters. Parameters can be any of the **supported data types**

## Returned Value

.Hash value having the **UInt64** data type

## Example

```
SELECT halfMD5(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS halfMD5hash, toTypeName(halfMD5hash) AS type
```

```
┌-----halfMD5hash-----type-----┐
│ UInt64 │ 186182704141653334 │
└-----┘
```

## MD5

.(Calculates the MD5 from a string and returns the resulting set of bytes as FixedString(16

If you don't need MD5 in particular, but you need a decent cryptographic 128-bit hash, use the 'sipHash128'

.function instead

.(If you want to get the same result as output by the md5sum utility, use lower(hex(MD5(s

# sipHash64

.Produces 64-bit **SipHash** hash value

```
(...,sipHash64(par1
```

This function **interprets** all the input parameters as strings and calculates the hash value for each of them. Then .combines hashes

.This is a cryptographic hash function. It works at least three times faster than the **MD5** function

## Parameters

.The function takes a variable number of input parameters. Parameters can be any of the **supported data types**

## Returned Value

.Hash value having the **UInt64** data type

## Example

```
SELECT sipHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS SipHash, toTypeName(SipHash) AS type
```

SipHash	type
UInt64	13726873534472839665

# sipHash128

.Calculates SipHash from a string

.(Accepts a String-type argument. Returns FixedString(16)

.Differs from sipHash64 in that the final xor-folding state is only done up to 128 bits

# cityHash64

.Produces 64-bit hash value

```
(...,cityHash64(par1
```

This is the fast non-cryptographic hash function. It uses **CityHash** algorithm for string parameters and implementation-specific fast non-cryptographic hash function for the parameters with other data types. To get the .final result, the function uses the CityHash combinator

## Parameters

.The function takes a variable number of input parameters. Parameters can be any of the **supported data types**

## Returned Value

.Hash value having the **UInt64** data type

## Examples

:Call example

```
SELECT cityHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS CityHash, toTypeName(CityHash) AS type
```

CityHash	type
UInt64	12072650598913549138

:The following example shows how to compute the checksum of the entire table with accuracy up to the row order

```
SELECT sum(cityHash64(*)) FROM table
```



## intHash32

.Calculates a 32-bit hash code from any type of integer

.This is a relatively fast non-cryptographic hash function of average quality for numbers

## intHash64

.Calculates a 64-bit hash code from any type of integer

.It works faster than intHash32. Average quality

## SHA1

## SHA224

## SHA256

Calculates SHA-1, SHA-224, or SHA-256 from a string and returns the resulting set of bytes as FixedString(20), .FixedString(28), or FixedString(32

The function works fairly slowly (SHA-1 processes about 5 million short strings per second per processor core, .(while SHA-224 and SHA-256 process about 2.2 million

.We recommend using this function only in cases when you need a specific hash function and you can't select it

Even in these cases, we recommend applying the function offline and pre-calculating values when inserting them .into the table, instead of applying it in SELECTS

## (URLHash(url[, N

A fast, decent-quality non-cryptographic hash function for a string obtained from a URL using some type of .normalization

.URLHash(s) – Calculates a hash from a string without one of the trailing symbols /, ? or # at the end, if present

URLHash(s, N) – Calculates a hash from a string up to the N level in the URL hierarchy, without one of the trailing .symbols /, ? or # at the end, if present

.Levels are the same as in URLHierarchy. This function is specific to Yandex.Metrica

## farmHash64

.Produces a 64-bit FarmHash hash value

```
(... ,farmHash64(par1
```

.The function uses the Hash64 method from all available methods

### Parameters

.The function takes a variable number of input parameters. Parameters can be any of the supported data types

### Returned Value

.Hash value having the UInt64 data type

### Example

```
SELECT farmHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS FarmHash, toTypeName(FarmHash) AS type
```

```
| FarmHash | type |
| UInt64   | 17790458267262532859 |
```

## javaHash

.Calculates JavaHash from a string

.Accepts a String-type argument. Returns Int32

For more information, see the link: [JavaHash](#)

# hiveHash

- .Calculates HiveHash from a string
- .Accepts a String-type argument. Returns Int32
- .Same as for [JavaHash](#), except that the return value never has a negative number

# metroHash64

- .Produces a 64-bit [MetroHash](#) hash value

```
(... ,metroHash64(par1
```

## Parameters

- .The function takes a variable number of input parameters. Parameters can be any of the [supported data types](#)

## Returned Value

- .Hash value having the [UInt64](#) data type

## Example

```
SELECT metroHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS MetroHash, toTypeName(MetroHash) AS type
```

MetroHash		type
UInt64	14235658766382344533	

# jumpConsistentHash

- .Calculates JumpConsistentHash form a UInt64
- .Accepts a UInt64-type argument. Returns Int32
- For more information, see the link: [JumpConsistentHash](#)

# murmurHash2\_32, murmurHash2\_64

- .Produces a [MurmurHash2](#) hash value

```
(... ,murmurHash2_32(par1  
(... ,murmurHash2_64(par1
```

## Parameters

- .Both functions take a variable number of input parameters. Parameters can be any of the [supported data types](#)

## Returned Value

- .The [murmurHash2\\_32](#) function returns hash value having the [UInt32](#) data type
- .The [murmurHash2\\_64](#) function returns hash value having the [UInt64](#) data type

## Example

```
SELECT murmurHash2_64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS MurmurHash2, toTypeName(MurmurHash2) AS type
```

MurmurHash2		type
UInt64	11832096901709403633	

# murmurHash3\_32, murmurHash3\_64

- .Produces a [MurmurHash3](#) hash value

```
(... ,murmurHash3_32(par1  
(... ,murmurHash3_64(par1
```

### Parameters

.Both functions take a variable number of input parameters. Parameters can be any of the [supported data types](#)

### Returned Value

- .The [murmurHash3\\_32](#) function returns hash value having the [UInt32](#) data type
- .The [murmurHash3\\_64](#) function returns hash value having the [UInt64](#) data type

### Example

```
SELECT murmurHash3_32(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS MurmurHash3,  
toTypeName(MurmurHash3) AS type
```

```
┌───MurmurHash3───┐ type ─┐  
│ UInt32 │ 2152717 │  
└──────────┘└───┘
```

## murmurHash3\_128

.Produces a 128-bit [MurmurHash3](#) hash value

```
( murmurHash3_128( expr
```

### Parameters

- .[expr](#) — [Expressions](#) returning [String](#)-typed value

### Returned Value

.Hash value having [FixedString\(16\)](#) data type

### Example

```
SELECT murmurHash3_128('example_string') AS MurmurHash3, toTypeName(MurmurHash3) AS type
```

```
┌───MurmurHash3───┐ type ─┐  
│ (S5KT~~~q │ FixedString(16"416  
└──────────┘└───┘
```

## xxHash32, xxHash64

- .Calculates xxHash from a string
  - .Accepts a String-type argument. Returns UInt64 Or UInt32
- For more information, see the link: [xxHash](#)

## Functions for generating pseudo-random numbers

.Non-cryptographic generators of pseudo-random numbers are used

- .All the functions accept zero arguments or one argument
  - .If an argument is passed, it can be any type, and its value is not used for anything
- The only purpose of this argument is to prevent common subexpression elimination, so that two different instances of the same function return different columns with different random numbers

### rand

- .Returns a pseudo-random UInt32 number, evenly distributed among all UInt32-type numbers
- .Uses a linear congruential generator

### rand64

.Returns a pseudo-random UInt64 number, evenly distributed among all UInt64-type numbers  
.Uses a linear congruential generator

## randConstant

.Returns a pseudo-random UInt32 number, The value is one for different blocks

## Encoding functions

### hex

Accepts arguments of types: **String**, **unsigned integer**, **Date**, or **DateTime**. Returns a string containing the argument's hexadecimal representation. Uses uppercase letters **A-F**. Does not use **0x** prefixes or **h** suffixes. For strings, all bytes are simply encoded as two hexadecimal numbers. Numbers are converted to big endian ("human readable") format. For numbers, older zeros are trimmed, but only by entire bytes. For example, **hex (1) = '01'**. **Date** is encoded as the number of days since the beginning of the Unix epoch. **DateTime** is encoded as the number of seconds since the beginning of the Unix epoch

### (unhex(str

Accepts a string containing any number of hexadecimal digits, and returns a string containing the corresponding bytes. Supports both uppercase and lowercase letters A-F. The number of hexadecimal digits does not have to be even. If it is odd, the last digit is interpreted as the younger half of the 00-0F byte. If the argument string contains anything other than hexadecimal digits, some implementation-defined result is returned (an exception isn't thrown)  
.If you want to convert the result to a number, you can use the 'reverse' and 'reinterpretAsType' functions

### (UUIDStringToNum(str

Accepts a string containing 36 characters in the format **123e4567-e89b-12d3-a456-426655440000**, and returns it as a  
.set of bytes in a FixedString(16

### (UUIDNumToString(str

.Accepts a FixedString(16) value. Returns a string containing 36 characters in text format

### (bitmaskToList(num

Accepts an integer. Returns a string containing the list of powers of two that total the source number when summed. They are comma-separated without spaces in text format, in ascending order

### (bitmaskToArray(num

Accepts an integer. Returns an array of UInt64 numbers containing the list of powers of two that total the source number when summed. Numbers in the array are in ascending order

## Functions for working with UUID

.The functions for working with UUID are listed below

### generateUUIDv4

.Generates the **UUID** of **version 4**

```
()generateUUIDv4
```

#### Returned value

.The UUID type value

#### Usage example

.This example demonstrates creating a table with the UUID type column and inserting a value into the table

```
CREATE TABLE t_uuid (x UUID) ENGINE=TinyLog (:

)INSERT INTO t_uuid SELECT generateUUIDv4 (:

SELECT * FROM t_uuid (:

┌──x──┐
│ f4bf890f-f9dc-4332-ad5c-0c18e73f28e9 │
└──┬──┘
```

## (toUUID (x

.Converts String type value to UUID type

```
(toUUID(String
```

### Returned value

.The UUID type value

### Usage example

```
SELECT toUUID('61f0c404-5cb3-11e7-907b-a6006ad3dba0') AS uuid (:

┌──uuid──┐
│ 61f0c404-5cb3-11e7-907b-a6006ad3dba0 │
└──┬──┘
```

## UUIDStringToNum

Accepts a string containing 36 characters in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, and returns it as a set  
.of bytes in a FixedString(16

```
((UUIDStringToNum(String
```

### Returned value

(FixedString(16

### Usage examples

```
SELECT (:
,612f3c40-5d3b-217e-707b-6a546a3d7b29' AS uuid'
UUIDStringToNum(uuid) AS bytes

┌─────uuid──┐┌──bytes──┐
│ {}=612f3c40-5d3b-217e-707b-6a546a3d7b29 │ a/<@];!~p{jTj │
└──┬──┘└──┬──┘
```

## UUIDNumToString

.Accepts a FixedString(16) value, and returns a string containing 36 characters in text format

```
((UUIDNumToString(FixedString(16
```

### Returned value

.String

### Usage example

```
SELECT
,a/<@];!~p{jTj={})' AS bytes'
UUIDNumToString(toFixedString(bytes, 16)) AS uuid
```

```
|-----bytes-----|-----uuid-----|
| a/<@];!~p{jTj={}) | 612f3c40-5d3b-217e-707b-6a546a3d7b29 |
```

## See also

[dictGetUUID](#)

[dictGetUUIDOrDefault](#)

- 
- 

## Functions for working with URLs

.All these functions don't follow the RFC. They are maximally simplified for improved performance

### Functions that extract part of a URL

.If there isn't anything similar in a URL, an empty string is returned

#### protocol

...Returns the protocol. Examples: http, ftp, mailto, magnet

#### domain

.Gets the domain

#### domainWithoutWWW

.Returns the domain and removes no more than one 'www.' from the beginning of it, if present

#### topLevelDomain

.Returns the top-level domain. Example: .ru

#### firstSignificantSubdomain

Returns the "first significant subdomain". This is a non-standard concept specific to Yandex.Metrica. The first significant subdomain is a second-level domain if it is 'com', 'net', 'org', or 'co'. Otherwise, it is a third-level domain. For example, `firstSignificantSubdomain('https://news.yandex.ru/') = 'yandex '`, `firstSignificantSubdomain('https://news.yandex.com.tr/') = 'yandex '`. The list of "insignificant" second-level domains and other implementation details may change in the future

#### cutToFirstSignificantSubdomain

Returns the part of the domain that includes top-level subdomains up to the "first significant subdomain" (see the .(explanation above

.'For example, `cutToFirstSignificantSubdomain('https://news.yandex.com.tr/') = 'yandex.com.tr`

#### path

.Returns the path. Example: `/top/news.html` The path does not include the query string

#### pathFull

The same as above, but including query string and fragment. Example: `/top/news.html?page=2#comments`

#### queryString

Returns the query string. Example: `page=1&lr=213`. query-string does not include the initial question mark, as .# well as # and everything after

#### fragment

.Returns the fragment identifier. fragment does not include the initial hash symbol

## queryStringAndFragment

.Returns the query string and fragment identifier. Example: page=1#29390

## (extractURLParameter(URL, name

Returns the value of the 'name' parameter in the URL, if present. Otherwise, an empty string. If there are many parameters with this name, it returns the first occurrence. This function works under the assumption that the .parameter name is encoded in the URL exactly the same way as in the passed argument

## (extractURLParameters(URL

Returns an array of name=value strings corresponding to the URL parameters. The values are not decoded in any .way

## (extractURLParameterNames(URL

Returns an array of name strings corresponding to the names of URL parameters. The values are not decoded in .any way

## (URLHierarchy(URL

Returns an array containing the URL, truncated at the end by the symbols /,? in the path and query-string. Consecutive separator characters are counted as one. The cut is made in the position after all the consecutive .separator characters

## (URLPathHierarchy(URL

The same as above, but without the protocol and host in the result. The / element (root) is not included. Example: .the function is used to implement tree reports the URL in Yandex. Metric

```
= ('URLPathHierarchy('https://example.com/browse/CONV-6788
]
, '/browse/'
'browse/CONV-6788/'
[
```

## (decodeURLComponent(URL

.Returns the decoded URL

:Example

```
;SELECT decodeURLComponent('http://127.0.0.1:8123/?query=SELECT%201%3B') AS DecodedURL
```

```
|-----DecodedURL-----|
| ;http://127.0.0.1:8123/?query=SELECT 1 |
|-----L
```

## .Functions that remove part of a URL

.If the URL doesn't have anything similar, the URL remains unchanged

### cutWWW

.Removes no more than one 'www.' from the beginning of the URL's domain, if present

### cutQueryString

.Removes query string. The question mark is also removed

### cutFragment

.Removes the fragment identifier. The number sign is also removed

### cutQueryStringAndFragment

.Removes the query string and fragment identifier. The question mark and number sign are also removed

## (cutURLParameter(URL, name

Removes the 'name' URL parameter, if present. This function works under the assumption that the parameter .name is encoded in the URL exactly the same way as in the passed argument

## Functions for working with IP addresses

### (IPv4NumToString(num

Takes a UInt32 number. Interprets it as an IPv4 address in big endian. Returns a string containing the .(corresponding IPv4 address in the format A.B.C.d (dot-separated numbers in decimal form

### (IPv4StringToNum(s

.The reverse function of IPv4NumToString. If the IPv4 address has an invalid format, it returns 0

### (IPv4NumToStringClassC(num

.Similar to IPv4NumToString, but using xxx instead of the last octet

:Example

```
SELECT
,IPv4NumToStringClassC(ClientIP) AS k
count() AS c
FROM test.hits
GROUP BY k
ORDER BY c DESC
LIMIT 10
```

```
┌──k──┐┌──c──┐
│ xxx │ 26238.83.149.9 │
│ xxx │ 26074.217.118.81 │
│ xxx │ 25481.213.87.129 │
│ xxx │ 24984.83.149.8 │
│ xxx │ 22797.217.118.83 │
│ xxx │ 22354.78.25.120 │
│ xxx │ 21285.213.87.131 │
│ xxx │ 20887.78.25.121 │
│ xxx │ 19694.188.162.65 │
│ xxx │ 17406.83.149.48 │
└──┴──┘└──┴──┘
```

Since using 'xxx' is highly unusual, this may be changed in the future. We recommend that you don't rely on the .exact format of this fragment

### (IPv6NumToString(x

Accepts a FixedString(16) value containing the IPv6 address in binary format. Returns a string containing this .address in text format

:IPv6-mapped IPv4 addresses are output in the format ::ffff:111.222.33.44. Examples

```
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:bbb::2	24695
2a02:2698:abcd:abcd:abcd:8888:5555	22408
2a02:6b8:0:fff::ff	16389
2a01:4f8:111:6666::2	16016
2a02:2168:888:222::1	15896
2a01:7e00::ffff:ffff:ffff:222	14774
2a02:8109:eee:ee:eeee:eeee:eeee:eeee	14443
2a02:810b:8888:888:8888:8888:8888	14345
2a02:6b8:0:444:4444:4444:4444	14279
2a01:7e00::ffff:ffff:ffff	13880

```
SELECT
,(IPv6NumToString(ClientIP6 AS k
count() AS c
FROM hits_all
()WHERE EventDate = today
GROUP BY k
ORDER BY c DESC
LIMIT 10
```

IPv6NumToString(ClientIP6)	c
ffff:94.26.111.111	747440::
ffff:37.143.222.4	529483::
ffff:5.166.111.99	317707::
ffff:46.38.11.77	263086::
ffff:79.105.111.111	186611::
ffff:93.92.111.88	176773::
ffff:84.53.111.33	158709::
ffff:217.118.11.22	154004::
ffff:217.118.11.33	148449::
ffff:217.118.11.44	148243::

## (IPv6StringToNum(s

.The reverse function of IPv6NumToString. If the IPv6 address has an invalid format, it returns a string of null bytes  
.HEX can be uppercase or lowercase

## (IPv4ToIPv6(x

Takes a **UInt32** number. Interprets it as an IPv4 address in **big endian**. Returns a **FixedString(16)** value containing the  
:IPv6 address in binary format. Examples

```
SELECT IPv6NumToString(IPv4ToIPv6(IPv4StringToNum('192.168.0.1'))) AS addr
```

addr
ffff:192.168.0.1::

## (cutIPv6(x, bitsToCutForIPv6, bitsToCutForIPv4

Accepts a **FixedString(16)** value containing the IPv6 address in binary format. Returns a string containing the  
:address of the specified number of bits removed in text format. For example

```
WITH
,IPv6StringToNum('2001:0DB8:AC10:FE01:FEED:BABE:CAFE:F00D') AS ipv6
IPv4ToIPv6(IPv4StringToNum('192.168.0.1')) AS ipv4
SELECT
,(cutIPv6(ipv6, 2, 0
(cutIPv6(ipv4, 0, 2
```

```
)-(cutIPv6(ipv6, 2, 0)-----cutIPv6(ipv4, 0, 2)-r
| db8:ac10:fe01:feed:babe:cafe:0 | ::ffff:192.168.0.0:2001 |
|-----L
```

## ,(IPv4CIDRtoIPv4Range(ipv4, cidr

Accepts an IPv4 and an UInt8 value containing the **CIDR**. Return a tuple with two IPv4 containing the lower range .and the higher range of the subnet

```
(SELECT IPv4CIDRtoIPv4Range(toIPv4('192.168.5.2'), 16
```

```
)-(IPv4CIDRtoIPv4Range(toIPv4('192.168.5.2'), 16)-r
| ('192.168.255.255','192.168.0.0') |
|-----L
```

## ,(IPv6CIDRtoIPv6Range(ipv6, cidr

Accepts an IPv6 and an UInt8 value containing the CIDR. Return a tuple with two IPv6 containing the lower range .and the higher range of the subnet

```
;(SELECT IPv6CIDRtoIPv6Range(toIPv6('2001:0db8:0000:85a3:0000:0000:ac1f:8001'), 32
```

```
)-(IPv6CIDRtoIPv6Range(toIPv6('2001:0db8:0000:85a3:0000:0000:ac1f:8001'), 32)-r
| ('db8::','2001:db8:ffff:ffff:ffff:ffff:ffff:2001') |
|-----L
```

## (toIPv4(string

An alias to **IPv4StringToNum()** that takes a string form of IPv4 address and returns value of **IPv4** type, which is binary .()equal to value returned by **IPv4StringToNum**

```
WITH
as IPv4_string '171.225.130.45'
SELECT
,(toTypeName(IPv4StringToNum(IPv4_string
((toTypeName(toIPv4(IPv4_string
```

```
)-((toTypeName(IPv4StringToNum(IPv4_string))-toTypeName(toIPv4(IPv4_string)-r
| UInt32 | IPv4 |
|-----L
```

```
WITH
as IPv4_string '171.225.130.45'
SELECT
,(hex(IPv4StringToNum(IPv4_string
((hex(toIPv4(IPv4_string
```

```
)-((hex(IPv4StringToNum(IPv4_string))-hex(toIPv4(IPv4_string)-r
| ABE1822D | ABE1822D |
|-----L
```

## (toIPv6(string

An alias to **IPv6StringToNum()** that takes a string form of IPv6 address and returns value of **IPv6** type, which is binary .()equal to value returned by **IPv6StringToNum**

```
--((toTypeName(IPv6StringToNum(IPv6_string)))--toTypeName(toIPv6(IPv6_string)-r
```

FixedString(16) | IPv6 |

---

```

|-----((hex(IPv6StringToNum(IPv6_string)))---hex(tolIPv6(IPv6_string)-r
| 20010438FFFF0000000000000407D1BC1 | 20010438FFFF0000000000000407D1BC1 |
|-----|-----|

```

# Functions for working with JSON

In Yandex.Metrica, JSON is transmitted by users as session parameters. There are some special functions for working with this JSON. (Although in most of the cases, the JSONs are additionally pre-processed, and the resulting values are put in separate columns in their processed format.) All these functions are based on strong assumptions about what the JSON can be, but they try to do as little as possible to get the job done

:The following assumptions are made

- .The field name (function argument) must be a constant .1
- The field name is somehow canonically encoded in JSON. For example: `visitParamHas({'"abc":"def"}', 'abc') = 1`,  
but `visitParamHas({'"\u0061\u0062\u0063":"def"}', 'abc') = 0` .2
- Fields are searched for on any nesting level, indiscriminately. If there are multiple matching fields, the first  
.occurrence is used .3
- .The JSON doesn't have space characters outside of string literals .4

```
(visitParamHas(params, name
```

.Checks whether there is a field with the 'name' name

```
(visitParamExtractUInt(params, name
```

Parses UInt64 from the value of the field named 'name'. If this is a string field, it tries to parse a number from the .beginning of the string. If the field doesn't exist, or it exists but doesn't contain a number, it returns 0

```
(visitParamExtractInt(params, name
```

.The same as for Int64

```
(visitParamExtractFloat(params, name
```

.The same as for Float64

```
(visitParamExtractBool(params, name
```

.Parses a true/false value. The result is UInt8

```
(visitParamExtractRaw(params, name
```

.Returns the value of a field, including separators

## :Examples

```
"visitParamExtractRaw('{ "abc": "\n\u0000"}', 'abc') = "\n\u0000"
'[visitParamExtractRaw('{ "abc": {"def": [1,2,3]} }', 'abc') = '{"def": [1,2,3
```

## (visitParamExtractString(params, name

.Parses the string in double quotes. The value is unescaped. If unescaping failed, it returns an empty string

:Examples

```
'visitParamExtractString('{ "abc": "\n\u0000"}', 'abc') = '\n\u0000'
'☺' = ('visitParamExtractString('{ "abc": "\u263a"}', 'abc')
'' = ('visitParamExtractString('{ "abc": "\u263"}', 'abc')
'' = ('visitParamExtractString('{ "abc": "hello"}', 'abc')
```

There is currently no support for code points in the format `\uXXXX\uYYYY` that are not from the basic multilingual .(plane (they are converted to CESU-8 instead of UTF-8

The following functions are based on [simdjson](#) designed for more complex JSON parsing requirements. The .assumption 2 mentioned above still applies

## (...[JSONHas(json[, indices\_or\_keys

.If the value exists in the JSON document, **1** will be returned

.If the value does not exist, **0** will be returned

:Examples

```
select JSONHas('{ "a": "hello", "b": [-100, 200.0, 300]}', 'b') = 1
select JSONHas('{ "a": "hello", "b": [-100, 200.0, 300]}', 'b', 4) = 0
```

.**indices\_or\_keys** is a list of zero or more arguments each of them can be either string or integer

.String = access object member by key

.Positive integer = access the n-th member/key from the beginning

.Negative integer = access the n-th member/key from the end

.Minimum index of the element is 1. Thus the element 0 doesn't exist

.You may use integers to access both JSON arrays and JSON objects

:So, for example

```
'select JSONExtractKey('{ "a": "hello", "b": [-100, 200.0, 300]}', 1) = 'a'
'select JSONExtractKey('{ "a": "hello", "b": [-100, 200.0, 300]}', 2) = 'b'
'select JSONExtractKey('{ "a": "hello", "b": [-100, 200.0, 300]}', -1) = 'b'
'select JSONExtractKey('{ "a": "hello", "b": [-100, 200.0, 300]}', -2) = 'a'
'select JSONExtractString('{ "a": "hello", "b": [-100, 200.0, 300]}', 1) = 'hello'
```

## (...[JSONLength(json[, indices\_or\_keys

.Return the length of a JSON array or a JSON object

.If the value does not exist or has a wrong type, **0** will be returned

:Examples

```
select JSONLength('{ "a": "hello", "b": [-100, 200.0, 300]}', 'b') = 3
select JSONLength('{ "a": "hello", "b": [-100, 200.0, 300]}') = 2
```

## (...[JSONType(json[, indices\_or\_keys

.Return the type of a JSON value

.If the value does not exist, **Null** will be returned

:Examples

```
'select JSONType('{ "a": "hello", "b": [-100, 200.0, 300] }') = 'Object'
'select JSONType('{ "a": "hello", "b": [-100, 200.0, 300] }', 'a') = 'String'
'select JSONType('{ "a": "hello", "b": [-100, 200.0, 300] }', 'b') = 'Array'
```

(...[JSONExtractUInt(json[, indices\_or\_keys

(...[JSONExtractInt(json[, indices\_or\_keys

(...[JSONExtractFloat(json[, indices\_or\_keys

(...[JSONExtractBool(json[, indices\_or\_keys

.Parses a JSON and extract a value. These functions are similar to [visitParam](#) functions

.If the value does not exist or has a wrong type, 0 will be returned

:Examples

```
select JSONExtractInt('{ "a": "hello", "b": [-100, 200.0, 300] }', 'b', 1) = -100
select JSONExtractFloat('{ "a": "hello", "b": [-100, 200.0, 300] }', 'b', 2) = 200.0
select JSONExtractUInt('{ "a": "hello", "b": [-100, 200.0, 300] }', 'b', -1) = 300
```

(...[JSONExtractString(json[, indices\_or\_keys

.Parses a JSON and extract a string. This function is similar to [visitParamExtractString](#) functions

.If the value does not exist or has a wrong type, an empty string will be returned

.The value is unescaped. If unescaping failed, it returns an empty string

:Examples

```
'select JSONExtractString('{ "a": "hello", "b": [-100, 200.0, 300] }', 'a') = 'hello'
'select JSONExtractString('{ "abc": "\n\u0000" }', 'abc') = '\n\u0000'
'☺' = ('select JSONExtractString('{ "abc": "\u263a" }', 'abc'
" = ('select JSONExtractString('{ "abc": "\u263" }', 'abc'
" = ('select JSONExtractString('{ "abc": "hello" }', 'abc'
```

(JSONExtract(json[, indices\_or\_keys...], return\_type

.Parses a JSON and extract a value of the given ClickHouse data type

.This is a generalization of the previous [JSONExtract<type>](#) functions

This means

.()JSONExtract(..., 'String') returns exactly the same as [JSONExtractString](#)

.()JSONExtract(..., 'Float64') returns exactly the same as [JSONExtractFloat](#)

:Examples

```
([SELECT JSONExtract('{ "a": "hello", "b": [-100, 200.0, 300] }', 'Tuple(String, Array(Float64))') = ('hello', [-100, 200, 300]
('SELECT JSONExtract('{ "a": "hello", "b": [-100, 200.0, 300] }', 'Tuple(b Array(Float64), a String)') = ([-100, 200, 300], 'hello'
[SELECT JSONExtract('{ "a": "hello", "b": [-100, 200.0, 300] }', 'b', 'Array(Nullable(Int8))') = [-100, NULL, NULL]
SELECT JSONExtract('{ "a": "hello", "b": [-100, 200.0, 300] }', 'b', 4, 'Nullable(Int64)') = NULL
SELECT JSONExtract('{ "passed": true }', 'passed', 'UInt8') = 1
SELECT JSONExtract('{ "day": "Thursday" }', 'day', 'Enum8(\ \'Sunday\' = 0, \ \'Monday\' = 1, \ \'Tuesday\' = 2, \ \'Wednesday\' = 3,
\ \'Thursday\' = 4, \ \'Friday\' = 5, \ \'Saturday\' = 6)') = 'Thursday'
SELECT JSONExtract('{ "day": 5 }', 'day', 'Enum8(\ \'Sunday\' = 0, \ \'Monday\' = 1, \ \'Tuesday\' = 2, \ \'Wednesday\' = 3, \ \'Thursday\' =
\ \'Friday\' = 5, \ \'Saturday\' = 6)') = 'Friday'
```

(JSONExtractKeysAndValues(json[, indices\_or\_keys...], value\_type

.Parse key-value pairs from a JSON where the values are of the given ClickHouse data type

:Example

```
[(SELECT JSONExtractKeysAndValues('{ "x": { "a": 5, "b": 7, "c": 11 } }', 'x', 'Int8') = [(('a',5),('b',7),('c',11
```

## (...[JSONExtractRaw(json[, indices\_or\_keys

.Returns a part of JSON

.If the part does not exist or has a wrong type, an empty string will be returned

:Example

```
[select JSONExtractRaw('{ "a": "hello", "b": [-100, 200.0, 300] }', 'b') = '[-100, 200.0, 300
```

## Higher-order functions

### operator, lambda(params, expr) function <-

Allows describing a lambda function for passing to a higher-order function. The left side of the arrow has a formal parameter, which is any ID, or multiple formal parameters – any IDs in a tuple. The right side of the arrow has an expression that can use these formal parameters, as well as any table columns

.Examples: `x -> 2 * x`, `str -> str != Referer`

.Higher-order functions can only accept lambda functions as their functional argument

A lambda function that accepts multiple arguments can be passed to a higher-order function. In this case, the higher-order function is passed several arrays of identical length that these arguments will correspond to

For some functions, such as `arrayCount` or `arraySum`, the first argument (the lambda function) can be omitted. In this case, identical mapping is assumed

:A lambda function can't be omitted for the following functions

`arrayMap`

`arrayFilter`

`arrayFirst`

`arrayFirstIndex`

- 
- 
- 
- 

## (... ,arrayMap(func, arr1

.Returns an array obtained from the original application of the `func` function to each element in the `arr` array

:Examples

```
;SELECT arrayMap(x -> (x + 2), [1, 2, 3]) as res
```

```
┌─── res ──┐
│ [3,4,5] │
└───┬───┘
```

:The following example shows how to create a tuple of elements from different arrays

```
SELECT arrayMap((x, y) -> (x, y), [1, 2, 3], [4, 5, 6]) AS res
```

```
┌─── res ──┐
│ [(3,6),(2,5),(1,4)] │
└───┬───┘
```

.Note that the first argument (lambda function) can't be omitted in the `arrayMap` function

## (... ,arrayFilter(func, arr1

.Returns an array containing only the elements in `arr1` for which `func` returns something other than 0

## :Examples

```
SELECT arrayFilter(x -> x LIKE '%World%', ['Hello', 'abc World']) AS res
```

```
┌──res─┐
│ ['abc World'] │
└───┘
```

```
SELECT
)arrayFilter
,'%i, x) -> x LIKE '%World)
,(arrayEnumerate(arr
('Hello', 'abc World'] AS arr']
AS res
```

```
┌──res─┐
│ [2] │
└───┘
```

.Note that the first argument (lambda function) can't be omitted in the [arrayFilter](#) function

## (... ,arrayCount([func,] arr1

Returns the number of elements in the arr array for which func returns something other than 0. If 'func' is not .specified, it returns the number of non-zero elements in the array

## (... ,arrayExists([func,] arr1

Returns 1 if there is at least one element in 'arr' for which 'func' returns something other than 0. Otherwise, it .returns 0

## (... ,arrayAll([func,] arr1

.Returns 1 if 'func' returns something other than 0 for all the elements in 'arr'. Otherwise, it returns 0

## (... ,arraySum([func,] arr1

.Returns the sum of the 'func' values. If the function is omitted, it just returns the sum of the array elements

## (... ,arrayFirst(func, arr1

.Returns the first element in the 'arr1' array for which 'func' returns something other than 0

.Note that the first argument (lambda function) can't be omitted in the [arrayFirst](#) function

## (... ,arrayFirstIndex(func, arr1

.Returns the index of the first element in the 'arr1' array for which 'func' returns something other than 0

.Note that the first argument (lambda function) can't be omitted in the [arrayFirstIndex](#) function

## (... ,arrayCumSum([func,] arr1

Returns an array of partial sums of elements in the source array (a running sum). If the [func](#) function is specified, .then the values of the array elements are converted by this function before summing

## :Example

```
SELECT arrayCumSum([1, 1, 1, 1]) AS res
```

```
┌──res─┐
│ [4 ,3 ,2 ,1] │
└───┘
```

## (arrayCumSumNonNegative(arr

Same as `arrayCumSum`, returns an array of partial sums of elements in the source array (a running sum). Different `arrayCumSum`, when then returned value contains a value less than zero, the value is replace with zero and the :subsequent calculation is performed with zero parameters. For example

```
SELECT arrayCumSumNonNegative([1, 1, -4, 1]) AS res
```

```
┌─── res ──┐
│ [1,2,0,1] │
└─────────┘
```

## `(... ,arraySort([func,] arr1`

Returns an array as result of sorting the elements of `arr1` in ascending order. If the `func` function is specified, (sorting order is determined by the result of the function `func` applied to the elements of array (arrays

.The **Schwartzian transform** is used to improve sorting efficiency

:Example

```
;(SELECT arraySort((x, y) -> y, ['hello', 'world'], [2, 1
```

```
┌─── res ──┐
│ ['world', 'hello'] │
└─────────┘
```

.For more information about the `arraySort` method, see the **Functions for Working With Arrays** section

## `(... ,arrayReverseSort([func,] arr1`

Returns an array as result of sorting the elements of `arr1` in descending order. If the `func` function is specified, (sorting order is determined by the result of the function `func` applied to the elements of array (arrays

:Example

```
;(SELECT arrayReverseSort((x, y) -> y, ['hello', 'world'], [2, 1]) as res
```

```
┌─── res ──┐
│ ['hello', 'world'] │
└─────────┘
```

.For more information about the `arrayReverseSort` method, see the **Functions for Working With Arrays** section

# Functions for working with external dictionaries

.For information on connecting and configuring external dictionaries, see **External dictionaries**

`dictGetUInt8, dictGetUInt16, dictGetUInt32, dictGetUInt64`

`dictGetInt8, dictGetInt16, dictGetInt32, dictGetInt64`

`dictGetFloat32, dictGetFloat64`

`dictGetDate, dictGetDateTime`

`dictGetUUID`

`dictGetString`

`(dictGetT('dict_name', 'attr_name', id`

Get the value of the `attr_name` attribute from the `dict_name` dictionary using the 'id' key.`dict_name` and `.attr_name` are constant strings.`id` must be `UInt64`

.If there is no `id` key in the dictionary, it returns the default value specified in the dictionary description

`dictGetTOrDefault`



`(dictGetOrDefault('dict_name', 'attr_name', id, default`

.The same as the `dictGetT` functions, but the default value is taken from the function's last argument

## dictIsIn

`(dictIsIn ('dict_name', child_id, ancestor_id`

For the 'dict\_name' hierarchical dictionary, finds out whether the 'child\_id' key is located inside 'ancestor\_id' •  
(or matches 'ancestor\_id'). Returns UInt8

## dictGetHierarchy

`(dictGetHierarchy('dict_name', id`

For the 'dict\_name' hierarchical dictionary, returns an array of dictionary keys starting from 'id' and •  
(continuing along the chain of parent elements. Returns Array(UInt64

## dictHas

`(dictHas('dict_name', id`

Check whether the dictionary has the key. Returns a UInt8 value equal to 0 if there is no key and 1 if there is •  
.a key

# Functions for working with Yandex.Metrica dictionaries

In order for the functions below to work, the server config must specify the paths and addresses for getting all the Yandex.Metrica dictionaries. The dictionaries are loaded at the first call of any of these functions. If the reference lists can't be loaded, an exception is thrown

."For information about creating reference lists, see the section "Dictionaries

## Multiple geobases

ClickHouse supports working with multiple alternative geobases (regional hierarchies) simultaneously, in order to support various perspectives on which countries certain regions belong to

The 'clickhouse-server' config specifies the file with the regional

`<hierarchy::<path_to_regions_hierarchy_file>/opt/geo/regions_hierarchy.txt</path_to_regions_hierarchy_file`

Besides this file, it also searches for files nearby that have the \_ symbol and any suffix appended to the name •  
(before the file extension

.For example, it will also find the file `/opt/geo/regions_hierarchy_ua.txt`, if present

.ua is called the dictionary key. For a dictionary without a suffix, the key is an empty string

All the dictionaries are re-loaded in runtime (once every certain number of seconds, as defined in the builtin\_dictionaries\_reload\_interval config parameter, or once an hour by default). However, the list of available dictionaries is defined one time, when the server starts

All functions for working with regions have an optional argument at the end – the dictionary key. It is referred to as the geobase

:Example

regionToCountry(RegionID) – Uses the default dictionary: /opt/geo/regions_hierarchy.txt
regionToCountry(RegionID, '') – Uses the default dictionary: /opt/geo/regions_hierarchy.txt
regionToCountry(RegionID, 'ua') – Uses the dictionary for the 'ua' key: /opt/geo/regions_hierarchy_ua.txt

## ([regionToCity(id[, geobase

Accepts a UInt32 number – the region ID from the Yandex geobase. If this region is a city or part of a city, it returns the region ID for the appropriate city. Otherwise, returns 0

## ([regionToArea(id[, geobase

Converts a region to an area (type 5 in the geobase). In every other way, this function is the same as .regionToCity

```
((SELECT DISTINCT regionToName(regionToArea(toUInt32(number), 'ua-
FROM system.numbers
LIMIT 15
```

--('regionToName(regionToArea(toUInt32(number), \'ua-r
Moscow and Moscow region
St. Petersburg and Leningrad region
Belgorod region
Ivanovsk region
Kaluga region
Kostroma region
Kursk region
Lipetsk region
Orlov region
Ryazan region
Smolensk region
Tambov region
Tver region
Tula region
_____L

## ([regionToDistrict(id[, geobase

Converts a region to a federal district (type 4 in the geobase). In every other way, this function is the same as .regionToCity

```
((SELECT DISTINCT regionToName(regionToDistrict(toUInt32(number), 'ua-
FROM system.numbers
LIMIT 15
```

--('regionToName(regionToDistrict(toUInt32(number), \'ua-r
Central federal district
Northwest federal district
South federal district
North Caucasus federal district
Privolga federal district
Ural federal district
Siberian federal district
Far East federal district
Scotland
Faroe Islands
Flemish region
Brussels capital region
Wallonia
Federation of Bosnia and Herzegovina
_____L

## ([regionToCountry(id[, geobase

.Converts a region to a country. In every other way, this function is the same as 'regionToCity  
(Example: regionToCountry(toUInt32(213)) = 225 converts Moscow (213) to Russia (225

## ([regionToContinent(id[, geobase

.Converts a region to a continent. In every other way, this function is the same as 'regionToCity  
(Example: regionToContinent(toUInt32(213)) = 10001 converts Moscow (213) to Eurasia (10001

## ([regionToPopulation(id[, geobase

.Gets the population for a region

."The population can be recorded in files with the geobase. See the section "External dictionaries"

.If the population is not recorded for the region, it returns 0

.In the Yandex geobase, the population might be recorded for child regions, but not for parent regions

## `([regionIn(lhs, rhs[, geobase`

Checks whether a 'lhs' region belongs to a 'rhs' region. Returns a UInt8 number equal to 1 if it belongs, or 0 if it doesn't belong

.The relationship is reflexive – any region also belongs to itself

## `([regionHierarchy(id[, geobase`

Accepts a UInt32 number – the region ID from the Yandex geobase. Returns an array of region IDs consisting of the passed region and all parents along the chain

[Example: `regionHierarchy(toUInt32(213)) = [213,1,3,225,10001,10000`

## `([regionToName(id[, lang`

Accepts a UInt32 number – the region ID from the Yandex geobase. A string with the name of the language can be passed as a second argument. Supported languages are: ru, en, ua, uk, by, kz, tr. If the second argument is omitted, the language 'ru' is used. If the language is not supported, an exception is thrown. Returns a string – the name of the region in the corresponding language. If the region with the specified ID doesn't exist, an empty string is returned

.ua and uk both mean Ukrainian

# Functions for implementing the IN operator

## `in, notIn, globalIn, globalNotIn`

.See the section [IN operators](#)

## `(... ,tuple(x, y, ...), operator (x, y`

.A function that allows grouping multiple columns

For columns with the types T1, T2, ..., it returns a Tuple(T1, T2, ...) type tuple containing these columns. There is no cost to execute the function

Tuples are normally used as intermediate values for an argument of IN operators, or for creating a list of formal parameters of lambda functions. Tuples can't be written to a table

## `tupleElement(tuple, n), operator x.N`

.A function that allows getting a column from a tuple

N' is the column index, starting from 1. N must be a constant. 'N' must be a constant. 'N' must be a strict positive integer no greater than the size of the tuple

.There is no cost to execute the function

# arrayJoin function

.This is a very unusual function

.(Normal functions don't change a set of rows, but just change the values in each row (map

.(Aggregate functions compress a set of rows (fold or reduce

.(The 'arrayJoin' function takes each row and generates a set of rows (unfold

This function takes an array as an argument, and propagates the source row to multiple rows for the number of elements in the array

All the values in columns are simply copied, except the values in the column where this function is applied; it is replaced with the corresponding array value

.A query can use multiple `arrayJoin` functions. In this case, the transformation is performed multiple times

.Note the ARRAY JOIN syntax in the SELECT query, which provides broader possibilities

:Example

```
SELECT arrayJoin([1, 2, 3] AS src) AS dst, 'Hello', src
```

dst	'Hello'	src
[Hello	[1,2,3	1
[Hello	[1,2,3	2
[Hello	[1,2,3	3

# Functions for working with geographical coordinates

## greatCircleDistance

.Calculate the distance between two points on the Earth's surface using the great-circle formula

```
(greatCircleDistance(lon1Deg, lat1Deg, lon2Deg, lat2Deg
```

### Input parameters

- .lon1Deg — Longitude of the first point in degrees. Range: [-180°, 180°
- .lat1Deg — Latitude of the first point in degrees. Range: [-90°, 90°
- .lon2Deg — Longitude of the second point in degrees. Range: [-180°, 180°
- .lat2Deg — Latitude of the second point in degrees. Range: [-90°, 90°

Positive values correspond to North latitude and East longitude, and negative values correspond to South latitude and West longitude

### Returned value

- .The distance between two points on the Earth's surface, in meters
- .Generates an exception when the input parameter values fall outside of the range

### Example

```
(SELECT greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673
```

(greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673
14132374.194975413

## pointInEllipses

.Checks whether the point belongs to at least one of the ellipses

```
(pointInEllipses(x, y, x0, y0, a0, b0,...,xn, yn, an, bn
```

### Input parameters

- .x, y — Coordinates of a point on the plane
- .xi, yi — Coordinates of the center of the i-th ellipsis
- .ai, bi — Axes of the i-th ellipsis in meters

.The input parameters must be 2+4·n, where n is the number of ellipses

### Returned values

.if the point is inside at least one of the ellipses; 0 if it is not 1

### Example

```
(SELECT pointInEllipses(55.755831, 37.617673, 55.755831, 37.617673, 1.0, 2.0
```

```

┌--(.pointInEllipses(55.755831, 37.617673, 55.755831, 37.617673, 1., 2--r
| 1 |
└-----┘

```

## pointInPolygon

.Checks whether the point belongs to the polygon on the plane

```

(... ,[... (pointInPolygon((x, y), [(a, b), (c, d

```

### Input values

- **.x, y)** — Coordinates of a point on the plane. Data type — **Tuple** — A tuple of two numbers)
- **a, b), (c, d) ...]** — Polygon vertices. Data type — **Array**. Each vertex is represented by a pair of coordinates **(a,)]**
- **b).** Vertices should be specified in a clockwise or counterclockwise order. The minimum number of vertices is **.3**. The polygon must be constant
- The function also supports polygons with holes (cut out sections). In this case, add polygons that define the cut out sections using additional arguments of the function. The function does not support non-simply-connected polygons

### Returned values

- .if the point is inside the polygon, **0** if it is not **1**
- .If the point is on the polygon boundary, the function may return either 0 or 1

### Example

```

SELECT pointInPolygon((3., 3.), [(6, 0), (8, 4), (5, 8), (0, 2)]) AS res

```

```

┌--res--r
| 1 |
└-----┘

```

## geohashEncode

Encodes latitude and longitude as a geohash-string, please see (<http://geohash.org/>, <https://en.wikipedia.org/wiki/Geohash>)

```

([geohashEncode(longitude, latitude, [precision

```

### Input values

- **[longitude** - longitude part of the coordinate you want to encode. Floating in range **[-180°, 180°**
- **[latitude** - latitude part of the coordinate you want to encode. Floating in range **[-90°, 90°**
- **precision** - Optional, length of the resulting encoded string, defaults to **12**. Integer in range **[1, 12]**. Any value less than **1** or greater than **12** is silently converted to **12**

### Returned values

- .(alphanumeric **String** of encoded coordinate (modified version of the base32-encoding alphabet is used)

### Example

```

SELECT geohashEncode(-5.60302734375, 42.593994140625, 0) AS res

```

```

┌-----res--r
| ezs42d000000 |
└-----┘

```

## geohashDecode

.Decodes any geohash-encoded string into longitude and latitude

### Input values

.encoded string - geohash-encoded string

Returned values

.longitude, latitude) - 2-tuple of Float64 values of longitude and latitude)

Example

```
SELECT geohashDecode('ezs42') AS res
```

res
(5.60302734375,42.60498046875-)

# Functions for working with Nullable aggregates

## isNull

.Checks whether the argument is NULL

```
(isNull(x
```

Parameters

.x — A value with a non-compound data type

Returned value

- .if x is NULL 1
- .if x is not NULL 0

Example

Input table

x	y
NULL	1
3	2

Query

```
(SELECT x FROM t_null WHERE isNull(y (:
```

```
SELECT x
FROM t_null
(WHERE isNull(y
```

x
1

.rows in set. Elapsed: 0.010 sec 1

## isNotNull

.Checks whether the argument is NULL

```
(isNotNull(x
```

Parameters

.x — A value with a non-compound data type

Returned value

- .if **x** is **NULL 0**
- .if **x** is not **NULL 1**

### Example

Input table

x	y
NULL	1
3	2

Query

```
(SELECT x FROM t_null WHERE isNotNull(y (:

SELECT x
FROM t_null
(WHERE isNotNull(y

x
| 2 |
| 2 |

.rows in set. Elapsed: 0.010 sec 1
```

## coalesce

.Checks from left to right whether **NULL** arguments were passed and returns the first non-**NULL** argument

```
(...,coalesce(x
```

### :Parameters

.Any number of parameters of a non-compound type. All parameters must be compatible by data type

### Returned values

- .The first non-**NULL** argument
- .**NULL**, if all arguments are **NULL**

### Example

.Consider a list of contacts that may specify multiple ways to contact a customer

name	mail	phone	icq
client 1	NULL	123-45-67	123
client 2	NULL	NULL	NULL

.The **mail** and **phone** fields are of type String, but the **icq** field is **UInt32**, so it needs to be converted to **String**

:Get the first available contact method for the customer from the contact list

```
SELECT coalesce(mail, phone, CAST(icq,'Nullable(String)')) FROM aBook (:

(('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 |

.rows in set. Elapsed: 0.006 sec 2
```

# ifNull

.Returns an alternative value if the main argument is **NULL**

```
(ifNull(x,alt
```

## :Parameters

- .**x** — The value to check for **NULL**
- .**alt** — The value that the function returns if **x** is **NULL**

## Returned values

- .The value **x**, if **x** is not **NULL**
- .The value **alt**, if **x** is **NULL**

## Example

```
(SELECT ifNull('a', 'b')
FROM (SELECT 'a' AS a)
LIMIT 1
```

```
(SELECT ifNull(NULL, 'b')
FROM (SELECT NULL AS b)
LIMIT 1
```

# nullIf

.Returns **NULL** if the arguments are equal

```
(nullIf(x, y
```

## :Parameters

.**x**, **y** — Values for comparison. They must be compatible types, or ClickHouse will generate an exception

## Returned values

- .**NULL**, if the arguments are equal
- .The **x** value, if the arguments are not equal

## Example

```
(SELECT nullIf(1, 1)
FROM (SELECT 1)
LIMIT 1
```

```
(SELECT nullIf(1, 2)
FROM (SELECT 1)
LIMIT 1
```

# assumeNotNull

.Results in a value of type **Nullable** for a non- **Nullable**, if the value is not **NULL**

```
(assumeNotNull(x
```

## :Parameters



.x — The original value

Returned values

- .The original value from the non-Nullable type, if it is not NULL
- .The default value for the non-Nullable type if the original value was NULL

Example

.Consider the t\_null table

```
SHOW CREATE TABLE t_null
--statement--
CREATE TABLE default.t_null ( x Int8, y Nullable(Int8)) ENGINE = TinyLog |
--L
```

x	y
NULL	1
3	2

.Apply the resumenotnull function to the y column

```
SELECT assumeNotNull(y) FROM t_null
--(assumeNotNull(y)
0
3
--L
```

```
SELECT toTypeName(assumeNotNull(y)) FROM t_null
--((toTypeName(assumeNotNull(y)
Int8
Int8
--L
```

toNullable

.Converts the argument type to Nullable

```
(toNullable(x
```

:Parameters

.x — The value of any non-compound type

Returned value

.The input value with a Nullable type

Example

```
(SELECT toTypeName(10
--(toTypeName(10
UInt8
--L

((SELECT toTypeName(toNullable(10
--((toTypeName(toNullable(10
(Nullable(UInt8
--L
```

# Machine learning methods

## (evalMLMethod (prediction

.Prediction using fitted regression models uses [evalMLMethod](#) function. See link in [linearRegression](#)

## Stochastic Linear Regression

The [stochasticLinearRegression](#) aggregate function implements stochastic gradient descent method using linear .model and MSE loss function. Uses [evalMLMethod](#) to predict on new data

## Stochastic Logistic Regression

The [stochasticLogisticRegression](#) aggregate function implements stochastic gradient descent method for binary .classification problem. Uses [evalMLMethod](#) to predict on new data

## Other functions

### ()hostName

Returns a string with the name of the host that this function was performed on. For distributed processing, this is .the name of the remote server host, if the function is performed on a remote server

### basename

Extracts trailing part of a string after the last slash or backslash. This function if often used to extract the filename .from the path

```
( basename( expr
```

#### Parameters

[expr](#) — Expression, resulting in the [String](#)-type value. All the backslashes must be escaped in the resulting •  
.value

#### Returned Value

:A String-type value that contains

.Trailing part of a string after the last slash or backslash in it •

If the input string contains a path, ending with slash or backslash, for example, / or c:\, the function returns an .empty string

.Original string if there are no slashes or backslashes in it •

#### Example

```
(SELECT 'some/long/path/to/file' AS a, basename(a
```

```
|('a | basename('some\\long\\path\\to\\file- |  
| some\\long\\path\\to\\file | file |  
|_____|  
|_____L
```

```
(SELECT 'some\\long\\path\\to\\file' AS a, basename(a
```

```
|('a | basename('some\\long\\path\\to\\file- |  
| some\\long\\path\\to\\file | file |  
|_____|  
|_____L
```

```
(SELECT 'some-file-name' AS a, basename(a
```

```
|('a | basename('some-file-name- |  
| some-file-name | some-file-name |  
|_____|  
|_____L
```

```
(visibleWidth(x
```

.Calculates the approximate width when outputting values to the console in text format (tab-separated  
 .This function is used by the system for implementing Pretty formats

.NULL is represented as a string corresponding to NULL in Pretty formats

```
(SELECT visibleWidth(NULL
- (visibleWidth(NULL-r
4          |
|_____L
```

```
(typeName(x
```

.Returns a string containing the type name of the passed argument

If **NULL** is passed to the function as input, then it returns the **Nullable(Nothing)** type, which corresponds to an internal **.NULL** representation in ClickHouse

()blockSize

.Gets the size of the block

In ClickHouse, queries are always run on blocks (sets of column parts). This function allows getting the size of the .block that you called it for

```
(materialize(x
```

- Turns a constant into a full column containing just one value

In ClickHouse, full columns and constants are represented differently in memory. Functions work differently for constant arguments and normal arguments (different code is executed), although the result is almost always the same. This function is for debugging this behavior

(...)ignore

.Accepts any arguments, including **NULL**. Always returns 0

.However, the argument is still evaluated. This can be used for benchmarks

```
(sleep(seconds
```

.Sleeps 'seconds' seconds on each data block. You can specify an integer or a floating-point number

```
(sleepEachRow(seconds
```

.Sleeps 'seconds' seconds on each row. You can specify an integer or a floating-point number

`()currentDatabase`

.Returns the name of the current database

You can use this function in table engine parameters in a CREATE TABLE query where you need to specify the .database

```
(isFinite(x
```

Accepts Float32 and Float64 and returns UInt8 equal to 1 if the argument is not infinite and not a NaN, otherwise .0

```
(isInfinite(x
```

Accepts Float32 and Float64 and returns UInt8 equal to 1 if the argument is infinite, otherwise 0. Note that 0 is returned for a NaN

```
(isNaN(x
```

.Accepts Float32 and Float64 and returns UInt8 equal to 1 if the argument is a NaN, otherwise 0

# hasColumnInTable(['hostname'[, 'username'[, 'password']], ('database', 'table', 'column

Accepts constant strings: database name, table name, and column name. Returns a UInt8 constant expression .equal to 1 if there is a column, otherwise 0. If the hostname parameter is set, the test will run on a remote server .The function throws an exception if the table does not exist For elements in a nested data structure, the function checks for the existence of a column. For the nested data .structure itself, the function returns 0

## bar

.Allows building a unicode-art diagram

.bar(x, min, max, width) draws a band with a width proportional to (x - min) and equal to width characters when x = max

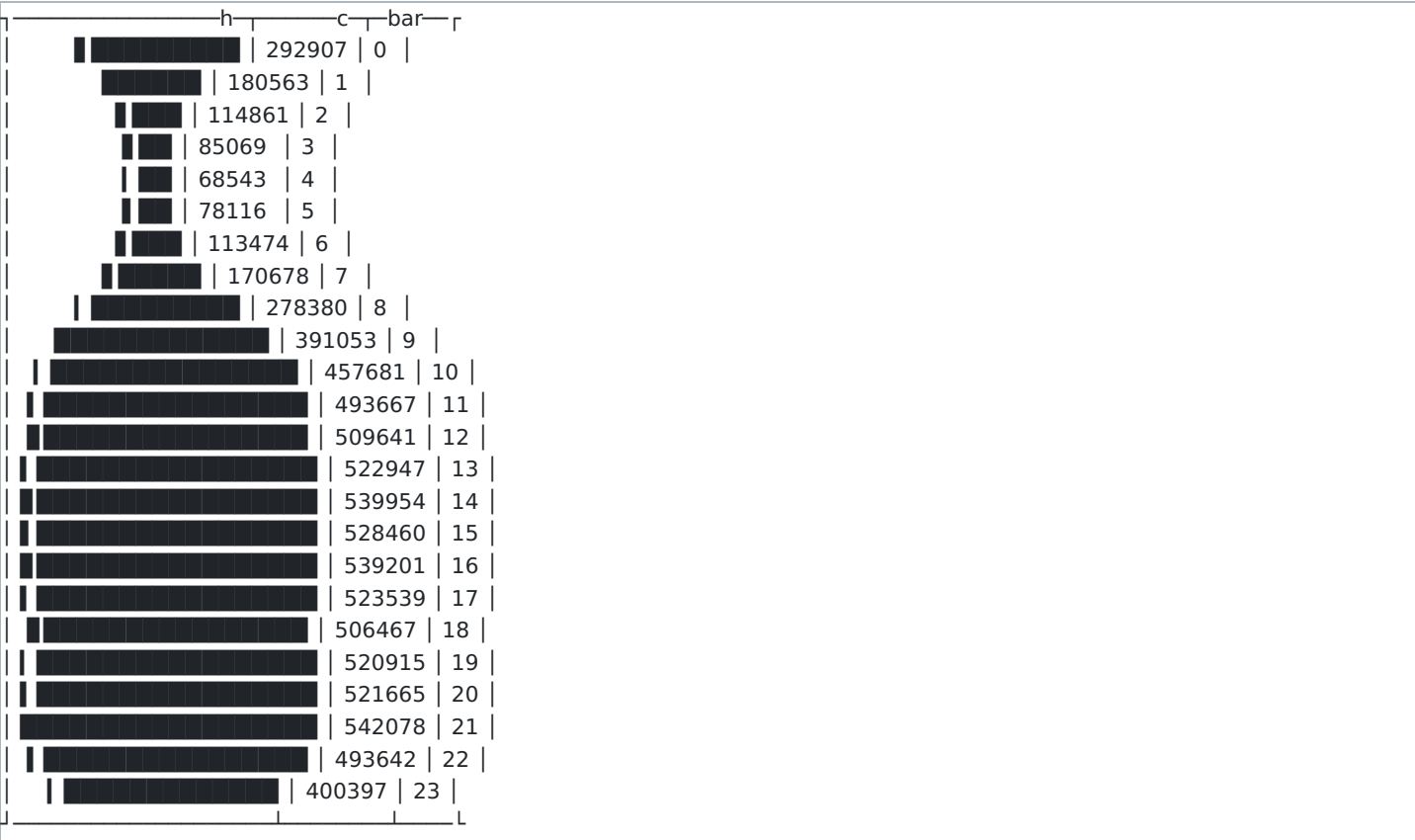
:Parameters

- .x — Size to display
- .min, max — Integer constants. The value must fit in Int64
- .width — Constant, positive integer, can be fractional

.The band is drawn with accuracy to one eighth of a symbol

:Example

```
SELECT
,toHour(EventTime) AS h
,count() AS c
,bar(c, 0, 600000, 20) AS bar
FROM test.hits
GROUP BY h
ORDER BY h ASC
```



## transform

.Transforms a value according to the explicitly defined mapping of some elements to other ones  
:There are two variations of this function

`(transform(x, array_from, array_to, default`

.1

.`x` – What to transform

.`array_from` – Constant array of values for converting

.`array_to` – Constant array of values to convert the values in 'from' to

.`'default'` – Which value to use if 'x' is not equal to any of the values in 'from'

.`array_from` and `array_to` – Arrays of the same size

:Types

`transform(T, Array(T), Array(U), U) -> U`

.`T` and `U` can be numeric, string, or Date or DateTime types

Where the same letter is indicated (T or U), for numeric types these might not be matching types, but types that  
.have a common type

.For example, the first argument can have the Int64 type, while the second has the Array(UInt16) type

If the 'x' value is equal to one of the elements in the 'array\_from' array, it returns the existing element (that is  
numbered the same) from the 'array\_to' array. Otherwise, it returns 'default'. If there are multiple matching  
elements in 'array\_from', it returns one of the matches

:Example

```
SELECT
,transform(SearchEngineID, [2, 3], ['Yandex', 'Google'], 'Other') AS title
count() AS c
FROM test.hits
WHERE SearchEngineID != 0
GROUP BY title
ORDER BY c DESC
```

title	c
Yandex	498635
Google	229872
Other	104472

`(transform(x, array_from, array_to`

.2

.Differs from the first variation in that the 'default' argument is omitted

If the 'x' value is equal to one of the elements in the 'array\_from' array, it returns the matching element (that is  
.'numbered the same) from the 'array\_to' array. Otherwise, it returns 'x'

:Types

`transform(T, Array(T), Array(T)) -> T`

:Example

```
SELECT
,transform(domain(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	r
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

.Accepts the size (number of bytes). Returns a rounded size with a suffix (KiB, MiB, etc.) as a string

:Example

```
SELECT
,arrayJoin([1, 1024, 1024*1024, 192851925]) AS filesize_bytes
formatReadableSize(filesize_bytes) AS filesize
```

filesize_bytes	filesize
B 1.00	1
KiB 1.00	1024
MiB 1.00	1048576
MiB 183.92	192851925

## (least(a, b

.Returns the smallest value from a and b

## (greatest(a, b

.Returns the largest value of a and b

## ()uptime

.Returns the server's uptime in seconds

## ()version

.Returns the version of the server as a string

## ()timezone

.Returns the timezone of the server

## blockNumber

.Returns the sequence number of the data block where the row is located

## rowNumberInBlock

.Returns the ordinal number of the row in the data block. Different data blocks are always recalculated

## ()rowNumberInAllBlocks

.Returns the ordinal number of the row in the data block. This function only considers the affected data blocks

## (runningDifference(x

.Calculates the difference between successive row values in the data block

.Returns 0 for the first row and the difference from the previous row for each subsequent row

.The result of the function depends on the affected data blocks and the order of data in the block  
If you make a subquery with ORDER BY and call the function from outside the subquery, you can get the expected .result

:Example

```
SELECT
,EventID
,EventTime
runningDifference(EventTime) AS delta
FROM
)
SELECT
,EventID
EventTime
FROM events
'WHERE EventDate = '2016-11-24
ORDER BY EventTime ASC
LIMIT 5
(
```

EventID	EventTime	delta
0	00:00:04 2016-11-24	1106
1	00:00:05 2016-11-24	1107
0	00:00:05 2016-11-24	1108
4	00:00:09 2016-11-24	1109
1	00:00:10 2016-11-24	1110

## runningDifferenceStartingWithFirstValue

Same as for **runningDifference**, the difference is the value of the first row, returned the value of the first row, and .each subsequent row returns the difference from the previous row

## (MACNumToString(num

Accepts a UInt64 number. Interprets it as a MAC address in big endian. Returns a string containing the .(corresponding MAC address in the format AA:BB:CC:DD:EE:FF (colon-separated numbers in hexadecimal form

## (MACStringToNum(s

.The inverse function of MACNumToString. If the MAC address has an invalid format, it returns 0

## (MACStringToOUI(s

Accepts a MAC address in the format AA:BB:CC:DD:EE:FF (colon-separated numbers in hexadecimal form). Returns .the first three octets as a UInt64 number. If the MAC address has an invalid format, it returns 0

## getsizeofEnumType

.Returns the number of fields in **Enum**

```
(getsizeofEnumType(value
```

### :Parameters

.**value** — Value of type **Enum** •

### Returned values

- .The number of fields with **Enum** input values •
- .An exception is thrown if the type is not **Enum** •

### Example

```
SELECT getSizeOfEnumType( CAST('a' AS Enum8('a' = 1, 'b' = 2) ) ) AS x
```

```
┌─X─┐
│ 2 │
└───┘
```

## toColumnName

.Returns the name of the class that represents the data type of the column in RAM

```
(toColumnName(value
```

### :Parameters

.**value** — Any type of value •

### Returned values

.A string with the name of the class that is used for representing the **value** data type in RAM •

### Example of the difference between **toTypeName** ' and ' **toColumnName**

```
((select toTypeName(cast('2018-01-01 01:02:03' AS DateTime (:
('SELECT toTypeName(CAST('2018-01-01 01:02:03', 'DateTime
┌─('toTypeName(CAST('2018-01-01 01:02:03', 'DateTime─┐
│                               DateTime │
└────────────────────────────────────────┘

.rows in set. Elapsed: 0.008 sec 1

((select toColumnName(cast('2018-01-01 01:02:03' AS DateTime (:
('SELECT toColumnName(CAST('2018-01-01 01:02:03', 'DateTime
┌─('toColumnName(CAST('2018-01-01 01:02:03', 'DateTime─┐
│                               (Const(UInt32 │
└────────────────────────────────────────┘
```

.(The example shows that the **DateTime** data type is stored in memory as **Const(UInt32**

## dumpColumnStructure

Outputs a detailed description of data structures in RAM

```
(dumpColumnStructure(value
```

### :Parameters

.**value** — Any type of value •

### Returned values

.A string describing the structure that is used for representing the **value** data type in RAM •

### Example

```
('SELECT dumpColumnStructure(CAST('2018-01-01 01:02:03', 'DateTime
┌─('dumpColumnStructure(CAST('2018-01-01 01:02:03', 'DateTime─┐
│                               ((DateTime, Const(size = 1, UInt32(size = 1 │
└────────────────────────────────────────┘
```

## defaultValueOfArgumentType



- .Outputs the default value for the data type
- .Does not include default values for custom columns set by the user

```
(defaultValueOfArgumentType(expression
```

**:Parameters**

- .**expression** — Arbitrary type of value or an expression that results in a value of an arbitrary type

**Returned values**

- .for numbers **0**
- .Empty string for strings
- .**NULL** for **Nullable**

**Example**

```
( (SELECT defaultValueOfArgumentType( CAST(1 AS Int8 (:
('SELECT defaultValueOfArgumentType(CAST(1, 'Int8
| 0 |
|-----|
rows in set. Elapsed: 0.002 sec 1

( ( (SELECT defaultValueOfArgumentType( CAST(1 AS Nullable(Int8 (:
('SELECT defaultValueOfArgumentType(CAST(1, 'Nullable(Int8
| NULL |
|-----|
rows in set. Elapsed: 0.002 sec 1
```

# indexHint

- .Outputs data in the range selected by the index without filtering by the expression specified as an argument
- The expression passed to the function is not calculated, but ClickHouse applies the index to this expression in the same way as if the expression was in the query without **indexHint**

**Returned value**

- .1

**Example**

- .Here is a table with the test data for **ontime**

```
SELECT count() FROM ontime
| 4276457 |
|-----|
```

- .((The table has indexes for the fields (**FlightDate**, (**Year**, **FlightDate**
- :Create a selection by date like this

```
SELECT FlightDate AS k, count() FROM ontime GROUP BY k ORDER BY k (:
```

```
SELECT
,FlightDate AS k
()count
FROM ontime
GROUP BY k
ORDER BY k ASC
```

```
┌──()k──┴──count──┐
| 13970 | 2017-01-01 |
| 15882 | 2017-01-02 |
.....
| 16411 | 2017-09-28 |
| 16384 | 2017-09-29 |
| 12520 | 2017-09-30 |
└────────┴────────┘
```

(.rows in set. Elapsed: 0.072 sec. Processed 4.28 million rows, 8.55 MB (59.00 million rows/s., 118.01 MB/s 273

In this selection, the index is not used and ClickHouse processed the entire table (Processed 4.28 million rows). To :apply the index, select a specific date and run the following query

```
SELECT FlightDate AS k, count() FROM ontime WHERE k = '2017-09-15' GROUP BY k ORDER BY k (:
```

```
SELECT
,FlightDate AS k
()count
FROM ontime
'WHERE k = '2017-09-15
GROUP BY k
ORDER BY k ASC
```

```
┌──()k──┴──count──┐
| 16428 | 2017-09-15 |
└────────┴────────┘
```

(.rows in set. Elapsed: 0.014 sec. Processed 32.74 thousand rows, 65.49 KB (2.31 million rows/s., 4.63 MB/s 1

The last line of output shows that by using the index, ClickHouse processed a significantly smaller number of rows .((Processed 32.74 thousand rows

:Now pass the expression k = '2017-09-15' to the indexHint function

```
SELECT FlightDate AS k, count() FROM ontime WHERE indexHint(k = '2017-09-15') GROUP BY k ORDER BY k (:
```

```
SELECT
,FlightDate AS k
()count
FROM ontime
('WHERE indexHint(k = '2017-09-15
GROUP BY k
ORDER BY k ASC
```

```
┌──()k──┴──count──┐
| 7071 | 2017-09-14 |
| 16428 | 2017-09-15 |
| 1077 | 2017-09-16 |
| 8167 | 2017-09-30 |
└────────┴────────┘
```

(.rows in set. Elapsed: 0.004 sec. Processed 32.74 thousand rows, 65.49 KB (8.97 million rows/s., 17.94 MB/s 4

The response to the request shows that ClickHouse applied the index in the same way as the previous time (Processed 32.74 thousand rows). However, the resulting set of rows shows that the expression `k = '2017-09-15'` was not used when generating the result

Because the index is sparse in ClickHouse, "extra" data ends up in the response when reading a range (in this case, the adjacent dates). Use the `indexHint` function to see it

## replicate

.Creates an array with a single value

.Used for internal implementation of `arrayJoin`

```
(replicate(x, arr
```

### :Parameters

`arr` — Original array. ClickHouse creates a new array of the same length as the original and fills it with the value `x`

`x` — The value that the resulting array will be filled with

### Output value

.An array filled with the value `x`

### Example

```
(['SELECT replicate(1, ['a', 'b', 'c
--(['replicate(1, ['a', 'b', 'c--r
|           [1,1,1] |
|_____L
```

## filesystemAvailable

Returns the remaining space information of the disk, in bytes. This information is evaluated using the configured by path

## filesystemCapacity

.Returns the capacity information of the disk, in bytes. This information is evaluated using the configured by path

## finalizeAggregation

.(Takes state of aggregate function. Returns result of aggregation (finalized state

## runningAccumulate

Takes the states of the aggregate function and returns a column with values, are the result of the accumulation of these states for a set of block lines, from the first to the current line

For example, takes state of aggregate function (example `runningAccumulate(uniqState(UserID)))`, and for each row of block, return result of aggregate function on merge of states of all previous rows and current row. So, result of function depends on partition of data to blocks and on order of data in block

## (joinGet('join\_storage\_table\_name', 'get\_column', join\_key

.Get data from a table of type Join using the specified join key

## (... ,modelEvaluate(model\_name

.Evaluate external model

.Accepts a model name and model arguments. Returns Float64

## (throwIf(x

.Throw an exception if the argument is non zero

## Aggregate functions

.Aggregate functions work in the **normal** way as expected by database experts

:ClickHouse also supports

- .**Parametric aggregate functions**, which accept other parameters in addition to columns
- .**Combinators**, which change the behavior of aggregate functions

## NULL processing

.During aggregation, all **NULLs** are skipped

### :Examples

:Consider this table

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

:Let's say you need to total the values in the **y** column

```
SELECT sum(y) FROM t_null_big (:  
  
(SELECT sum(y)  
FROM t_null_big  
  
--(sum(y--  
| 7 |  
|_____|  
  
.rows in set. Elapsed: 0.002 sec 1
```

The **sum** function interprets **NULL** as **0**. In particular, this means that if the function receives input of a selection where all the values are **NULL**, then the result will be **0**, not **NULL**

:Now you can use the **groupArray** function to create an array from the **y** column

```
SELECT groupArray(y) FROM t_null_big (:  
  
(SELECT groupArray(y)  
FROM t_null_big  
  
--(groupArray(y--  
| [2,2,3] |  
|_____|  
  
.rows in set. Elapsed: 0.002 sec 1
```

.**groupArray** does not include **NULL** in the resulting array

## Function Reference

### count

.Counts the number of rows or not-NULL values

:ClickHouse supports the following syntaxes for **count**

- .**count(expr)** or **COUNT(DISTINCT expr)** -
- .**count()** or **COUNT(\*)**. The **count()** syntax is a ClickHouse-specific implementation -

Parameters

- :The function can take
- .Zero parameters
  - .One **expression**

Returned value

- .If the function is called without parameters it counts the number of rows
- If the **expression** is passed, then the function counts how many times this expression returned not null. If the expression returns a value of the **Nullable** data type, then the result of **count** stays not **Nullable**. The function returns 0 if the expression returned **NULL** for all the rows
- .In both cases the type of the returned value is **UInt64**

Details

ClickHouse supports the **COUNT(DISTINCT ...)** syntax. The behavior of this construction depends on the **count\_distinct\_implementation** setting. It defines which of the **uniq\*** functions is used to perform the operation. By default the **uniqExact** function

A **SELECT count() FROM table** query is not optimized, because the number of entries in the table is not stored separately. It chooses some small column from the table and count the number of values in it

Examples

:Example 1

```
SELECT count() FROM t
```

count()
5

:Example 2

```
'SELECT name, value FROM system.settings WHERE name = 'count_distinct_implementation
```

name	value
count_distinct_implementation	uniqExact

```
SELECT count(DISTINCT num) FROM t
```

uniqExact(num)
3

This example shows that **count(DISTINCT num)** is performed by the function **uniqExact** corresponding to the **count\_distinct\_implementation** setting value

(any(x

- .Selects the first encountered value
- The query can be executed in any order and even in a different order each time, so the result of this function is indeterminate
- .To get a determinate result, you can use the 'min' or 'max' function instead of 'any

In some cases, you can rely on the order of execution. This applies to cases when `SELECT` comes from a subquery that uses `ORDER BY`

When a `SELECT` query has the `GROUP BY` clause or at least one aggregate function, ClickHouse (in contrast to MySQL) requires that all expressions in the `SELECT`, `HAVING`, and `ORDER BY` clauses be calculated from keys or from aggregate functions. In other words, each column selected from the table must be used either in keys or inside aggregate functions. To get behavior like in MySQL, you can put the other columns in the `any` aggregate function

## (anyHeavy(x

Selects a frequently occurring value using the `heavy hitters` algorithm. If there is a value that occurs more than in half the cases in each of the query's execution threads, this value is returned. Normally, the result is nondeterministic

```
(anyHeavy(column
```

### Arguments

`column` – The column name

### Example

Take the `OnTime` data set and select any frequently occurring value in the `AirlineID` column

```
SELECT anyHeavy(AirlineID) AS res
FROM ontime
```

```
┌--res--┐
│ 19690 │
└-----┘
```

## (anyLast(x

Selects the last value encountered

The result is just as indeterminate as for the `any` function

## groupBitAnd

Applies bitwise `AND` for series of numbers

```
(groupBitAnd(expr
```

### Parameters

`expr` – An expression that results in `UInt*` type

### Return value

Value of the `UInt*` type

### Example

:Test data

```
binary  decimal
44 = 00101100
28 = 00011100
13 = 00001101
85 = 01010101
```

:Query

```
SELECT groupBitAnd(num) FROM t
```

Where `num` is the column with the test data

:Result

binary	decimal
4	= 00000100

## groupBitOr

.Applies bitwise **OR** for series of numbers

(groupBitOr(expr
------------------

### Parameters

.**expr** - An expression that results in **UInt\*** type

### Return value

.Value of the **UInt\*** type

### Example

:Test data

binary	decimal
44	= 00101100
28	= 00011100
13	= 00001101
85	= 01010101

:Query

SELECT groupBitOr(num) FROM t
-------------------------------

.Where **num** is the column with the test data

:Result

binary	decimal
125	= 01111101

## groupBitXor

.Applies bitwise **XOR** for series of numbers

(groupBitXor(expr
-------------------

### Parameters

.**expr** - An expression that results in **UInt\*** type

### Return value

.Value of the **UInt\*** type

### Example

:Test data

binary	decimal
44	= 00101100
28	= 00011100
13	= 00001101
85	= 01010101

:Query

SELECT groupBitXor(num) FROM t
--------------------------------

.Where **num** is the column with the test data

:Result

binary	decimal
104	= 01101000

## groupByBitmap

Bitmap or Aggregate calculations from a unsigned integer column, return cardinality of type UInt64, if add suffix - .State, then return **bitmap object**

(groupByBitmap(expr
---------------------

### Parameters

.**expr** – An expression that results in **UInt\*** type

### Return value

.Value of the **UInt64** type

### Example

:Test data

userid
1
1
2
3

:Query

SELECT groupBitmap(userid) as num FROM t
--

:Result

num
3

## (min(x

.Calculates the minimum

## (max(x

.Calculates the maximum

## (argMin(arg, val

Calculates the 'arg' value for a minimal 'val' value. If there are several different values of 'arg' for minimal values .of 'val', the first of these values encountered is output

### :Example



user	salary
director	5000
manager	3000
worker	1000

```
SELECT argMin(user, salary) FROM salary
```

(argMin(user, salary)
worker

## (argMax(arg, val

Calculates the 'arg' value for a maximum 'val' value. If there are several different values of 'arg' for maximum values of 'val', the first of these values encountered is output

## (sum(x

- .Calculates the sum
- .Only works for numbers

## (sumWithOverflow(x

Computes the sum of the numbers, using the same data type for the result as for the input parameters. If the sum exceeds the maximum value for this data type, the function returns an error

- .Only works for numbers

## (sumMap(key, value

- .Totals the 'value' array according to the keys specified in the 'key' array
- .The number of elements in 'key' and 'value' must be the same for each row that is totaled
- .Returns a tuple of two arrays: keys in sorted order, and values summed for the corresponding keys

:Example

```
CREATE TABLE sum_map
, date Date
, timeslot DateTime
) statusMap Nested
, status UInt16
requests UInt64
(
;ENGINE = Log (
INSERT INTO sum_map VALUES
, ([10,10,10],[3,2,1], '00:00:00 2000-01-01', '2000-01-01')
, ([10,10,10],[5,4,3], '00:00:00 2000-01-01', '2000-01-01')
, ([10,10,10],[6,5,4], '00:01:00 2000-01-01', '2000-01-01')
, ([10,10,10],[8,7,6], '00:01:00 2000-01-01', '2000-01-01')
SELECT
, timeslot
(sumMap(statusMap.status, statusMap.requests)
FROM sum_map
GROUP BY timeslot
```

(timeslot	sumMap(statusMap.status, statusMap.requests)
([10,10,20,10,10],[1,2,3,4,5])	00:00:00 2000-01-01
([10,10,20,10,10],[4,5,6,7,8])	00:01:00 2000-01-01

## skewPop

- .Computes the **skewness** for sequence

```
(skewPop(expr
```

### Parameters

.*expr* — **Expression** returning a number

### Returned value

The skewness of given distribution. Type — **Float64**

### Example of Use

```
SELECT skewPop(value) FROM series_with_value_column
```

## skewSamp

.Computes the **sample skewness** for sequence

.It represents an unbiased estimate of the skewness of a random variable, if passed values form it's sample

```
(skewSamp(expr
```

### Parameters

.*expr* — **Expression** returning a number

### Returned value

The skewness of given distribution. Type — **Float64**. If  $n \leq 1$  ( $n$  is a size of the sample), then the function returns **.nan**

### Example of Use

```
SELECT skewSamp(value) FROM series_with_value_column
```

## kurtPop

.Computes the **kurtosis** for sequence

```
(kurtPop(expr
```

### Parameters

.*expr* — **Expression** returning a number

### Returned value

The kurtosis of given distribution. Type — **Float64**

### Example of Use

```
SELECT kurtPop(value) FROM series_with_value_column
```

## kurtSamp

.Computes the **sample kurtosis** for sequence

.It represents an unbiased estimate of the kurtosis of a random variable, if passed values form it's sample

```
(kurtSamp(expr
```

### Parameters

.*expr* — **Expression** returning a number

### Returned value

The kurtosis of given distribution. Type — **Float64**. If  $n \leq 1$  ( $n$  is a size of the sample), then the function returns **.nan**

Example of Use

```
SELECT kurtSamp(value) FROM series_with_value_column
```

(timeSeriesGroupSum(uid, timestamp, value

**.timeSeriesGroupSum** can aggregate different time series that sample timestamp not alignment  
.It will use linear interpolation between two sample timestamp and then sum time-series together

- .uid** is the time series unique id, **UInt64**
- .timestamp** is Int64 type in order to support millisecond or microsecond
- .value** is the metric

.The function returns array of tuples with (timestamp, aggregated\_value) pairs

.Before using this function make sure **timestamp** is in ascending order

:Example

uid	timestamp	value
0.2	2	1
0.7	7	1
1.2	12	1
1.7	17	1
2.5	25	1
0.6	3	2
1.6	8	2
2.4	12	2
3.6	18	2
4.8	24	2

```
CREATE TABLE time_series
,uid      UInt64
,timestamp Int64
,value    Float64
;ENGINE = Memory (
INSERT INTO time_series VALUES
,(1,25,2.5),(1,17,1.7),(1,12,1.2),(1,7,0.7),(1,2,0.2)
;(2,24,4.8),(2,18,3.6),(2,12,2.4),(2,8,1.6),(2,3,0.6)

(SELECT timeSeriesGroupSum(uid, timestamp, value
) FROM
SELECT * FROM time_series order by timestamp ASC
;{
```

:And the result will be

```
[(25,2.5),(24,7.2),(18,5.4),(17,5.1),(12,3.6),(8,2.4),(7,2.1),(3,0.9),(2,0.2)]
```

(timeSeriesGroupRateSum(uid, ts, val

Similarly timeSeriesGroupRateSum, timeSeriesGroupRateSum will Calculate the rate of time-series and then sum  
.rates together

.Also, timestamp should be in ascend order before use this function

:Use this function, the result above case will be

```
[(25,0.1),(24,0.3),(18,0.3),(17,0.3),(12,0.3),(8,0.3),(7,0.3),(3,0.1),(2,0)]
```

(avg(x

- .Calculates the average
- .Only works for numbers
- .The result is always Float64

## uniq

- .Calculates the approximate number of different values of the argument

```
([... ],)uniq(x
```

### Parameters

- Function takes the variable number of parameters. Parameters can be of types: [Tuple](#), [Array](#), [Date](#), [DateTime](#), [String](#),  
.numeric types

### Returned value

- .The number of the [UInt64](#) type

### Implementation details

:Function

- .Calculates a hash for all parameters in the aggregate, then uses it in calculations
- Uses an adaptive sampling algorithm. For the calculation state, the function uses a sample of element hash
- .values with a size up to 65536

This algorithm is very accurate and very efficient on CPU. When query contains several of these functions,  
.using [uniq](#) is almost as fast as using other aggregate functions

- .(Provides the result deterministically (it doesn't depend on the order of query processing
- .We recommend to use this function in almost all scenarios

### See Also

- [uniqCombined](#)
- [uniqHLL12](#)
- [uniqExact](#)

## uniqCombined

- .Calculates the approximate number of different values of the argument

```
([... ],)uniqCombined(HLL_precision)(x
```

The [uniqCombined](#) function is a good choice for calculating the number of different values, but keep in mind that  
the estimation error for large sets (200 million elements and more) will become larger than theoretical value due  
.to poor choice of hash function

### Parameters

- Function takes the variable number of parameters. Parameters can be of types: [Tuple](#), [Array](#), [Date](#), [DateTime](#), [String](#),  
.numeric types

[HLL\\_precision](#) is the base-2 logarithm of the number of cells in [HyperLogLog](#). Optional, you can use the function as  
[uniqCombined](#)(x[, ...]). The default value for [HLL\\_precision](#) is 17, that is effectively 96 KiB of space (2^17 cells of 6 bits  
.each

### Returned value

- .The number of the [UInt64](#) type

### Implementation details

:Function

- .Calculates a hash for all parameters in the aggregate, then uses it in calculations
- .Uses combination of three algorithms: array, hash table and HyperLogLog with an error correction table

For small number of distinct elements, the array is used. When the set size becomes larger the hash table is used, while it is smaller than HyperLogLog data structure. For larger number of elements, the HyperLogLog is used, and it will occupy fixed amount of memory

- .(Provides the result deterministically (it doesn't depend on the order of query processing

:In comparison with the [uniq](#) function the [uniqCombined](#)

- .Consumes several times less memory
- .Calculates with several times higher accuracy
- Performs slightly lower usually. In some scenarios [uniqCombined](#) can perform better than [uniq](#), for example, .with distributed queries that transmit a large number of aggregation states over the network

### See Also

- [uniq](#)
- [uniqHLL12](#)
- [uniqExact](#)

## uniqHLL12

.Calculates the approximate number of different values of the argument, using the [HyperLogLog](#) algorithm

```
([... ],)uniqHLL12(x
```

### Parameters

Function takes the variable number of parameters. Parameters can be of types: [Tuple](#), [Array](#), [Date](#), [DateTime](#), [String](#), .numeric types

### Returned value

- .The number of the [UInt64](#) type

### Implementation details

:Function

- .Calculates a hash for all parameters in the aggregate, then uses it in calculations
- .Uses the HyperLogLog algorithm to approximate the number of different values of the argument

bit cells are used. The size of the state is slightly more than 2.5 KB. The result is not very accurate (up-5 212 to ~10% error) for small data sets (<10K elements). However, the result is fairly accurate for high-cardinality data sets (10K-100M), with a maximum error of ~1.6%. Starting from 100M, the estimation error increases, and the function will return very inaccurate results for data sets with extremely high cardinality (1B+ .elements

- .(Provides the determinate result (it doesn't depend on the order of query processing

.We don't recommend using this function. In most cases, use the [uniq](#) or [uniqCombined](#) function

### See Also

- [uniq](#)
- [uniqCombined](#)
- [uniqExact](#)

## uniqExact

.Calculates the exact number of different values of the argument

[... ,]uniqExact(x)

.Use the **uniqExact** function if you definitely need an exact result. Otherwise use the **uniq** function

The **uniqExact** function uses more memory than the **uniq**, because the size of the state has unbounded growth as the number of different values increases

### Parameters

Function takes the variable number of parameters. Parameters can be of types: **Tuple**, **Array**, **Date**, **DateTime**, **String**, numeric types

### See Also

**uniq**  
**uniqCombined**  
**uniqHLL12**

- 
- 
- 

## (groupBy(x), groupArray(max\_size))(x

.Creates an array of argument values

.Values can be added to the array in any (indeterminate) order

.The second version (with the **max\_size** parameter) limits the size of the resulting array to **max\_size** elements

[(For example, **groupBy(1)(x)** is equivalent to **[any(x**

In some cases, you can still rely on the order of execution. This applies to cases when **SELECT** comes from a subquery that uses **ORDER BY**

## (groupByInsertAt(x

.Inserts a value into the array in the specified position

Accepts the value and position as input. If several values are inserted into the same position, any of them might end up in the resulting array (the first one will be used in the case of single-threaded execution). If no value is inserted into a position, the position is assigned the default value

:Optional parameters

.The default value for substituting in empty positions

The length of the resulting array. This allows you to receive arrays of the same size for all the aggregate keys. When using this parameter, the default value must be specified

- 
- 

## (groupUniqArray(x), groupUniqArray(max\_size)(x

.Creates an array from different argument values. Memory consumption is the same as for the **uniqExact** function

.The second version (with the **max\_size** parameter) limits the size of the resulting array to **max\_size** elements

[(For example, **groupUniqArray(1)(x)** is equivalent to **[any(x**

## (quantile(level)(x

.Approximates the **level** quantile. **level** is a constant, a floating-point number from 0 to 1

[We recommend using a **level** value in the range of **[0.01, 0.99**

.Don't use a **level** value equal to 0 or 1 – use the **min** and **max** functions for these cases

In this function, as well as in all functions for calculating quantiles, the **level** parameter can be omitted. In this case, (it is assumed to be equal to 0.5 (in other words, the function will calculate the median

.Works for numbers, dates, and dates with times

.Returns: for numbers – **Float64**; for dates – a date; for dates with times – a date with time

- .Uses [reservoir sampling](#) with a reservoir size up to 8192
- .If necessary, the result is output with linear approximation from the two neighboring values
- .This algorithm provides very low accuracy. See also: [quantileTiming](#), [quantileTDigest](#), [quantileExact](#)
- .The result depends on the order of running the query, and is nondeterministic

When using multiple [quantile](#) (and similar) functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the [quantiles](#) (and similar) functions

## (quantileDeterministic(level))(x, determinator

Works the same way as the [quantile](#) function, but the result is deterministic and does not depend on the order of query execution

To achieve this, the function takes a second argument – the "determinator". This is a number whose hash is used instead of a random number generator in the reservoir sampling algorithm. For the function to work correctly, the same determinator value should not occur too often. For the determinator, you can use an event ID, user ID, and so on

.Don't use this function for calculating timings. There is a more suitable function for this purpose: [quantileTiming](#)

## (quantileTiming(level))(x

- .Computes the quantile of 'level' with a fixed precision
- .Works for numbers. Intended for calculating quantiles of page loading time in milliseconds
- .If the value is greater than 30,000 (a page loading time of more than 30 seconds), the result is equated to 30,000
- .If the total value is not more than about 5670, then the calculation is accurate
- .Otherwise
  - .if the time is less than 1024 ms, then the calculation is accurate
  - .otherwise the calculation is rounded to a multiple of 16 ms
- .When passing negative values to the function, the behavior is undefined

The returned value has the Float32 type. If no values were passed to the function (when using [quantileTimingIf](#)), 'nan' is returned. The purpose of this is to differentiate these instances from zeros. See the note on sorting NaNs  
."in "ORDER BY clause

.(The result is determinate (it doesn't depend on the order of query processing

For its purpose (calculating quantiles of page loading times), using this function is more effective and the result is more accurate than for the [quantile](#) function

## (quantileTimingWeighted(level))(x, weight

- Differs from the [quantileTiming](#) function in that it has a second argument, "weights". Weight is a non-negative integer
- .The result is calculated as if the [x](#) value were passed [weight](#) number of times to the [quantileTiming](#) function

## (quantileExact(level))(x

Computes the quantile of 'level' exactly. To do this, all the passed values are combined into an array, which is then partially sorted. Therefore, the function consumes O(n) memory, where 'n' is the number of values that were passed. However, for a small number of values, the function is very effective

## (quantileExactWeighted(level))(x, weight

Computes the quantile of 'level' exactly. In addition, each value is counted with its weight, as if it is present 'weight' times. The arguments of the function can be considered as histograms, where the value 'x' corresponds to a histogram "column" of the height 'weight', and the function itself can be considered as a summation of histograms

A hash table is used as the algorithm. Because of this, if the passed values are frequently repeated, the function consumes less RAM than `quantileExact`. You can use this function instead of `quantileExact` and specify the weight as `.1`

## `(quantileTDigest(level))(x`

Approximates the quantile level using the `t-digest` algorithm. The maximum error is 1%. Memory consumption by `.State` is proportional to the logarithm of the number of passed values

The performance of the function is lower than for `quantile` or `quantileTiming`. In terms of the ratio of State size to precision, this function is much better than `quantile`

`.The` result depends on the order of running the query, and is nondeterministic

## `(median(x`

All the quantile functions have corresponding median functions: `median`, `medianDeterministic`, `medianTiming`, `medianTimingWeighted`, `medianExact`, `medianExactWeighted`, `medianTDigest`. They are synonyms and their behavior is identical

## `(quantiles(level1, level2, ...))(x`

All the quantile functions also have corresponding quantiles functions: `quantiles`, `quantilesDeterministic`, `quantilesTiming`, `quantilesTimingWeighted`, `quantilesExact`, `quantilesExactWeighted`, `quantilesTDigest`. These functions calculate all the quantiles of the listed levels in one pass, and return an array of the resulting values

## `(varSamp(x`

`.Calculates` the amount  $\Sigma((x - \bar{x})^2) / (n - 1)$ , where `n` is the sample size and  $\bar{x}$  is the average value of `x`

`.It` represents an unbiased estimate of the variance of a random variable, if passed values form it's sample

`.∞+ Returns` `Float64`. When `n <= 1`, returns

## `(varPop(x`

`.Calculates` the amount  $\Sigma((x - \bar{x})^2) / n$ , where `n` is the sample size and  $\bar{x}$  is the average value of `x`

`.In` other words, dispersion for a set of values. Returns `Float64`

## `(stddevSamp(x`

`.The` result is equal to the square root of `varSamp(x`

## `(stddevPop(x`

`.The` result is equal to the square root of `varPop(x`

## `(topK(N)(column`

Returns an array of the most frequent values in the specified column. The resulting array is sorted in descending order of frequency of values (not by the values themselves)

Implements the `Filtered Space-Saving` algorithm for analyzing TopK, based on the reduce-and-combine algorithm from `Parallel Space Saving`

`(topK(N)(column`



This function doesn't provide a guaranteed result. In certain situations, errors might occur and it might return frequent values that aren't the most frequent values

.We recommend using the  $N < 10$  value; performance is reduced with large  $N$  values. Maximum value of  $N = 65536$

Arguments

- .N' is the number of values'
- .x ' - The column '

Example

.Take the OnTime data set and select the three most frequently occurring values in the AirlineID column

```
SELECT topK(3)(AirlineID) AS res
FROM ontime
```

res
[19393,19790,19805]

(covarSamp(x, y

.(Calculates the value of  $\Sigma((x - \bar{x})(y - \bar{y})) / (n - 1)$

.∞+ Returns Float64. When  $n \leq 1$ , returns

(covarPop(x, y

.Calculates the value of  $\Sigma((x - \bar{x})(y - \bar{y})) / n$

(corr(x, y

.(Calculates the Pearson correlation coefficient:  $\Sigma((x - \bar{x})(y - \bar{y})) / \sqrt{\Sigma((x - \bar{x})^2) * \Sigma((y - \bar{y})^2)}$

simpleLinearRegression

.Performs simple (unidimensional) linear regression

```
(simpleLinearRegression(x, y
```

:Parameters

- .x — Column with values of dependent variable
- .y — Column with explanatory variable

:Returned values

.Parameters (a, b) of the resulting line  $y = a*x + b$

Examples

```
([SELECT arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [0, 1, 2, 3
```

([arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [0, 1, 2, 3]
(1,0)

```
([SELECT arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [3, 4, 5, 6
```

([arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [3, 4, 5, 6]
(1,3)

stochasticLinearRegression

This function implements stochastic linear regression. It supports custom parameters for learning rate, L2 regularization coefficient, mini-batch size and has few methods for updating weights ([simple SGD](#), [Momentum](#), [Nesterov](#)).

## Parameters

There are 4 customizable parameters. They are passed to the function sequentially, but there is no need to pass all four - default values will be used, however good model required some parameter tuning

```
('stochasticLinearRegression(1.0, 1.0, 10, 'SGD
```

- [learning rate](#) is the coefficient on step length, when gradient descent step is performed. Too big learning rate may cause infinite weights of the model. Default is [0.00001](#) .1
- [L2 regularization coefficient](#) which may help to prevent overfitting. Default is [0.1](#) .2
- [mini-batch size](#) sets the number of elements, which gradients will be computed and summed to perform one step of gradient descent. Pure stochastic descent uses one element, however having small batches (about 10 elements) make gradient steps more stable. Default is [15](#) .3
- [method for updating weights](#), there are 3 of them: [SGD](#), [Momentum](#), [Nesterov](#). [Momentum](#) and [Nesterov](#) require little bit more computations and memory, however they happen to be useful in terms of speed of convergence and stability of stochastic gradient methods. Default is ['SGD](#) .4

## Usage

[stochasticLinearRegression](#) is used in two steps: fitting the model and predicting on new data. In order to fit the model and save its state for later usage we use [-State](#) combinator, which basically saves the state (model weights, etc)

To predict we use function [evalMLMethod](#), which takes a state as an argument as well as features to predict on

### Fitting .1

Such query may be used

```
sql```\nCREATE TABLE IF NOT EXISTS train_data\n(\n,param1 Float64\n,param2 Float64\n,target Float64\n;ENGINE = Memory (\n\nCREATE TABLE your_model ENGINE = Memory AS SELECT\n(stochasticLinearRegressionState(0.1, 0.0, 5, 'SGD')(target, param1, param2\n;AS state FROM train_data\n\n```\n
```

Here we also need to insert data into `train\_data` table. The number of parameters is not fixed, it depends only on number of arguments, passed into `linearRegressionState`. They all must be numeric values  
Note that the column with target value (which we would like to learn to predict) is inserted as the first argument

### Predicting

.2

After saving a state into the table, we may use it multiple times for prediction, or even merge with other states and create new even better models

```
sql WITH (SELECT state FROM your_model) AS model SELECT evalMLMethod(model, param1, param2) FROM test_data
```

The query will return a column of predicted values. Note that first argument of [evalMLMethod](#) is [AggregateFunctionState](#) object, next are columns of features

[test\\_data](#) is a table like [train\\_data](#) but may not contain target value

## Notes

:To merge two models user may create such query .1

`sql SELECT state1 + state2 FROM your_models`

.where `your_models` table contains both models. This query will return new `AggregateFunctionState` object

User may fetch weights of the created model for its own purposes without saving the model if no `-State` .2  
.combinator is used

`sql SELECT stochasticLinearRegression(0.01)(target, param1, param2) FROM train_data`

Such query will fit the model and return its weights - first are weights, which correspond to the parameters of

.the model, the last one is bias. So in the example above the query will return a column with 3 values

### See Also

[stochasticLogisticRegression](#) •

[Difference between linear and logistic regressions](#) •

## stochasticLogisticRegression

This function implements stochastic logistic regression. It can be used for binary classification problem, supports

.the same custom parameters as `stochasticLinearRegression` and works the same way

### Parameters

:Parameters are exactly the same as in `stochasticLinearRegression`

.[learning rate](#), [l2 regularization coefficient](#), [mini-batch size](#), [method for updating weights](#)

.For more information see [parameters](#)

```
('stochasticLogisticRegression(1.0, 1.0, 10, 'SGD
```

Fitting .1

.See the [Fitting](#) section in the [stochasticLinearRegression](#) description

.[Predicted labels have to be in [-1, 1

Predicting .2

.Using saved state we can predict probability of object having label `1`

`sql WITH (SELECT state FROM your_model) AS model SELECT evalMLMethod(model, param1, param2) FROM test_data`

The query will return a column of probabilities. Note that first argument of `evalMLMethod` is

.`AggregateFunctionState` object, next are columns of features

.We can also set a bound of probability, which assigns elements to different labels

`sql SELECT ans < 1.1 AND ans > 0.5 FROM (WITH (SELECT state FROM your_model) AS model SELECT evalMLMethod(model, (param1, param2) AS ans FROM test_data`

.Then the result will be labels

.`test_data` is a table like `train_data` but may not contain target value

### See Also

[stochasticLinearRegression](#) •

[Difference between linear and logistic regressions](#) •

## Aggregate function combinators

The name of an aggregate function can have a suffix appended to it. This changes the way the aggregate function

.works

## If-

The suffix -If can be appended to the name of any aggregate function. In this case, the aggregate function accepts an extra argument – a condition (UInt8 type). The aggregate function processes only the rows that trigger the .condition. If the condition was not triggered even once, it returns a default value (usually zeros or empty strings

Examples: `sumIf(column, cond)`, `countIf(cond)`, `avgIf(x, cond)`, `quantilesTimingIf(level1, level2)(x, cond)`, `argMinIf(arg, val, cond)` .and so on

With conditional aggregate functions, you can calculate aggregates for several conditions at once, without using subqueries and `JOINS`. For example, in Yandex.Metrica, conditional aggregate functions are used to implement the .segment comparison functionality

## Array-

The -Array suffix can be appended to any aggregate function. In this case, the aggregate function takes arguments of the 'Array(T)' type (arrays) instead of 'T' type arguments. If the aggregate function accepts multiple arguments, this must be arrays of equal lengths. When processing arrays, the aggregate function works like the .original aggregate function across all array elements

Example 1: `sumArray(arr)` - Totals all the elements of all 'arr' arrays. In this example, it could have been written .((more simply: `sum(arraySum(arr`

Example 2: `uniqArray(arr)` - Count the number of unique elements in all 'arr' arrays. This could be done an easier .way: `uniq(arrayJoin(arr))`, but it's not always possible to add 'arrayJoin' to a query

If and -Array can be combined. However, 'Array' must come first, then 'If'. Examples: `uniqArrayIf(arr, cond)`,  
.`quantilesTimingArrayIf(level1, level2)(arr, cond)`. Due to this order, the 'cond' argument can't be an array

## State-

If you apply this combinator, the aggregate function doesn't return the resulting value (such as the number of unique values for the `uniq` function), but an intermediate state of the aggregation (for `uniq`, this is the hash table for calculating the number of unique values). This is an `AggregateFunction(...)` that can be used for further processing or stored in a table to finish aggregating later. See the sections "AggregatingMergeTree" and "Functions for .working with intermediate aggregation states

## Merge-

If you apply this combinator, the aggregate function takes the intermediate aggregation state as an argument, .combines the states to finish aggregation, and returns the resulting value

## .MergeState-

Merges the intermediate aggregation states in the same way as the -Merge combinator. However, it doesn't return .the resulting value, but an intermediate aggregation state, similar to the -State combinator

## ForEach-

Converts an aggregate function for tables into an aggregate function for arrays that aggregates the corresponding array items and returns an array of results. For example, `sumForEach` for the arrays `[1, 2]`, `[3, 4, 5]` and `[6, 7]` returns .the result `[10, 13, 5]` after adding together the corresponding array items

## Parametric aggregate functions

Some aggregate functions can accept not only argument columns (used for compression), but a set of parameters – constants for initialization. The syntax is two pairs of brackets instead of one. The first is for parameters, and the .second is for arguments

## (... ,sequenceMatch(pattern)(time, cond1, cond2

.Pattern matching for event chains

`.pattern` is a string containing a pattern to match. The pattern is similar to a regular expression

`.time` is the time of the event, type support: `Date`, `DateTime`, and other unsigned integer types

`cond1, cond2 ...` is from one to 32 arguments of type `UInt8` that indicate whether a certain condition was met for the `.event`

The function collects a sequence of events in RAM. Then it checks whether this sequence matches the pattern. It returns `UInt8`: 0 if the pattern isn't matched, or 1 if it matches

(%Example: `sequenceMatch ('(?1).*(?2)')(EventTime, URL LIKE '%company%', URL LIKE '%cart`

whether there was a chain of events in which a pageview with 'company' in the address occurred earlier than •  
a pageview with 'cart' in the address

:This is a singular example. You could write it using other aggregate functions

```
('%minIf(EventTime, URL LIKE '%company%') < maxIf(EventTime, URL LIKE '%cart
```

However, there is no such solution for more complex situations

:Pattern syntax

.(refers to the condition (any number can be used in place of 1 (`1?`)

.is any number of any events `*`.

.`t>=1800`) is a time condition?)

.Any quantity of any type of events is allowed over the specified time

.`=>`, `<`, `>`:Instead of `>=`, the following operators can be used

.Any number may be specified in place of 1800

Events that occur during the same second can be put in the chain in any order. This may affect the result of the `.function`

## (... ,sequenceCount(pattern)(time, cond1, cond2

Works the same way as the `sequenceMatch` function, but instead of returning whether there is an event chain, it `.returns` `UInt64` with the number of event chains found

Chains are searched for without overlapping. In other words, the next chain can start only after the end of the `.previous` one

## (... ,windowFunnel(window)(timestamp, cond1, cond2, cond3

Searches for event chains in a sliding time window and calculates the maximum number of events that occurred `.from` the chain

```
(... ,windowFunnel(window)(timestamp, cond1, cond2, cond3
```

### :Parameters

`.window` — Length of the sliding window in seconds

`timestamp` — Name of the column containing the timestamp. Data type support: `Date`, `DateTime`, and other unsigned integer types (Note that though timestamp support `UInt64` type, there is a limitation it's value can't `.(overflow` maximum of `Int64`, which is  $2^{63} - 1$

`.cond1, cond2...` — Conditions or data describing the chain of events. Data type: `UInt8`. Values can be 0 or 1

### Algorithm

The function searches for data that triggers the first condition in the chain and sets the event counter to 1. •  
This is the moment when the sliding window starts

If events from the chain occur sequentially within the window, the counter is incremented. If the sequence of events is disrupted, the counter isn't incremented

If the data has multiple event chains at varying points of completion, the function will only output the size of the longest chain

### Returned value

Integer. The maximum number of consecutive triggered conditions from the chain within the sliding time window. All the chains in the selection are analyzed

### Example

.Determine if one hour is enough for the user to select a phone and purchase it in the online store

:Set the following chain of events

.(The user logged in to their account on the store (eventID=1001 .1

.(The user searches for a phone (eventID = 1003, product = 'phone' .2

.(The user placed an order (eventID = 1009 .3

:To find out how far the user user\_id could get through the chain in an hour in January of 2017, make the query

```
SELECT
,level
count() AS c
FROM
)
SELECT
,user_id
windowFunnel(3600)(timestamp, eventID = 1001, eventID = 1003 AND product = 'phone', eventID = 1009) AS level
FROM trend_event
('WHERE (event_date >= '2017-01-01') AND (event_date <= '2017-01-31
GROUP BY user_id
(
GROUP BY level
ORDER BY level
```

.Simply, the level value could only be 0, 1, 2, 3, it means the maxium event action stage that one user could reach

## (... ,retention(cond1, cond2

.Retention refers to the ability of a company or product to retain its customers over some specified periods

cond1, cond2 ... is from one to 32 arguments of type UInt8 that indicate whether a certain condition was met for the event

:Example

Consider you are doing a website analytics, intend to calculate the retention of customers

This could be easily calculate by retention

```

SELECT
, sum(r[1]) AS r1
, sum(r[2]) AS r2
, sum(r[3]) AS r3
FROM
)
SELECT
, uid
retention(date = '2018-08-10', date = '2018-08-11', date = '2018-08-12') AS r
FROM events
(WHERE date IN ('2018-08-10', '2018-08-11', '2018-08-12
GROUP BY uid
(

```

Simply, **r1** means the number of unique visitors who met the **cond1** condition, **r2** means the number of unique visitors who met **cond1** and **cond2** conditions, **r3** means the number of unique visitors who met **cond1** and **cond3** conditions

## (uniqUpTo(N))(x

Calculates the number of different argument values if it is less than or equal to N. If the number of different argument values is greater than N, it returns N + 1

.Recommended for use with small Ns, up to 10. The maximum value of N is 100

For the state of an aggregate function, it uses the amount of memory equal to 1 + N \* the size of one value of bytes

.For strings, it stores a non-cryptographic hash of 8 bytes. That is, the calculation is approximated for strings

.The function also works for several arguments

It works as fast as possible, except for cases when a large N value is used and the number of unique values is slightly less than N

:Usage example

.Problem: Generate a report that shows only keywords that produced at least 5 unique users  
Solution: Write in the GROUP BY query SearchPhrase HAVING uniqUpTo(4)(UserID) >= 5

## (sumMapFiltered(keys\_to\_keep)(keys, values

Same behavior as **sumMap** except that an array of keys is passed as a parameter. This can be especially useful when working with a high cardinality of keys

## Table functions

.Table functions can be specified in the FROM clause instead of the database and table names

.Table functions can only be used if 'readonly' is not set

.Table functions aren't related to other functions

## file

.Creates a table from a file

```
(file(path, format, structure
```

### Input parameters

- .**path** — The relative path to the file from **user\_files\_path**
- .**format** — The **format** of the file
- .**'... ,structure** — Structure of the table. Format '**column1\_name column1\_type, column2\_name column2\_type**

### Returned value

.A table with the specified structure for reading or writing data in the specified file

## Example

:Setting `user_files_path` and the contents of the file `test.csv`

```
grep user_files_path /etc/clickhouse-server/config.xml $
<user_files_path>/var/lib/clickhouse/user_files/</user_files_path>

cat /var/lib/clickhouse/user_files/test.csv $
1,2,3
3,2,1
78,43,45
```

:Table from `test.csv` and selection of the first two rows from it

```
* SELECT
('FROM file('test.csv', 'CSV', 'column1 UInt32, column2 UInt32, column3 UInt32
LIMIT 2
```

column1	column2	column3
3	2	1
1	2	3

getting the first 10 lines of a table that contains 3 columns of UInt32 type from a CSV file --  
SELECT \* FROM file('test.csv', 'CSV', 'column1 UInt32, column2 UInt32, column3 UInt32') LIMIT 10

## merge

`merge(db_name, 'tables_regexp')` - Creates a temporary Merge table. For more information, see the section "Table engines, Merge"

.The table structure is taken from the first table encountered that matches the regular expression

## numbers

.`numbers(N)` - Returns a table with the single 'number' column (UInt64) that contains integers from 0 to N-1

.`numbers(N, M)` - Returns a table with the single 'number' column (UInt64) that contains integers from N to (N + M - 1)

Similar to the `system.numbers` table, it can be used for testing and generating successive values, `numbers(N, M)` more efficient than `system.numbers`

:The following queries are equivalent

```
;(SELECT * FROM numbers(10)
;(SELECT * FROM numbers(0, 10)
;(SELECT * FROM system.numbers LIMIT 10
```

:Examples

```
Generate a sequence of dates from 2010-01-01 to 2010-12-31 --
;(select toDate('2010-01-01') + number as d FROM numbers(365
```

## remote, remoteSecure

.Allows you to access remote servers without creating a `Distributed` table

:Signatures

```
((['remote('addresses_expr', db, table[, 'user'[, 'password
((['remote('addresses_expr', db.table[, 'user'[, 'password
```



`addresses_expr` – An expression that generates addresses of remote servers. This may be just one server address. The server address is `host:port`, or just `host`. The host can be specified as the server name, or as the IPv4 or IPv6 address. An IPv6 address is specified in square brackets. The port is the TCP port on the remote server. If the port is omitted, it uses `tcp_port` from the server's config file (by default, 9000)

Important

.The port is required for an IPv6 address

:Examples

```
example01-01-1
example01-01-1:9000
localhost
127.0.0.1
9000[::]
2a02:6b8:0:1111::11]:9000]
```

Multiple addresses can be comma-separated. In this case, ClickHouse will use distributed processing, so it will send the query to all specified addresses (like to shards with different data)

:Example

```
example01-01-1,example01-02-1
```

:Part of the expression can be specified in curly brackets. The previous example can be written as follows

```
example01-0{1,2}-1
```

Curly brackets can contain a range of numbers separated by two dots (non-negative integers). In this case, the range is expanded to a set of values that generate shard addresses. If the first number starts with zero, the values are formed with the same zero alignment. The previous example can be written as follows

```
example01-{01..02}-1
```

.If you have multiple pairs of curly brackets, it generates the direct product of the corresponding sets

Addresses and parts of addresses in curly brackets can be separated by the pipe symbol (`|`). In this case, the corresponding sets of addresses are interpreted as replicas, and the query will be sent to the first healthy replica. However, the replicas are iterated in the order currently set in the `load_balancing` setting

:Example

```
{example01-{01..02}-{1|2}
```

.This example specifies two shards that each have two replicas

.The number of addresses generated is limited by a constant. Right now this is 1000 addresses

Using the `remote` table function is less optimal than creating a `Distributed` table, because in this case, the server connection is re-established for every request. In addition, if host names are set, the names are resolved, and errors are not counted when working with various replicas. When processing a large number of queries, always create the `Distributed` table ahead of time, and don't use the `remote` table function

:The `remote` table function can be useful in the following cases

- .Accessing a specific server for data comparison, debugging, and testing
- .Queries between various ClickHouse clusters for research purposes
- .Infrequent distributed requests that are made manually
- .Distributed requests where the set of servers is re-defined each time

.If the user is not specified, `default` is used

.If the password is not specified, an empty password is used

`.remoteSecure` - same as `remote` but with secured connection. Default port — `tcp_port_secure` from config or 9440

## url

`url(URL, format, structure)` - returns a table created from the `URL` with given

`.format` and `structure`

`.URL` - HTTP or HTTPS server address, which can accept `GET` and/or `POST` requests

`.format` - `format` of the data

`.structure` - table structure in '`UserID UInt64, Name String`' format. Determines column names and types

### Example

```
getting the first 3 lines of a table that contains columns of String and UInt32 type from HTTP-server which answers in CSV --
.format
SELECT * FROM url('http://127.0.0.1:12345/', CSV, 'column1 String, column2 UInt32') LIMIT 3
```

## mysql

`.Allows SELECT` queries to be performed on data that is stored on a remote MySQL server

```
;([mysql('host:port', 'database', 'table', 'user', 'password'[, replace_query, 'on_duplicate_clause
```

### Parameters

`.host:port` — MySQL server address

`.database` — Remote database name

`.table` — Remote table name

`.user` — MySQL user

`.password` — User password

`replace_query` — Flag that converts `INSERT INTO` queries to `REPLACE INTO`. If `replace_query=1`, the query is replaced

`on_duplicate_clause` — The `ON DUPLICATE KEY on_duplicate_clause` expression that is added to the `INSERT` query

Example: `INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1` where `on_duplicate_clause` is `UPDATE c2 = c2 + 1`. See the MySQL documentation to find which `on_duplicate_clause` you can use with the `ON DUPLICATE KEY` clause

To specify `on_duplicate_clause` you need to pass `0` to the `replace_query` parameter. If you simultaneously pass `replace_query = 1` and `on_duplicate_clause`, ClickHouse generates an exception

Simple `WHERE` clauses such as `=`, `!=`, `>`, `>=`, `<`, `<=` are currently executed on the MySQL server

The rest of the conditions and the `LIMIT` sampling constraint are executed in ClickHouse only after the query to MySQL finishes

### Returned Value

A table object with the same columns as the original MySQL table

## Usage Example

:Table in MySQL

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

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

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

:Selecting data from ClickHouse

```
('SELECT * FROM mysql('localhost:3306', 'test', 'test', 'bayonet', '123
```

int_id	int_nullable	float	float_nullable
NULL	2	NULL	1

See Also

- [The 'MySQL' table engine](#)
- [Using MySQL as a source of external dictionary](#)

jdbc

.[jdbc\(jdbc\\_connection\\_uri, schema, table\)](#) - returns table that is connected via JDBC driver

.This table function requires separate [clickhouse-jdbc-bridge](#) program to be running  
(It supports Nullable types (based on DDL of remote table that is queried

Examples

```
('SELECT * FROM jdbc('jdbc:mysql://localhost:3306/?user=root&password=root', 'schema', 'table
```

```
('SELECT * FROM jdbc('mysql://localhost:3306/?user=root&password=root', 'schema', 'table
```

```
('SELECT * FROM jdbc('datasource://mysql-local', 'schema', 'table
```

odbc

.Returns table that is connected via [ODBC](#)

```
(odbc(connection_settings, external_database, external_table
```

:Parameters

- .[connection\\_settings](#) — Name of the section with connection settings in the [odbc.ini](#) file
- .[external\\_database](#) — Name of a database in an external DBMS
- .[external\\_table](#) — Name of a table in the [external\\_database](#)

To safely implement ODBC connections, ClickHouse uses a separate program [clickhouse-odbc-bridge](#). If the ODBC driver is loaded directly from [clickhouse-server](#), driver problems can crash the ClickHouse server. ClickHouse automatically starts [clickhouse-odbc-bridge](#) when it is required. The ODBC bridge program is installed from the same package as the [clickhouse-server](#)

The fields with the **NULL** values from the external table are converted into the default values for the base data type. For example, if a remote MySQL table field has the **INT NULL** type it is converted to 0 (the default value for **Int32** data type).

## Usage example

## Getting data from the local MySQL installation via ODBC

.This example is checked for Ubuntu Linux 18.04 and MySQL server 5.7

.Ensure that unixODBC and MySQL Connector are installed

By default (if installed from packages), ClickHouse starts as user `clickhouse`. Thus you need to create and configure this user in the MySQL server

```
sudo mysql
;mysql> CREATE USER 'clickhouse'@'localhost' IDENTIFIED BY 'clickhouse'
;mysql> GRANT ALL PRIVILEGES ON *.* TO 'clickhouse'@'clickhouse' WITH GRANT OPTION
```

.Then configure the connection in `/etc/odbc.ini`

```
cat /etc/odbc.ini $
[mysqlconn]
DRIVER = /usr/local/lib/libmyodbc5w.so
SERVER = 127.0.0.1
PORT = 3306
DATABASE = test
USERNAME = clickhouse
PASSWORD = clickhouse
```

.You can check the connection using the **isql** utility from the unixODBC installation

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

## :Table in MySQL

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

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

;mysql> select * from test

+-----+-----+-----+-----+
| int_id | int_nullable | float | float_nullable |
+-----+-----+-----+-----+
| NULL | 2 | NULL | 1 |
+-----+-----+-----+-----+
(row in set (0,00 sec 1
```

## :Retrieving data from the MySQL table in ClickHouse

```
('SELECT * FROM odbc('DSN=mysqlconn', 'test', 'test
```

int_id	int_nullable	float	float_nullable
0	2	0	1

## See Also

[ODBC external dictionaries](#)

[.ODBC table engine](#)

- 
- 

## Dictionaries

A dictionary is a mapping ([key -> attributes](#)) that is convenient for various types of reference lists

ClickHouse supports special functions for working with dictionaries that can be used in queries. It is easier and more efficient to use dictionaries with functions than a [JOIN](#) with reference tables

[.NULL](#) values can't be stored in a dictionary

:ClickHouse supports

[.Built-in dictionaries](#) with a specific [set of functions](#)

[.Plug-in \(external\) dictionaries](#) with a [set of functions](#)

- 
- 

## External Dictionaries

You can add your own dictionaries from various data sources. The data source for a dictionary can be a local text or executable file, an HTTP(s) resource, or another DBMS. For more information, see "[Sources for external dictionaries](#)"

:ClickHouse

[.Fully or partially stores dictionaries in RAM](#)

Periodically updates dictionaries and dynamically loads missing values. In other words, dictionaries can be loaded dynamically

- 
- 

The configuration of external dictionaries is located in one or more files. The path to the configuration is specified in the [dictionaries\\_config](#) parameter

Dictionaries can be loaded at server startup or at first use, depending on the [dictionaries\\_lazy\\_load](#) setting

:The dictionary config file has the following format

```
<yandex>
<comment>An optional element with any content. Ignored by the ClickHouse server.</comment>

<!--Optional element. File name with substitutions--!>
<include_from>/etc/metrika.xml</include_from>

<dictionary>
<!-- Dictionary configuration --!>
</dictionary>

...

<dictionary>
<!-- Dictionary configuration --!>
</dictionary>
</yandex>
```

You can [configure](#) any number of dictionaries in the same file. The file format is preserved even if there is only one dictionary (i.e. `<yandex><dictionary> <!--configuration --!> </dictionary></yandex>`)

. "See also "[Functions for working with external dictionaries](#)

## Attention

You can convert values for a small dictionary by describing it in a [SELECT](#) query (see the [transform](#) function). This functionality is not related to external dictionaries

# Configuring an External Dictionary

:The dictionary configuration has the following structure

```
<dictionary>
<name>dict_name</name>

<source>
<-- Source configuration --!>
</source>

<layout>
<-- Memory layout configuration --!>
</layout>

<structure>
<-- Complex key configuration --!>
</structure>

<lifetime>
<-- Lifetime of dictionary in memory --!>
</lifetime>
</dictionary>
```

- . [dict\\_name](#) — The identifier that can be used to access the dictionary. Use the characters [\[a-zA-Z0-9\]](#)
- . [source](#) — Source of the dictionary
- . [layout](#) — Dictionary layout in memory
- . [structure](#) — Structure of the dictionary . A key and attributes that can be retrieved by this key
- . [lifetime](#) — Frequency of dictionary updates

## Storing Dictionaries in Memory

.There are a variety of ways to store dictionaries in memory

.We recommend [flat](#), [hashed](#) and [complex\\_key\\_hashed](#). which provide optimal processing speed

Caching is not recommended because of potentially poor performance and difficulties in selecting optimal parameters. Read more in the section "[cache](#)

:There are several ways to improve dictionary performance

- .Call the function for working with the dictionary after [GROUP BY](#)
- Mark attributes to extract as injective. An attribute is called injective if different attribute values correspond to different keys. So when [GROUP BY](#) uses a function that fetches an attribute value by the key, this function is automatically taken out of [GROUP BY](#)

:ClickHouse generates an exception for errors with dictionaries. Examples of errors

- .The dictionary being accessed could not be loaded
- .Error querying a [cached](#) dictionary

.You can view the list of external dictionaries and their statuses in the [system.dictionaries](#) table

:The configuration looks like this

```

<yandex>
<dictionary>
...
<layout>
<layout_type>
<-- layout settings --!>
<layout_type/>
<layout/>
...
<dictionary/>
<yandex/>

```

## Ways to Store Dictionaries in Memory

[flat](#)  
[hashed](#)  
[cache](#)  
[range\\_hashed](#)  
[complex\\_key\\_hashed](#)  
[complex\\_key\\_cache](#)  
[ip\\_trie](#)

- 
- 
- 
- 
- 
- 
- 

### flat

The dictionary is completely stored in memory in the form of flat arrays. How much memory does the dictionary use? The amount is proportional to the size of the largest key (in space used

The dictionary key has the [UInt64](#) type and the value is limited to 500,000. If a larger key is discovered when creating the dictionary, ClickHouse throws an exception and does not create the dictionary

All types of sources are supported. When updating, data (from a file or from a table) is read in its entirety

This method provides the best performance among all available methods of storing the dictionary

:Configuration example

```

<layout>
</ flat>
<layout/>

```

### hashed

The dictionary is completely stored in memory in the form of a hash table. The dictionary can contain any number of elements with any identifiers In practice, the number of keys can reach tens of millions of items

All types of sources are supported. When updating, data (from a file or from a table) is read in its entirety

:Configuration example

```

<layout>
</ hashed>
<layout/>

```

### complex\_key\_hashed

This type of storage is for use with composite [keys](#). Similar to [hashed](#)

:Configuration example

```

<layout>
</ complex_key_hashed>
<layout/>

```

### range\_hashed

The dictionary is stored in memory in the form of a hash table with an ordered array of ranges and their corresponding values

This storage method works the same way as hashed and allows using date/time ranges in addition to the key, if they appear in the dictionary

:Example: The table contains discounts for each advertiser in the format

advertiser id	discount start date	discount end date	amount
0.15	2015-01-15	2015-01-01	123
0.25	2015-01-31	2015-01-16	123
0.05	2015-01-15	2015-01-01	456

.To use a sample for date ranges, define the **range\_min** and **range\_max** elements in the **structure**

:Example

```
<structure>
<id>
<name>Id</name>
</id>
<range_min>
<name>first</name>
</range_min>
<range_max>
<name>last</name>
</range_max>
...
```

:To work with these dictionaries, you need to pass an additional date argument to the **dictGetT** function

```
(dictGetT('dict_name', 'attr_name', id, date
```

.This function returns the value for the specified **ids** and the date range that includes the passed date

:Details of the algorithm

- .If the **id** is not found or a range is not found for the **id**, it returns the default value for the dictionary
- .If there are overlapping ranges, you can use any
- If the range delimiter is **NULL** or an invalid date (such as 1900-01-01 or 2039-01-01), the range is left open.
- .The range can be open on both sides

:Configuration example



```

<yandex>
<dictionary>

...

<layout>
</ range_hashed>
<layout/>

<structure>
<id>
<name>Abcdef</name>
<id/>
<range_min>
<name>StartDate</name>
<range_min/>
<range_max>
<name>EndDate</name>
<range_max/>
<attribute>
<name>XXXType</name>
<type>String</type>
</ null_value>
<attribute/>
<structure/>

<dictionary/>
<yandex/>

```

## cache

.The dictionary is stored in a cache that has a fixed number of cells. These cells contain frequently used elements

When searching for a dictionary, the cache is searched first. For each block of data, all keys that are not found in the cache or are outdated are requested from the source using `SELECT attrs... FROM db.table WHERE id IN (k1, k2, ...)`.

.The received data is then written to the cache

For cache dictionaries, the expiration **lifetime** of data in the cache can be set. If more time than **lifetime** has passed since loading the data in a cell, the cell's value is not used, and it is re-requested the next time it needs to be .used

This is the least effective of all the ways to store dictionaries. The speed of the cache depends strongly on correct settings and the usage scenario. A cache type dictionary performs well only when the hit rates are high enough .(recommended 99% and higher). You can view the average hit rate in the `system.dictionaries` table

.To improve cache performance, use a subquery with **LIMIT**, and call the function with the dictionary externally

.Supported **sources**: MySQL, ClickHouse, executable, HTTP

:Example of settings

```

<layout>
<cache>
<-- .The size of the cache, in number of cells. Rounded up to a power of two --!>
<size_in_cells>1000000000</size_in_cells>
<cache/>
<layout/>

```

:Set a large enough cache size. You need to experiment to select the number of cells

- .Set some value .1
- .Run queries until the cache is completely full .2
- .Assess memory consumption using the `system.dictionaries` table .3
- .Increase or decrease the number of cells until the required memory consumption is reached .4

## Warning

.Do not use ClickHouse as a source, because it is slow to process queries with random reads

## complex\_key\_cache

.This type of storage is for use with composite **keys**. Similar to **cache**

## ip\_trie

.This type of storage is for mapping network prefixes (IP addresses) to metadata such as ASN

:Example: The table contains network prefixes and their corresponding AS number and country code

```
+-----+-----+-----+
| prefix      | asn  | cca2 |
+=====+=====+=====+
| NP | 17501 | 202.79.32.0/20 |
+-----+-----+-----+
| US | 3856 | 48::2620:0:870 |
+-----+-----+-----+
| 2a02:6b8:1::/48 | 13238 | RU |
+-----+-----+-----+
| db8::/32 | 65536 | ZZ:2001 |
+-----+-----+-----+
```

.When using this type of layout, the structure must have a composite key

:Example

```
<structure>
<key>
<attribute>
<name>prefix</name>
<type>String</type>
<attribute/>
<key/>
<attribute>
<name>asn</name>
<type>UInt32</type>
</ null_value>
<attribute/>
<attribute>
<name>cca2</name>
<type>String</type>
<null_value>??</null_value>
<attribute/>
...
```

The key must have only one String type attribute that contains an allowed IP prefix. Other types are not supported yet

:For queries, you must use the same functions (**dictGetT** with a tuple) as for dictionaries with composite keys

```
((dictGetT('dict_name', 'attr_name', tuple(ip
```

:The function takes either **UInt32** for IPv4, or **FixedString(16)** for IPv6

```
((('dictGetString('prefix', 'asn', tuple(IPv6StringToNum('2001:db8::1
```

Other types are not supported yet. The function returns the attribute for the prefix that corresponds to this IP address. If there are overlapping prefixes, the most specific one is returned

.Data is stored in a **trie**. It must completely fit into RAM

# Dictionary Updates

ClickHouse periodically updates the dictionaries. The update interval for fully downloaded dictionaries and the .invalidation interval for cached dictionaries are defined in the `<lifetime>` tag in seconds

Dictionary updates (other than loading for first use) do not block queries. During updates, the old version of a dictionary is used. If an error occurs during an update, the error is written to the server log, and queries continue .using the old version of dictionaries

:Example of settings

```
<dictionary>
...
<lifetime>300</lifetime>
...
</dictionary>
```

.Setting `<lifetime> 0</lifetime>` prevents updating dictionaries

You can set a time interval for upgrades, and ClickHouse will choose a uniformly random time within this range. This is necessary in order to distribute the load on the dictionary source when upgrading on a large number of .servers

:Example of settings

```
<dictionary>
...
<lifetime>
<min>300</min>
<max>360</max>
</lifetime>
...
</dictionary>
```

:When upgrading the dictionaries, the ClickHouse server applies different logic depending on the type of **source**

- For a text file, it checks the time of modification. If the time differs from the previously recorded time, the .dictionary is updated
- .For MyISAM tables, the time of modification is checked using a `SHOW TABLE STATUS` query
- .Dictionaries from other sources are updated every time by default

For MySQL (InnoDB), ODBC and ClickHouse sources, you can set up a query that will update the dictionaries only if :they really changed, rather than each time. To do this, follow these steps

- .The dictionary table must have a field that always changes when the source data is updated
- The settings of the source must specify a query that retrieves the changing field. The ClickHouse server interprets the query result as a row, and if this row has changed relative to its previous state, the dictionary .is updated. Specify the query in the `<invalidate_query>` field in the settings for the **source**

:Example of settings

```
<dictionary>
...
<odbc>
...
<invalidate_query>SELECT update_time FROM dictionary_source where id = 1</invalidate_query>
</odbc>
...
</dictionary>
```

## Sources of External Dictionaries

.An external dictionary can be connected from many different sources

:The configuration looks like this

```
<yandex>
<dictionary>
...
<source>
<source_type>
<-- Source configuration --!>
<source_type/>
<source/>
...
<dictionary/>
...
<yandex/>
```

.The source is configured in the **source** section

:(Types of sources (**source\_type**

Local file

Executable file

(HTTP(s

DBMS

MySQL

ClickHouse

MongoDB

ODBC

- 
- 
- 
- 

- 
- 
- 
- 

## Local File

:Example of settings

```
<source>
<file>
<path>/opt/dictionaries/os.tsv</path>
<format>TabSeparated</format>
<file/>
<source/>
```

:Setting fields

.**path** – The absolute path to the file

.**format** – The file format. All the formats described in "**Formats**" are supported

- 
- 

## Executable File

Working with executable files depends on **how the dictionary is stored in memory**. If the dictionary is stored using **cache** and **complex\_key\_cache**, ClickHouse requests the necessary keys by sending a request to the executable file's .STDIN. Otherwise, ClickHouse starts executable file and treats its output as dictionary data

:Example of settings

```
<source>
<executable>
<command>cat /opt/dictionaries/os.tsv</command>
<format>TabSeparated</format>
<executable/>
<source/>
```

:Setting fields

**command** – The absolute path to the executable file, or the file name (if the program directory is written to  
.(**PATH**

-

.[format](#) – The file format. All the formats described in "[Formats](#)" are supported

- 

## (HTTP(s

Working with an HTTP(s) server depends on [how the dictionary is stored in memory](#). If the dictionary is stored using [cache](#) and [complex\\_key\\_cache](#), ClickHouse requests the necessary keys by sending a request via the [POST](#) .method

:Example of settings

```
<source>
<http>
<url>http://[::1]/os.tsv</url>
<format>TabSeparated</format>
<http/>
<source/>
```

.In order for ClickHouse to access an HTTPS resource, you must [configure openSSL](#) in the server configuration

:Setting fields

.[url](#) – The source URL

- 

.[format](#) – The file format. All the formats described in "[Formats](#)" are supported

- 

## ODBC

.You can use this method to connect any database that has an ODBC driver

:Example of settings

```
<odbc>
<db>DatabaseName</db>
<table>ShemaName.TableName</table>
<connection_string>DSN=some_parameters</connection_string>
<invalidate_query>SQL_QUERY</invalidate_query>
<odbc/>
```

:Setting fields

.[db](#) – Name of the database. Omit it if the database name is set in the [<connection\\_string>](#) parameters

- 

.[table](#) – Name of the table and schema if exists

- 

.[connection\\_string](#) – Connection string

- 

[invalidate\\_query](#) – Query for checking the dictionary status. Optional parameter. Read more in the section

- 

.[Updating dictionaries](#)

ClickHouse receives quoting symbols from ODBC-driver and quote all settings in queries to driver, so it's necessary .to set table name accordingly to table name case in database

.If you have a problems with encodings when using Oracle, see the corresponding [FAQ](#) article

## Known vulnerability of the ODBC dictionary functionality

Attention

When connecting to the database through the ODBC driver connection parameter [Servername](#) can be substituted. In this case values of [USERNAME](#) and [PASSWORD](#) from [odbc.ini](#) are sent to the remote server and can be .compromised

### Example of insecure use

:Let's configure unixODBC for PostgreSQL. Content of [/etc/odbc.ini](#)

```
[gregtest]
Driver = /usr/lib/psqlodbc.so
Servername = localhost
PORT = 5432
DATABASE = test_db
OPTION = 3##
USERNAME = test
PASSWORD = test
```

If you then make a query such as

```
;('SELECT * FROM odbc('DSN=gregtest;Servername=some-server.com', 'test_db
```

.ODBC driver will send values of **USERNAME** and **PASSWORD** from **odbc.ini** to **some-server.com**

## Example of Connecting PostgreSQL

.Ubuntu OS

:Installing unixODBC and the ODBC driver for PostgreSQL

```
sudo apt-get install -y unixodbc odbcinst odbc-postgresql
```

:(Configuring **/etc/odbc.ini** (or **~/odbc.ini**

```
[DEFAULT]
Driver = myconnection

[myconnection]
Description      = PostgreSQL connection to my_db
Driver           = PostgreSQL Unicode
Database         = my_db
Servername       = 127.0.0.1
UserName         = username
Password         = password
Port             = 5432
Protocol         = 9.3
ReadOnly         = No
RowVersioning    = No
ShowSystemTables = No
= ConnSettings
```

:The dictionary configuration in ClickHouse

```
<yandex>
<dictionary>
<name>table_name</name>
<source>
<odbc>
<-- :You can specify the following parameters in connection_string --!>
<-- DSN=myconnection;UID=username;PWD=password;HOST=127.0.0.1;PORT=5432;DATABASE=my_db --!>
<connection_string>DSN=myconnection</connection_string>
<table>postgresql_table</table>
<odbc/>
<source/>
<lifetime>
<min>300</min>
<max>360</max>
<lifetime/>
<layout>
</hashed>
<layout/>
<structure>
<id>
<name>id</name>
<id/>
<attribute>
<name>some_column</name>
<type>UInt64</type>
<null_value>0</null_value>
<attribute/>
<structure/>
<dictionary/>
<yandex/>
```

.You may need to edit `odbc.ini` to specify the full path to the library with the driver `DRIVER=/usr/local/lib/psqlodbcw.so`

## Example of Connecting MS SQL Server

.Ubuntu OS

: :Installing the driver

```
sudo apt-get install tdsodbc freetds-bin sqsh
```

: :Configuring the driver

```
cat /etc/freetds/freetds.conf $
```

```
...
```

```
[MSSQL]
```

```
host = 192.168.56.101
```

```
port = 1433
```

```
tds version = 7.0
```

```
client charset = UTF-8
```

```
cat /etc/odbcinst.ini $
```

```
...
```

```
[FreeTDS]
```

```
Description    = FreeTDS
```

```
Driver         = /usr/lib/x86_64-linux-gnu/odbc/libtdsodbc.so
```

```
Setup         = /usr/lib/x86_64-linux-gnu/odbc/libtdsS.so
```

```
FileUsage      = 1
```

```
UsageCount     = 5
```

```
cat ~/.odbc.ini $
```

```
...
```

```
[MSSQL]
```

```
Description    = FreeTDS
```

```
Driver         = FreeTDS
```

```
Servername     = MSSQL
```

```
Database       = test
```

```
UID            = test
```

```
PWD            = test
```

```
Port           = 1433
```

:Configuring the dictionary in ClickHouse

```
<yandex>
```

```
<dictionary>
```

```
<name>test</name>
```

```
<source>
```

```
<odbc>
```

```
<table>dict</table>
```

```
<connection_string>DSN=MSSQL;UID=test;PWD=test</connection_string>
```

```
<odbc/>
```

```
<source/>
```

```
<lifetime>
```

```
<min>300</min>
```

```
<max>360</max>
```

```
<lifetime/>
```

```
<layout>
```

```
</ flat>
```

```
<layout/>
```

```
<structure>
```

```
<id>
```

```
<name>k</name>
```

```
<id/>
```

```
<attribute>
```

```
<name>s</name>
```

```
<type>String</type>
```

```
<null_value></null_value>
```

```
<attribute/>
```

```
<structure/>
```

```
<dictionary/>
```

```
<yandex/>
```



# DBMS

## MySQL

:Example of settings

```
<source>
<mysql>
<port>3306</port>
<user>clickhouse</user>
<password>qwerty</password>
<replica>
<host>example01-1</host>
<priority>1</priority>
<replica/>
<replica>
<host>example01-2</host>
<priority>1</priority>
<replica/>
<db>db_name</db>
<table>table_name</table>
<where>id=10</where>
<invalidate_query>SQL_QUERY</invalidate_query>
<mysql/>
<source/>
```

:Setting fields

- **port** – The port on the MySQL server. You can specify it for all replicas, or for each one individually (inside `.(<<replica`
- **user** – Name of the MySQL user. You can specify it for all replicas, or for each one individually (inside `.(<<replica`
- **password** – Password of the MySQL user. You can specify it for all replicas, or for each one individually (inside `.(<<replica`
- **.replica** – Section of replica configurations. There can be multiple sections
  - **.replica/host** – The MySQL host
  - **replica/priority** – The replica priority. When attempting to connect, ClickHouse traverses the replicas in order \* of priority. The lower the number, the higher the priority
- **.db** – Name of the database
- **.table** – Name of the table
- **.where** – The selection criteria. Optional parameter
- **invalidate\_query** – Query for checking the dictionary status. Optional parameter. Read more in the section [Updating dictionaries](#)
- **.MySQL** can be connected on a local host via sockets. To do this, set **host** and **socket**

:Example of settings

```
<source>
<mysql>
<host>localhost</host>
<socket>/path/to/socket/file.sock</socket>
<user>clickhouse</user>
<password>qwert</password>
<db>db_name</db>
<table>table_name</table>
<where>id=10</where>
<invalidate_query>SQL_QUERY</invalidate_query>
</mysql>
</source>
```

## ClickHouse

:Example of settings

```
<source>
<clickhouse>
<host>example01-01-1</host>
<port>9000</port>
<user>default</user>
<password></password>
<db>default</db>
<table>ids</table>
<where>id=10</where>
</clickhouse>
</source>
```

:Setting fields

**host** – The ClickHouse host. If it is a local host, the query is processed without any network activity. To improve fault tolerance, you can create a **Distributed** table and enter it in subsequent configurations

**port** – The port on the ClickHouse server

**user** – Name of the ClickHouse user

**password** – Password of the ClickHouse user

**db** – Name of the database

**table** – Name of the table

**where** – The selection criteria. May be omitted

**invalidate\_query** – Query for checking the dictionary status. Optional parameter. Read more in the section **Updating dictionaries**

## MongoDB

:Example of settings

```
<source>
<mongodb>
<host>localhost</host>
<port>27017</port>
<user></user>
<password></password>
<db>test</db>
<collection>dictionary_source</collection>
</mongodb>
</source>
```

:Setting fields

**host** – The MongoDB host

**port** – The port on the MongoDB server

**user** – Name of the MongoDB user

**password** – Password of the MongoDB user

- .db – Name of the database
- .collection – Name of the collection

## Dictionary Key and Fields

.The `<structure>` clause describes the dictionary key and fields available for queries

:Overall structure

```
<dictionary>
<structure>
<id>
<name>Id</name>
<id/>

<attribute>
<-- Attribute parameters --!>
<attribute/>

...

<structure/>
<dictionary/>
```

:Columns are described in the structure

- .id> - key column>
- .attribute> - data column. There can be a large number of columns>

## Key

:ClickHouse supports the following types of keys

- . <Numeric key. UInt64. Defined in the tag <id>
- . <Composite key. Set of values of different types. Defined in the tag <key>
- . <A structure can contain either <id> or <key>

Warning

.The key doesn't need to be defined separately in attributes

## Numeric Key

.Format: UInt64

:Configuration example

```
<id>
<name>Id</name>
<id/>
```

:Configuration fields

- .name – The name of the column with keys

## Composite Key

The key can be a tuple from any types of fields. The layout in this case must be complex\_key\_hashed or complex\_key\_cache

Tip

.A composite key can consist of a single element. This makes it possible to use a string as the key, for instance

The key structure is set in the element `<key>`. Key fields are specified in the same format as the dictionary `:attributes`. Example

```
<structure>
<key>
<attribute>
<name>field1</name>
<type>String</type>
<attribute/>
<attribute>
<name>field2</name>
<type>UInt32</type>
<attribute/>
...
<key/>
...
```

For a query to the `dictGet*` function, a tuple is passed as the key. Example: `dictGetString('dict_name', 'attr_name',  
.(tuple('string for field1', num_for_field2`

## Attributes

:Configuration example

```
<structure>
...
<attribute>
<name>Name</name>
<type>Type</type>
<null_value></null_value>
<expression>rand64()</expression>
<hierarchical>true</hierarchical>
<injective>true</injective>
<is_object_id>true</is_object_id>
<attribute/>
<structure/>
```

:Configuration fields

Tag | Description | Required

## Internal dictionaries

.ClickHouse contains a built-in feature for working with a geobase

:This allows you to

- .Use a region's ID to get its name in the desired language
- .Use a region's ID to get the ID of a city, area, federal district, country, or continent
- .Check whether a region is part of another region
- .Get a chain of parent regions

All the functions support "translocality," the ability to simultaneously use different perspectives on region "ownership. For more information, see the section "Functions for working with Yandex.Metrica dictionaries

.The internal dictionaries are disabled in the default package

To enable them, uncomment the parameters `path_to_regions_hierarchy_file` and `path_to_regions_names_files` in the .server configuration file

.The geobase is loaded from text files

Place the `regions_hierarchy*.txt` files into the `path_to_regions_hierarchy_file` directory. This configuration parameter must contain the path to the `regions_hierarchy.txt` file (the default regional hierarchy), and the other files (`regions_hierarchy_ua.txt`) must be located in the same directory

.Put the `regions_names_*.txt` files in the `path_to_regions_names_files` directory

:You can also create these files yourself. The file format is as follows

:`regions_hierarchy*.txt`: TabSeparated (no header), columns

(region ID (`UInt32`)  
(parent region ID (`UInt32`)  
region type (`UInt8`): 1 - continent, 3 - country, 4 - federal district, 5 - region, 6 - city; other types don't have values  
population (`UInt32`) — optional column

:`regions_names_*.txt`: TabSeparated (no header), columns

(region ID (`UInt32`)  
.region name (`String`) — Can't contain tabs or line feeds, even escaped ones

.A flat array is used for storing in RAM. For this reason, IDs shouldn't be more than a million

Dictionaries can be updated without restarting the server. However, the set of available dictionaries is not .updated

.For updates, the file modification times are checked. If a file has changed, the dictionary is updated

.The interval to check for changes is configured in the `builtin_dictionaries_reload_interval` parameter

Dictionary updates (other than loading at first use) do not block queries. During updates, queries use the old versions of dictionaries. If an error occurs during an update, the error is written to the server log, and queries .continue using the old version of dictionaries

We recommend periodically updating the dictionaries with the geobase. During an update, generate new files and .write them to a separate location. When everything is ready, rename them to the files used by the server

There are also functions for working with OS identifiers and Yandex.Metrica search engines, but they shouldn't be .used

## Operators

All operators are transformed to the corresponding functions at the query parsing stage, in accordance with their .precedence and associativity

Groups of operators are listed in order of priority (the higher it is in the list, the earlier the operator is connected to .(its arguments

## Access Operators

.`a[N]` Access to an element of an array; `arrayElement(a, N)` function

.`a.N` – Access to a tuple element; `tupleElement(a, N)` function

## Numeric Negation Operator

.`a` – The `negate(a)` function-

## Multiplication and Division Operators

.`a * b` – The `multiply(a, b)` function

.`a / b` – The `divide(a, b)` function

.`a % b` – The `modulo(a, b)` function

# Addition and Subtraction Operators

`.a + b` – The `plus(a, b)` function

`.a - b` – The `minus(a, b)` function

## Comparison Operators

`.a = b` – The `equals(a, b)` function

`.a == b` – The `equals(a, b)` function

`.a != b` – The `notEquals(a, b)` function

`.a <> b` – The `notEquals(a, b)` function

`.a <= b` – The `lessOrEquals(a, b)` function

`.a >= b` – The `greaterOrEquals(a, b)` function

`.a < b` – The `less(a, b)` function

`.a > b` – The `greater(a, b)` function

`.a LIKE s` – The `like(a, b)` function

`.a NOT LIKE s` – The `notLike(a, b)` function

`.a BETWEEN b AND c` – The same as `a >= b AND a <= c`

`.a NOT BETWEEN b AND c` – The same as `a < b OR a > c`

## Operators for Working With Data Sets

*.See the section [IN operators](#)*

`a IN ...` – The `in(a, b)` function

`.a NOT IN ...` – The `notIn(a, b)` function

`.a GLOBAL IN ...` – The `globalIn(a, b)` function

`.a GLOBAL NOT IN ...` – The `globalNotIn(a, b)` function

## Operator for Working With Dates and Times

`;EXTRACT(part FROM date`

Extracts a part from a given date. For example, you can retrieve a month from a given date, or a second from a `.time`

:The `part` parameter specifies which part of the date to retrieve. The following values are available

`.DAY` — The day of the month. Possible values: 1–31

`.MONTH` — The number of a month. Possible values: 1–12

`.YEAR` — The year

`.SECOND` — The second. Possible values: 0–59

`.MINUTE` — The minute. Possible values: 0–59

`.HOUR` — The hour. Possible values: 0–23

.The `part` parameter is case-insensitive

.The `date` parameter specifies the date or the time to process. Either `Date` or `DateTime` type is supported

:Examples

```

;(('SELECT EXTRACT(DAY FROM toDate('2017-06-15
;(('SELECT EXTRACT(MONTH FROM toDate('2017-06-15
;(('SELECT EXTRACT(YEAR FROM toDate('2017-06-15

```

.In the following example we create a table and insert into it a value with the **DateTime** type

```

CREATE TABLE test.Orders
)
,OrderId UInt64
,OrderName String
OrderDate DateTime
(
;ENGINE = Log

```

```

;(('INSERT INTO test.Orders VALUES (1, 'Jarlsberg Cheese', toDate('2008-10-11 13:23:44

```

```

SELECT
,toYear(OrderDate) AS OrderYear
,toMonth(OrderDate) AS OrderMonth
,toDayOfMonth(OrderDate) AS OrderDay
,toHour(OrderDate) AS OrderHour
,toMinute(OrderDate) AS OrderMinute
toSecond(OrderDate) AS OrderSecond
;FROM test.Orders

```

OrderYear	OrderMonth	OrderDay	OrderHour	OrderMinute	OrderSecond
44	23	13	11	10	2008

.You can see more examples in [tests](#)

## Logical Negation Operator

.NOT a The **not(a)** function

## Logical AND Operator

.a AND b - The **and(a, b)** function

## Logical OR Operator

.a OR b - The **or(a, b)** function

## Conditional Operator

.a ? b : c - The **if(a, b, c)** function

:Note

The conditional operator calculates the values of b and c, then checks whether condition a is met, and then returns the corresponding value. If **b** or **c** is an **arrayJoin()** function, each row will be replicated regardless of the "a" condition

## Conditional Expression

```

[CASE [x
WHEN a THEN b
[... WHEN ... THEN]
[ELSE c]
END

```

.(If **x** is specified, then **transform(x, [a, ...], [b, ...], c)** function is used. Otherwise - **multif(a, b, ..., c**

.If there is no **ELSE c** clause in the expression, the default value is **NULL**

.The **transform** function does not work with **NULL**

## Concatenation Operator

.**s1 || s2** – The **concat(s1, s2)** function

## Lambda Creation Operator

.**x -> expr** – The **lambda(x, expr)** function

:The following operators do not have a priority, since they are brackets

## Array Creation Operator

.**x1, ...]** – The **array(x1, ...)** function]

## Tuple Creation Operator

.**x1, x2, ...)** – The **tuple(x2, x2, ...)** function)

## Associativity

.(All binary operators have left associativity. For example, **1 + 2 + 3** is transformed to **plus(plus(1, 2), 3**

.Sometimes this doesn't work the way you expect. For example, **SELECT 4 > 2 > 3** will result in 0

For efficiency, the **and** and **or** functions accept any number of arguments. The corresponding chains of **AND** and **OR** operators are transformed to a single call of these functions

## Checking for **NULL**

.ClickHouse supports the **IS NULL** and **IS NOT NULL** operators

### IS NULL

:For **Nullable** type values, the **IS NULL** operator returns

.if the value is **NULL** ,**1**

.otherwise **0**

.For other values, the **IS NULL** operator always returns **0**

```
SELECT x+100 FROM t_null WHERE y IS NULL (:
```

```
SELECT x + 100
FROM t_null
(WHERE isNull(y
  -(plus(x, 100-r
  | 101          |
  |_____|
  |_____|
```

```
.rows in set. Elapsed: 0.002 sec 1
```

### IS NOT NULL

:For **Nullable** type values, the **IS NOT NULL** operator returns

.if the value is **NULL** ,**0**

.otherwise **1**

.For other values, the **IS NOT NULL** operator always returns **1**



```
SELECT * FROM t_null WHERE y IS NOT NULL (:
```

```
* SELECT
FROM t_null
(WHERE isNotNull(y
```

```
  x  y  r
  3  2
  _  _
```

```
.rows in set. Elapsed: 0.002 sec 1
```

## Syntax

There are two types of parsers in the system: the full SQL parser (a recursive descent parser), and the data format parser (a fast stream parser)

In all cases except the **INSERT** query, only the full SQL parser is used

The **INSERT** query uses both parsers

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

The **INSERT INTO t VALUES** fragment is parsed by the full parser, and the data (1, 'Hello, world'), (2, 'abc'), (3, 'def') is parsed by the fast stream parser. You can also turn on the full parser for the data by using the **input\_format\_values\_interpret\_expressions** setting. When **input\_format\_values\_interpret\_expressions = 1**, ClickHouse first tries to parse values with the fast stream parser. If it fails, ClickHouse tries to use the full parser for the data, treating it like an SQL **expression**

Data can have any format. When a query is received, the server calculates no more than **max\_query\_size** bytes of the request in RAM (by default, 1 MB), and the rest is stream parsed

This means the system doesn't have problems with large **INSERT** queries, like MySQL does

When using the **Values** format in an **INSERT** query, it may seem that data is parsed the same as expressions in a **SELECT** query, but this is not true. The **Values** format is much more limited

Next we will cover the full parser. For more information about format parsers, see the **Formats** section

## Spaces

There may be any number of space symbols between syntactical constructions (including the beginning and end of a query). Space symbols include the space, tab, line feed, CR, and form feed

## Comments

SQL-style and C-style comments are supported

SQL-style comments: from **--** to the end of the line. The space after **--** can be omitted

Comments in C-style: from **/\*** to **\*/**. These comments can be multiline. Spaces are not required here, either

## Keywords

Keywords (such as **SELECT**) are not case-sensitive. Everything else (column names, functions, and so on), in contrast to standard SQL, is case-sensitive

Keywords are not reserved (they are just parsed as keywords in the corresponding context). If you use **identifiers** the same as the keywords, enclose them into quotes. For example, the query **SELECT "FROM" FROM table\_name** is valid if the table **table\_name** has column with the name **"FROM"**

## Identifiers

Identifiers are

- Cluster, database, table, partition and column names
- Functions

- 
-

.Data types

.Expression aliases

- 
- 

.Identifiers can be quoted or non-quoted. It is recommended to use non-quoted identifiers

Non-quoted identifiers must match the regex `^[a-zA-Z_][0-9a-zA-Z_]*$` and can not be equal to **keywords**. Examples:  
`._x, _1, X_y_Z123`

If you want to use identifiers the same as keywords or you want to use other symbols in identifiers, quote it using  
double quotes or backticks, for example, `"id"`, ``id``

## Literals

.There are: numeric, string, compound and **NULL** literals

## Numeric

.A numeric literal tries to be parsed

.First as a 64-bit signed number, using the **strtoull** function

.If unsuccessful, as a 64-bit unsigned number, using the **strtoll** function

.If unsuccessful, as a floating-point number using the **strtod** function

.Otherwise, an error is returned

- 
- 
- 
- 

.The corresponding value will have the smallest type that the value fits in

.For example, 1 is parsed as **UInt8**, but 256 is parsed as **UInt16**. For more information, see **Data types**

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

## String

Only string literals in single quotes are supported. The enclosed characters can be backslash-escaped. The following escape sequences have a corresponding special value: `\b`, `\f`, `\r`, `\n`, `\t`, `\0`, `\a`, `\v`, `\xHH`. In all other cases, escape sequences in the format `\c`, where **c** is any character, are converted to **c**. This means that you can use the sequences `\and\\`. The value will have the **String** type

The minimum set of characters that you need to escape in string literals: `'` and `\`. Single quote can be escaped with the single quote, literals `'it's'` and `'it''s'` are equal

## Compound

..(Constructions are supported for arrays: **[1, 2, 3]** and tuples: **(1, 'Hello, world!', 2**

Actually, these are not literals, but expressions with the array creation operator and the tuple creation operator, respectively

.An array must consist of at least one item, and a tuple must have at least two items

Tuples have a special purpose for use in the **IN** clause of a **SELECT** query. Tuples can be obtained as the result of a query, but they can't be saved to a database (with the exception of **Memory** tables)

## NULL

.Indicates that the value is missing

.In order to store **NULL** in a table field, it must be of the **Nullable** type

Depending on the data format (input or output), **NULL** may have a different representation. For more information, see the documentation for **data formats**

There are many nuances to processing **NULL**. For example, if at least one of the arguments of a comparison operation is **NULL**, the result of this operation will also be **NULL**. The same is true for multiplication, addition, and other operations. For more information, read the documentation for each operation

In queries, you can check **NULL** using the **IS NULL** and **IS NOT NULL** operators and the related functions **isNull** and **isNotNull**

# Functions

Functions are written like an identifier with a list of arguments (possibly empty) in brackets. In contrast to standard SQL, the brackets are required, even for an empty arguments list. Example: `now()`

There are regular and aggregate functions (see the section "Aggregate functions"). Some aggregate functions can contain two lists of arguments in brackets. Example: `quantile(0.9)(x)`. These aggregate functions are called "parametric" functions, and the arguments in the first list are called "parameters". The syntax of aggregate functions without parameters is the same as for regular functions

## Operators

Operators are converted to their corresponding functions during query parsing, taking their priority and associativity into account

(For example, the expression `1 + 2 * 3 + 4` is transformed to `plus(plus(1, multiply(2, 3)), 4)`

## Data Types and Database Table Engines

Data types and table engines in the `CREATE` query are written the same way as identifiers or functions. In other words, they may or may not contain an arguments list in brackets. For more information, see the sections "Data types," "Table engines," and "CREATE

## Expression Aliases

An alias is a user-defined name for an expression in a query

```
expr AS alias
```

**AS** — The keyword for defining aliases. You can define the alias for a table name or a column name in a `SELECT` clause without using the **AS** keyword

For example, `SELECT table_name_alias.column_name FROM table_name table_name_alias`

In the `CAST` function, the **AS** keyword has another meaning. See the description of the function

**expr** — Any expression supported by ClickHouse

For example, `SELECT column_name * 2 AS double FROM some_table`

**alias** — Name for **expr**. Aliases should comply with the **identifiers** syntax

For example, `SELECT "table t".column_name FROM table_name AS "table t"`

## Notes on Usage

Aliases are global for a query or subquery and you can define an alias in any part of a query for any expression.

For example, `SELECT (1 AS n) + 2, n`

Aliases are not visible in subqueries and between subqueries. For example, while executing the query `SELECT (SELECT sum(b.a) + num FROM b) - a.a AS num FROM a` ClickHouse generates the exception `Unknown identifier: num`

If an alias is defined for the result columns in the `SELECT` clause of a subquery, these columns are visible in the (outer query. For example, `SELECT n + m FROM (SELECT 1 AS n, 2 AS m`

Be careful with aliases that are the same as column or table names. Let's consider the following example

```
CREATE TABLE t
(
  a Int
  b Int
  (
  )ENGINE = TinyLog
```

```
SELECT
,(argMax(a, b
sum(b) AS b
FROM t
```

:(Received exception from server (version 18.14.17

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

In this example, we declared table `t` with column `b`. Then, when selecting data, we defined the `sum(b) AS b` alias. As aliases are global, ClickHouse substituted the literal `b` in the expression `argMax(a, b)` with the expression `sum(b)`. This substitution caused the exception

## Asterisk

."In a `SELECT` query, an asterisk can replace the expression. For more information, see the section "SELECT

## Expressions

An expression is a function, identifier, literal, application of an operator, expression in brackets, subquery, or  
.asterisk. It can also contain an alias

.A list of expressions is one or more expressions separated by commas

.Functions and operators, in turn, can have expressions as arguments

## Settings

.There are multiple ways to make all the settings described below

.Settings are configured in layers, so each subsequent layer redefines the previous settings

:Ways to configure settings, in order of priority

.Settings in the `users.xml` server configuration file

.<Set in the element `<profiles`

.Session settings

.Send `SET setting=value` from the ClickHouse console client in interactive mode

Similarly, you can use ClickHouse sessions in the HTTP protocol. To do this, you need to specify the `session_id`

.HTTP parameter

.Query settings

When starting the ClickHouse console client in non-interactive mode, set the startup parameter --

.`setting=value`

.(...When using the HTTP API, pass CGI parameters (`URL?setting_1=value&setting_2=value`

.Settings that can only be made in the server config file are not covered in this section

## Requirements

### CPU

For installation from prebuilt deb packages, use a CPU with x86\_64 architecture and support for SSE 4.2 instructions. To run ClickHouse with processors that do not support SSE 4.2 or have AArch64 or PowerPC64LE architecture, you should build ClickHouse from sources

ClickHouse implements parallel data processing and uses all the hardware resources available. When choosing a processor, take into account that ClickHouse works more efficiently at configurations with a large number of cores but a lower clock rate than at configurations with fewer cores and a higher clock rate. For example, 16 cores with .2600 MHz is preferable to 8 cores with 3600 MHz

Use of **Turbo Boost** and **hyper-threading** technologies is recommended. It significantly improves performance with a typical load

## RAM

We recommend to use a minimum of 4GB of RAM in order to perform non-trivial queries. The ClickHouse server can run with a much smaller amount of RAM, but it requires memory for processing queries

The required volume of RAM depends on

- The complexity of queries
- The amount of data that is processed in queries

To calculate the required volume of RAM, you should estimate the size of temporary data for **GROUP BY**, **DISTINCT**, **JOIN** and other operations you use

ClickHouse can use external memory for temporary data. See **GROUP BY in External Memory** for details

## Swap File

Disable the swap file for production environments

## Storage Subsystem

You need to have 2GB of free disk space to install ClickHouse

The volume of storage required for your data should be calculated separately. Assessment should include

- Estimation of the data volume

You can take a sample of the data and get the average size of a row from it. Then multiply the value by the number of rows you plan to store

- The data compression coefficient

To estimate the data compression coefficient, load a sample of your data into ClickHouse and compare the actual size of the data with the size of the table stored. For example, clickstream data is usually compressed by 6-10 times

To calculate the final volume of data to be stored, apply the compression coefficient to the estimated data volume. If you plan to store data in several replicas, then multiply the estimated volume by the number of replicas

## Network

If possible, use networks of 10G or higher class

The network bandwidth is critical for processing distributed queries with a large amount of intermediate data. In addition, network speed affects replication processes

## Software

ClickHouse is developed for the Linux family of operating systems. The recommended Linux distribution is Ubuntu. The **tzdata** package should be installed in the system

ClickHouse can also work in other operating system families. See details in the **Getting started** section of the documentation

## Monitoring

You can monitor

- Utilization of hardware resources
- ClickHouse server metrics

# Resource Utilization

.ClickHouse does not monitor the state of hardware resources by itself

:It is highly recommended to set up monitoring for

.Load and temperature on processors

.You can use [dmesg](#), [turbostat](#) or other instruments

.Utilization of storage system, RAM and network

## ClickHouse Server Metrics

.ClickHouse server has embedded instruments for self-state monitoring

.To track server events use server logs. See the [logger](#) section of the configuration file

:ClickHouse collects

.Different metrics of how the server uses computational resources

.Common statistics on query processing

.You can find metrics in the [system.metrics](#), [system.events](#), and [system.asynchronous\\_metrics](#) tables

You can configure ClickHouse to export metrics to [Graphite](#). See the [Graphite section](#) in the ClickHouse server configuration file. Before configuring export of metrics, you should set up Graphite by following their official guide <https://graphite.readthedocs.io/en/latest/install.html>

Additionally, you can monitor server availability through the HTTP API. Send the [HTTP GET](#) request to [/](#). If the server is available, it responds with [200 OK](#)

To monitor servers in a cluster configuration, you should set the [max\\_replica\\_delay\\_for\\_distributed\\_queries](#) parameter and use the HTTP resource [/replicas-delay](#). A request to [/replicas-delay](#) returns [200 OK](#) if the replica is available and is not delayed behind the other replicas. If a replica is delayed, it returns information about the gap

## Troubleshooting

[Installation](#)

[Connecting to the server](#)

[Query processing](#)

[Efficiency of query processing](#)

## Installation

### You Cannot Get Deb Packages from ClickHouse Repository With apt-get

.Check firewall settings

If you cannot access the repository for any reason, download packages as described in the [Getting started](#) article and install them manually using the [sudo dpkg -i <packages>](#) command. You will also need the [tzdata](#) package

## Connecting to the Server

:Possible issues

.The server is not running

.Unexpected or wrong configuration parameters

## Server Is Not Running

**Check if server is running**

:Command

```
sudo service clickhouse-server status
```

:If the server is not running, start it with the command

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

## Check logs

.The main log of `clickhouse-server` is in `/var/log/clickhouse-server/clickhouse-server.log` by default

:If the server started successfully, you should see the strings

.Information> Application: starting up. — Server started>

.Information> Application: Ready for connections. — Server is running and ready for connections>

If `clickhouse-server` start failed with a configuration error, you should see the `<Error>` string with an error description.

:For example

```
Error> ExternalDictionaries: Failed reloading 'event2id' external dictionary:> {} [ 45 ] 15:23:25.549505 2019.01.11
Poco::Exception. Code: 1000, e.code() = 111, e.displayText() = Connection refused, e.what() = Connection refused
```

:If you don't see an error at the end of the file, look through the entire file starting from the string

```
.Information> Application: starting up>
```

:If you try to start a second instance of `clickhouse-server` on the server, you see the following log

```
Information> : Starting ClickHouse 19.1.0 with revision 54413> {} [ 1 ] 15:25:11.151730 2019.01.11
Information> Application: starting up> {} [ 1 ] 15:25:11.154578 2019.01.11
:Information> StatusFile: Status file ./status already exists - unclean restart. Contents> {} [ 1 ] 15:25:11.156361 2019.01.11
PID: 8510
Started at: 2019-01-11 15:24:23
Revision: 54413

Error> Application: DB::Exception: Cannot lock file ./status. Another server instance in same> {} [ 1 ] 15:25:11.156673 2019.01.11
.directory is already running
Information> Application: shutting down> {} [ 1 ] 15:25:11.156682 2019.01.11
Debug> Application: Uninitializing subsystem: Logging Subsystem> {} [ 1 ] 15:25:11.156686 2019.01.11
Information> BaseDaemon: Stop SignalListener thread> {} [ 2 ] 15:25:11.156716 2019.01.11
```

## See system.d logs

If you don't find any useful information in `clickhouse-server` logs or there aren't any logs, you can view `system.d` logs using the command

```
sudo journalctl -u clickhouse-server
```

## Start clickhouse-server in interactive mode

```
sudo -u clickhouse /usr/bin/clickhouse-server --config-file /etc/clickhouse-server/config.xml
```

This command starts the server as an interactive app with standard parameters of the autostart script. In this mode `clickhouse-server` prints all the event messages in the console

## Configuration Parameters

:Check

.Docker settings

.If you run ClickHouse in Docker in an IPv6 network, make sure that `network=host` is set

- .Endpoint settings
- .Check `listen_host` and `tcp_port` settings
- .ClickHouse server accepts localhost connections only by default
- .HTTP protocol settings
- .Check protocol settings for the HTTP API
- .Secure connection settings
- :Check
  - .The `tcp_port_secure` setting
  - .Settings for `SSL certificates`
- .Use proper parameters while connecting. For example, use the `port_secure` parameter with `clickhouse_client`
- .User settings
  - .You might be using the wrong user name or password

## Query Processing

If ClickHouse is not able to process the query, it sends an error description to the client. In the `clickhouse-client` you get a description of the error in the console. If you are using the HTTP interface, ClickHouse sends the error :description in the response body. For example

```
"curl 'http://localhost:8123/' --data-binary "SELECT a $
Code: 47, e.displayText() = DB::Exception: Unknown identifier: a. Note that there are no tables (FROM clause) in your query,
context: required_names: 'a' source_tables: table_aliases: private_aliases: column_aliases: public_columns: 'a' masked_columns:
array_join_columns: source_columns: , e.what() = DB::Exception
```

If you start `clickhouse-client` with the `stack-trace` parameter, ClickHouse returns the server stack trace with the .description of an error

You might see a message about a broken connection. In this case, you can repeat the query. If the connection .breaks every time you perform the query, check the server logs for errors

## Efficiency of Query Processing

If you see that ClickHouse is working too slowly, you need to profile the load on the server resources and network .for your queries

You can use the `clickhouse-benchmark` utility to profile queries. It shows the number of queries processed per .second, the number of rows processed per second, and percentiles of query processing times

## Usage Recommendations

### CPU Scaling Governor

Always use the `performance` scaling governor. The `on-demand` scaling governor works much worse with constantly .high demand

```
echo 'performance' | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

### CPU Limitations

- .Processors can overheat. Use `dmesg` to see if the CPU's clock rate was limited due to overheating
- .The restriction can also be set externally at the datacenter level. You can use `turbostat` to monitor it under a load

### RAM



.For small amounts of data (up to ~200 GB compressed), it is best to use as much memory as the volume of data  
For large amounts of data and when processing interactive (online) queries, you should use a reasonable amount  
.of RAM (128 GB or more) so the hot data subset will fit in the cache of pages  
Even for data volumes of ~50 TB per server, using 128 GB of RAM significantly improves query performance  
.compared to 64 GB

Do not disable overcommit. The value `cat /proc/sys/vm/overcommit_memory` should be 0 or 1. Run

```
echo 0 | sudo tee /proc/sys/vm/overcommit_memory
```

## Huge Pages

Always disable transparent huge pages. It interferes with memory allocators, which leads to significant  
.performance degradation

```
echo 'never' | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
```

.Use `perf top` to watch the time spent in the kernel for memory management  
.Permanent huge pages also do not need to be allocated

## Storage Subsystem

.If your budget allows you to use SSD, use SSD  
.If not, use HDD. SATA HDDs 7200 RPM will do

Give preference to a lot of servers with local hard drives over a smaller number of servers with attached disk  
.shelves  
.But for storing archives with rare queries, shelves will work

## RAID

.When using HDD, you can combine their RAID-10, RAID-5, RAID-6 or RAID-50  
.For Linux, software RAID is better (with `mdadm`). We don't recommend using LVM  
.When creating RAID-10, select the `far` layout  
.If your budget allows, choose RAID-10

.If you have more than 4 disks, use RAID-6 (preferred) or RAID-50, instead of RAID-5  
When using RAID-5, RAID-6 or RAID-50, always increase `stripe_cache_size`, since the default value is usually not  
.the best choice

```
echo 4096 | sudo tee /sys/block/md2/md/stripe_cache_size
```

Calculate the exact number from the number of devices and the block size, using the formula:  $2 * \text{num\_devices} * \text{chunk\_size\_in\_bytes} / 4096$

.A block size of 1024 KB is sufficient for all RAID configurations  
.Never set the block size too small or too large

.You can use RAID-0 on SSD  
.Regardless of RAID use, always use replication for data security

Enable NCQ with a long queue. For HDD, choose the CFQ scheduler, and for SSD, choose noop. Don't reduce the  
. 'readahead' setting  
.For HDD, enable the write cache

## File System

.Ext4 is the most reliable option. Set the mount options `noatime, nobarrier`  
.XFS is also suitable, but it hasn't been as thoroughly tested with ClickHouse  
.Most other file systems should also work fine. File systems with delayed allocation work better

## Linux Kernel

.Don't use an outdated Linux kernel

## Network

.If you are using IPv6, increase the size of the route cache

.The Linux kernel prior to 3.2 had a multitude of problems with IPv6 implementation

Use at least a 10 GB network, if possible. 1 Gb will also work, but it will be much worse for patching replicas with tens of terabytes of data, or for processing distributed queries with a large amount of intermediate data

## ZooKeeper

You are probably already using ZooKeeper for other purposes. You can use the same installation of ZooKeeper, if it isn't already overloaded

It's best to use a fresh version of ZooKeeper – 3.4.9 or later. The version in stable Linux distributions may be outdated

You should never use manually written scripts to transfer data between different ZooKeeper clusters, because the result will be incorrect for sequential nodes. Never use the "zkcopy" utility for the same reason:

<https://github.com/ksprojects/zkcopy/issues/15>

If you want to divide an existing ZooKeeper cluster into two, the correct way is to increase the number of its replicas and then reconfigure it as two independent clusters

Do not run ZooKeeper on the same servers as ClickHouse. Because ZooKeeper is very sensitive for latency and ClickHouse may utilize all available system resources

:With the default settings, ZooKeeper is a time bomb

The ZooKeeper server won't delete files from old snapshots and logs when using the default configuration (see autopurge), and this is the responsibility of the operator

.This bomb must be defused

The ZooKeeper (3.5.1) configuration below is used in the Yandex.Metrica production environment as of May 20, 2017

:zoo.cfg

<http://hadoop.apache.org/zookeeper/docs/current/zookeeperAdmin.html> ##

The number of milliseconds of each tick ##

tickTime=2000

The number of ticks that the initial ##

synchronization phase can take ##

initLimit=30000

The number of ticks that can pass between ##

sending a request and getting an acknowledgement ##

syncLimit=10

maxClientCnxns=2000

maxSessionTimeout=60000000

.the directory where the snapshot is stored ##

dataDir=/opt/zookeeper/{ { cluster['name'] } }/data

Place the dataLogDir to a separate physical disc for better performance ##

dataLogDir=/opt/zookeeper/{ { cluster['name'] } }/logs

autopurge.snapRetainCount=10

autopurge.purgeInterval=1

To avoid seeks ZooKeeper allocates space in the transaction log file in ##

blocks of preAllocSize kilobytes. The default block size is 64M. One reason ##

for changing the size of the blocks is to reduce the block size if snapshots ##

.(are taken more often. (Also, see snapCount ##

preAllocSize=131072

,Clients can submit requests faster than ZooKeeper can process them ##

especially if there are a lot of clients. To prevent ZooKeeper from running ##

out of memory due to queued requests, ZooKeeper will throttle clients so that ##

there is no more than globalOutstandingLimit outstanding requests in the ##

system. The default limit is 1,000.ZooKeeper logs transactions to a ##

transaction log. After snapCount transactions are written to a log file a ##

snapshot is started and a new transaction log file is started. The default ##

.snapCount is 10,000 ##

snapCount=3000000

If this option is defined, requests will be will logged to a trace file named ##

.traceFile.year.month.day ##

=traceFile##

Leader accepts client connections. Default value is "yes". The leader machine ##

coordinates updates. For higher update throughput at the slight expense of ##

read throughput the leader can be configured to not accept clients and focus ##

.on coordination ##

leaderServes=yes

standaloneEnabled=false

dynamicConfigFile=/etc/zookeeper-{ { cluster['name'] } }/conf/zoo.cfg.dynamic

:Java version

(Java(TM) SE Runtime Environment (build 1.8.0\_25-b17

(Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode

:JVM parameters

```

{{ ['NAME=zookeeper-{{ cluster['name
ZOOCFGDIR=/etc/$NAME/conf

TODO this is really ugly ##
?How to find out, which jars are needed ##
seems, that log4j requires the log4j.properties file to be in the classpath ##
CLASSPATH="$ZOOCFGDIR:/usr/build/classes:/usr/build/lib/*.jar:/usr/share/zookeeper/zookeeper-3.5.1-
metrika.jar:/usr/share/zookeeper/slf4j-log4j12-1.7.5.jar:/usr/share/zookeeper/slf4j-api-1.7.5.jar:/usr/share/zookeeper/servlet-api-
2.5-20081211.jar:/usr/share/zookeeper/netty-3.7.0.Final.jar:/usr/share/zookeeper/log4j-1.2.16.jar:/usr/share/zookeeper/jline-
2.11.jar:/usr/share/zookeeper/jetty-util-6.1.26.jar:/usr/share/zookeeper/jetty-
6.1.26.jar:/usr/share/zookeeper/javacc.jar:/usr/share/zookeeper/jackson-mapper-asl-1.9.11.jar:/usr/share/zookeeper/jackson-core-
"asl-1.9.11.jar:/usr/share/zookeeper/commons-cli-1.2.jar:/usr/src/java/lib/*.jar:/usr/etc/zookeeper

"ZOOCFG="$ZOOCFGDIR/zoo.cfg
ZOO_LOG_DIR=/var/log/$NAME
USER=zookeeper
GROUP=zookeeper
PIDDIR=/var/run/$NAME
PIDFILE=$PIDDIR/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME
JAVA=/usr/bin/java
"ZOOMAIN="org.apache.zookeeper.server.quorum.QuorumPeerMain
"ZOO_LOG4J_PROP="INFO,ROLLINGFILE
JMXLOCALONLY=false
\ {{ ('JAVA_OPTS="-Xms{{ cluster.get('xms','128M
\ {{ ('Xmx{{ cluster.get('xmx','1G-
\ Xloggc:/var/log/$NAME/zookeeper-gc.log-
\ XX:+UseGCLogFileRotation-
\ XX:NumberOfGCLogFiles=16-
\ XX:GCLogFileSize=16M-
\ verbose:gc-
\ XX:+PrintGCTimeStamps-
\ XX:+PrintGCDateStamps-
XX:+PrintGCDetails-
\ XX:+PrintTenuringDistribution-
\ XX:+PrintGCApplicationStoppedTime-
\ XX:+PrintGCApplicationConcurrentTime-
\ XX:+PrintSafepointStatistics-
\ XX:+UseParNewGC-
\ XX:+UseConcMarkSweepGC-
"XX:+CMSParallelRemarkEnabled-

```

:Salt init

```
"description "zookeeper-{{ cluster['name'] }} centralized coordination service
```

```
[start on runlevel [2345
```

```
[stop on runlevel [!2345
```

```
respawn
```

```
limit nofile 8192 8192
```

```
pre-start script
```

```
r "/etc/zookeeper-{{ cluster['name'] }}/conf/environment" ] || exit 0- ]
```

```
etc/zookeeper-{{ cluster['name'] }}/conf/environment/ .
```

```
d $ZOO_LOG_DIR ] || mkdir -p $ZOO_LOG_DIR- ]
```

```
chown $USER:$GROUP $ZOO_LOG_DIR
```

```
end script
```

```
script
```

```
etc/zookeeper-{{ cluster['name'] }}/conf/environment/ .
```

```
r /etc/default/zookeeper ] && . /etc/default/zookeeper- ]
```

```
if [ -z "$JMXDISABLE" ]; then
```

```
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote -
```

```
"Dcom.sun.management.jmxremote.local.only=$JMXLOCALONLY
```

```
fi
```

```
\ {{ ['exec start-stop-daemon --start -c $USER --exec $JAVA --name zookeeper-{{ cluster['name
```

```
\ {cp $CLASSPATH $JAVA_OPTS -Dzookeeper.log.dir=${ZOO_LOG_DIR- --
```

```
Dzookeeper.root.logger=${ZOO_LOG4J_PROP} $ZOOMAIN $ZOOCFG-
```

```
end script
```

## ClickHouse Update

:If ClickHouse was installed from deb packages, execute the following commands on the server

```
sudo apt-get update
```

```
sudo apt-get install clickhouse-client clickhouse-server
```

```
sudo service clickhouse-server restart
```

If you installed ClickHouse using something other than the recommended deb packages, use the appropriate .update method

ClickHouse does not support a distributed update. The operation should be performed consecutively on each separate server. Do not update all the servers on a cluster simultaneously, or the cluster will be unavailable for .some time

## Access Rights

.Users and access rights are set up in the user config. This is usually [users.xml](#)

:Users are recorded in the [users](#) section. Here is a fragment of the [users.xml](#) file

```

<-- .Users and ACL --!>
<users>
<-- .If the user name is not specified, the 'default' user is used --!>
<default>
.(Password could be specified in plaintext or in SHA256 (in hex format --!>

.If you want to specify password in plaintext (not recommended), place it in 'password' element
.<Example: <password>qwerty</password>
.Password could be empty

.If you want to specify SHA256, place it in 'password_sha256_hex' element
Example:
<<password_sha256_hex>65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5</password_sha256_hex>

:How to generate decent password
:execute: PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | sha256sum | tr -
'-' d
.In first line will be password and in second - corresponding SHA256
<--
<password></password>

.A list of networks that access is allowed from --!>
:Each list item has one of the following forms
.ip> The IP address or subnet mask. For example: 198.51.100.0/24 or 2001:DB8::/32>
.host> Host name. For example: example01. A DNS query is made for verification, and all addresses obtained are>
.compared with the address of the customer
.$host_regexp> Regular expression for host names. For example, ^example\d\d\d\d\d\.yandex\.ru>
.To check it, a DNS PTR request is made for the client's address and a regular expression is applied to the result
Then another DNS query is made for the result of the PTR query, and all received address are compared to the client
.address
.$We strongly recommend that the regex ends with \.yandex\.ru

:If you are installing ClickHouse yourself, specify here
<networks>
<ip>::/0</ip>
<networks/>
<--
</ "networks incl="networks>

<-- .Settings profile for the user --!>
<profile>default</profile>

<-- .Quota for the user --!>
<quota>default</quota>
<default/>

<-- .For requests from the Yandex.Metrica user interface via the API for data on specific counters --!>
<web>
<password></password>
</ "networks incl="networks>
<profile>web</profile>
<quota>default</quota>
<allow_databases>
<database>test</database>
<allow_databases/>
<web/>
<users/>

```

.You can see a declaration from two users: **default** and **web**. We added the **web** user separately

The **default** user is chosen in cases when the username is not passed. The **default** user is also used for distributed query processing, if the configuration of the server or cluster doesn't specify the **user** and **password** (see the section .(on the **Distributed** engine

The user that is used for exchanging information between servers combined in a cluster must not have substantial restrictions or quotas – otherwise, distributed queries will fail

The password is specified in clear text (not recommended) or in SHA-256. The hash isn't salted. In this regard, you should not consider these passwords as providing security against potential malicious attacks. Rather, they are necessary for protection from employees

A list of networks is specified that access is allowed from. In this example, the list of networks for both users is loaded from a separate file (`/etc/metrika.xml`) containing the `networks` substitution. Here is a fragment of it

```
<yandex>
...
<networks>
<ip>::/64</ip>
<ip>203.0.113.0/24</ip>
<ip>2001:DB8::/32</ip>
...
</networks>
</yandex>
```

You could define this list of networks directly in `users.xml`, or in a file in the `users.d` directory (for more information, see the section "Configuration files")

The config includes comments explaining how to open access from everywhere

For use in production, only specify `ip` elements (IP addresses and their masks), since using `host` and `host_regex` might cause extra latency

Next the user settings profile is specified (see the section "Settings profiles"). You can specify the default profile, `default`. The profile can have any name. You can specify the same profile for different users. The most important thing you can write in the settings profile is `readonly=1`, which ensures read-only access

Then specify the quota to be used (see the section "Quotas"). You can specify the default quota: `default`. It is set in the config by default to only count resource usage, without restricting it. The quota can have any name. You can specify the same quota for different users – in this case, resource usage is calculated for each user individually

In the optional `<allow_databases>` section, you can also specify a list of databases that the user can access. By default, all databases are available to the user. You can specify the `default` database. In this case, the user will receive access to the database by default

(Access to the `system` database is always allowed (since this database is used for processing queries

The user can get a list of all databases and tables in them by using `SHOW` queries or system tables, even if access to individual databases isn't allowed

Database access is not related to the `readonly` setting. You can't grant full access to one database and `readonly` access to another one

## Data Backup

While `replication` provides protection from hardware failures, it does not protect against human errors: accidental deletion of data, deletion of the wrong table or a table on the wrong cluster, and software bugs that result in incorrect data processing or data corruption. In many cases mistakes like these will affect all replicas. ClickHouse has built-in safeguards to prevent some types of mistakes — for example, by default **you can't just drop tables with a MergeTree-like engine containing more than 50 Gb of data**. However, these safeguards don't cover all possible cases and can be circumvented

In order to effectively mitigate possible human errors, you should carefully prepare a strategy for backing up and restoring your data **in advance**

Each company has different resources available and business requirements, so there's no universal solution for ClickHouse backups and restores that will fit every situation. What works for one gigabyte of data likely won't work for tens of petabytes. There are a variety of possible approaches with their own pros and cons, which will be discussed below. It is a good idea to use several approaches instead of just one in order to compensate for their various shortcomings

#### Note

Keep in mind that if you backed something up and never tried to restore it, chances are that restore will not work properly when you actually need it (or at least it will take longer than business can tolerate). So whatever backup approach you choose, make sure to automate the restore process as well, and practice it on a spare ClickHouse cluster regularly

## Duplicating Source Data Somewhere Else

Often data that is ingested into ClickHouse is delivered through some sort of persistent queue, such as [Apache Kafka](#). In this case it is possible to configure an additional set of subscribers that will read the same data stream while it is being written to ClickHouse and store it in cold storage somewhere. Most companies already have some default recommended cold storage, which could be an object store or a distributed filesystem like [HDFS](#)

## Filesystem Snapshots

Some local filesystems provide snapshot functionality (for example, [ZFS](#)), but they might not be the best choice for serving live queries. A possible solution is to create additional replicas with this kind of filesystem and exclude them from the [Distributed](#) tables that are used for [SELECT](#) queries. Snapshots on such replicas will be out of reach of any queries that modify data. As a bonus, these replicas might have special hardware configurations with more disks attached per server, which would be cost-effective

## clickhouse-copier

[clickhouse-copier](#) is a versatile tool that was initially created to re-shard petabyte-sized tables. It can also be used for backup and restore purposes because it reliably copies data between ClickHouse tables and clusters

For smaller volumes of data, a simple [INSERT INTO ... SELECT ...](#) to remote tables might work as well

## Manipulations with Parts

ClickHouse allows using the [ALTER TABLE ... FREEZE PARTITION ...](#) query to create a local copy of table partitions. This is implemented using hardlinks to the [/var/lib/clickhouse/shadow/](#) folder, so it usually does not consume extra disk space for old data. The created copies of files are not handled by ClickHouse server, so you can just leave them there: you will have a simple backup that doesn't require any additional external system, but it will still be prone to hardware issues. For this reason, it's better to remotely copy them to another location and then remove the local copies. Distributed filesystems and object stores are still a good options for this, but normal attached file servers with a large enough capacity might work as well (in this case the transfer will occur via the network filesystem or maybe [rsync](#))

For more information about queries related to partition manipulations, see the [ALTER documentation](#)

A third-party tool is available to automate this approach: [clickhouse-backup](#)

## Configuration Files

The main server config file is [config.xml](#). It resides in the [/etc/clickhouse-server/](#) directory

Individual settings can be overridden in the [\\*.xml](#) and [\\*.conf](#) files in the [config.d](#) directory next to the config file

The [replace](#) or [remove](#) attributes can be specified for the elements of these config files

If neither is specified, it combines the contents of elements recursively, replacing values of duplicate children

If [replace](#) is specified, it replaces the entire element with the specified one



If **remove** is specified, it deletes the element

The config can also define "substitutions". If an element has the **incl** attribute, the corresponding substitution from the file will be used as the value. By default, the path to the file with substitutions is **/etc/metrika.xml**. This can be changed in the **include\_from** element in the server config. The substitution values are specified in **/yandex/substitution\_name** elements in this file. If a substitution specified in **incl** does not exist, it is recorded in the log. To prevent ClickHouse from logging missing substitutions, specify the **optional="true"** attribute (for example, **.(settings for macros**

Substitutions can also be performed from ZooKeeper. To do this, specify the attribute **from\_zk = "/path/to/node"**. The element value is replaced with the contents of the node at **/path/to/node** in ZooKeeper. You can also put an entire XML subtree on the ZooKeeper node and it will be fully inserted into the source element

The **config.xml** file can specify a separate config with user settings, profiles, and quotas. The relative path to this config is set in the 'users\_config' element. By default, it is **users.xml**. If **users\_config** is omitted, the user settings, profiles, and quotas are specified directly in **config.xml**

In addition, **users\_config** may have overrides in files from the **users\_config.d** directory (for example, **users.d**) and substitutions. For example, you can have separate config file for each user like this

```
cat /etc/clickhouse-server/users.d/alice.xml $
<yandex>
<users>
<alice>
<profile>analytics</profile>
<networks>
<ip>::/0</ip>
<networks/>
<password_sha256_hex>...</password_sha256_hex>
<quota>analytics</quota>
</alice>
</users>
</yandex>
```

For each config file, the server also generates **file-preprocessed.xml** files when starting. These files contain all the completed substitutions and overrides, and they are intended for informational use. If ZooKeeper substitutions were used in the config files but ZooKeeper is not available on the server start, the server loads the configuration from the preprocessed file

The server tracks changes in config files, as well as files and ZooKeeper nodes that were used when performing substitutions and overrides, and reloads the settings for users and clusters on the fly. This means that you can modify the cluster, users, and their settings without restarting the server

## Quotas

Quotas allow you to limit resource usage over a period of time, or simply track the use of resources

Quotas are set up in the user config. This is usually 'users.xml

The system also has a feature for limiting the complexity of a single query. See the section "Restrictions on query complexity

In contrast to query complexity restrictions, quotas

- Place restrictions on a set of queries that can be run over a period of time, instead of limiting a single query
- Account for resources spent on all remote servers for distributed query processing

Let's look at the section of the 'users.xml' file that defines quotas

```

<-- Quotas --!>
<quotas>
<-- .Quota name --!>
<default>
<-- .Restrictions for a time period. You can set many intervals with different restrictions --!>
<interval>
<-- .Length of the interval --!>
<duration>3600</duration>

<-- .Unlimited. Just collect data for the specified time interval --!>
<queries>0</queries>
<errors>0</errors>
<result_rows>0</result_rows>
<read_rows>0</read_rows>
<execution_time>0</execution_time>
<interval/>
<default/>

```

.By default, the quota just tracks resource consumption for each hour, without limiting usage  
 .The resource consumption calculated for each interval is output to the server log after each request

```

<statbox>
<-- .Restrictions for a time period. You can set many intervals with different restrictions --!>
<interval>
<-- .Length of the interval --!>
<duration>3600</duration>

<queries>1000</queries>
<errors>100</errors>
<result_rows>1000000000</result_rows>
<read_rows>100000000000</read_rows>
<execution_time>900</execution_time>
<interval/>

<interval>
<duration>86400</duration>

<queries>10000</queries>
<errors>1000</errors>
<result_rows>5000000000</result_rows>
<read_rows>500000000000</read_rows>
<execution_time>7200</execution_time>
<interval/>
<statbox/>

```

For the 'statbox' quota, restrictions are set for every hour and for every 24 hours (86,400 seconds). The time interval is counted starting from an implementation-defined fixed moment in time. In other words, the 24-hour .interval doesn't necessarily begin at midnight

.When the interval ends, all collected values are cleared. For the next hour, the quota calculation starts over

:Here are the amounts that can be restricted

.queries – The total number of requests

.errors – The number of queries that threw an exception

.result\_rows – The total number of rows given as the result

.read\_rows – The total number of source rows read from tables for running the query, on all remote servers

.(execution\_time – The total query execution time, in seconds (wall time

If the limit is exceeded for at least one time interval, an exception is thrown with a text about which restriction  
. (was exceeded, for which interval, and when the new interval begins (when queries can be sent again

Quotas can use the "quota key" feature in order to report on resources for multiple keys independently. Here is an  
:example of this

```
<-- .For the global reports designer --!>
<web_global>
, keyed - The quota_key "key" is passed in the query parameter --!>
.and the quota is tracked separately for each key value
.For example, you can pass a Yandex.Metrica username as the key
.so the quota will be counted separately for each username
.Using keys makes sense only if quota_key is transmitted by the program, not by a user

.You can also write <keyed_by_ip /> so the IP address is used as the quota key
(.But keep in mind that users can change the IPv6 address fairly easily)
<--
</ keyed>
```

."The quota is assigned to users in the 'users' section of the config. See the section "Access rights

For distributed query processing, the accumulated amounts are stored on the requestor server. So if the user goes  
."to another server, the quota there will "start over

.When the server is restarted, quotas are reset

## System tables

System tables are used for implementing part of the system's functionality, and for providing access to  
. information about how the system is working  
(.You can't delete a system table (but you can perform DETACH  
System tables don't have files with data on the disk or files with metadata. The server creates all the system  
. tables when it starts  
.System tables are read-only  
.They are located in the 'system' database

## system.asynchronous\_metrics

.Contains metrics which are calculated periodically in background. For example, the amount of RAM in use

:Columns

- .metric (String) — Metric's name
- .value (Float64) — Metric's value

### Example

SELECT \* FROM system.asynchronous\_metrics LIMIT 10

metric	value
jemalloc.background_thread.run_interval	0
jemalloc.background_thread.num_runs	0
jemalloc.background_thread.num_threads	0
jemalloc.retained	422551552
jemalloc.mapped	1682989056
jemalloc.resident	1656446976
jemalloc.metadata_thp	0
jemalloc.metadata	10226856
UncompressedCacheCells	0
MarkCacheFiles	0

See Also

- [.Monitoring](#) — Base concepts of ClickHouse monitoring
- [.system.metrics](#) — Contains instantly calculated metrics
- [.system.events](#) — Contains a number of happened events

## system.clusters

.Contains information about clusters available in the config file and the servers in them  
:Columns

```
.cluster String      — The cluster name
.shard_num UInt32    — The shard number in the cluster, starting from 1
.shard_weight UInt32 — The relative weight of the shard when writing data
.replica_num UInt32  — The replica number in the shard, starting from 1
.host_name String    — The host name, as specified in the config
.String host_address — The host IP address obtained from DNS
.port UInt16         — The port to use for connecting to the server
.user String         — The name of the user for connecting to the server
```

## system.columns

.Contains information about the columns in all the tables

.You can use this table to get information similar to the [DESCRIBE TABLE](#) query, but for multiple tables at once

The [system.columns](#) table contains the following columns (the type of the corresponding column is shown in :[brackets

- [.database](#) (String) — Database name
- [.table](#) (String) — Table name
- [.name](#) (String) — Column name
- [.type](#) (String) — Column type
- [default\\_kind](#) (String) — Expression type ([DEFAULT](#), [MATERIALIZED](#), [ALIAS](#)) for the default value, or an empty string if it is not defined
- [.default\\_expression](#) (String) — Expression for the default value, or an empty string if it is not defined
- [.data\\_compressed\\_bytes](#) (UInt64) — The size of compressed data, in bytes
- [.data\\_uncompressed\\_bytes](#) (UInt64) — The size of decompressed data, in bytes
- [.marks\\_bytes](#) (UInt64) — The size of marks, in bytes
- [.comment](#) (String) — The comment about column, or an empty string if it is not defined
- [.is\\_in\\_partition\\_key](#) (UInt8) — Flag that indicates whether the column is in partition expression
- [.is\\_in\\_sorting\\_key](#) (UInt8) — Flag that indicates whether the column is in sorting key expression
- [.is\\_in\\_primary\\_key](#) (UInt8) — Flag that indicates whether the column is in primary key expression
- [.is\\_in\\_sampling\\_key](#) (UInt8) — Flag that indicates whether the column is in sampling key expression

## system.databases

.This table contains a single String column called 'name' – the name of a database  
.Each database that the server knows about has a corresponding entry in the table  
.This system table is used for implementing the [SHOW DATABASES](#) query

## system.detached\_parts

Contains information about detached parts of [MergeTree](#) tables. The [reason](#) column specifies why the part was detached. For user-detached parts, the reason is empty. Such parts can be attached with [ALTER TABLE ATTACH PARTITION|PART](#) command. For the description of other columns, see [system.parts](#)

## system.dictionaries

.Contains information about external dictionaries

:Columns

- .name String — Dictionary name
- .type String — Dictionary type: Flat, Hashed, Cache
- .origin String — Path to the configuration file that describes the dictionary
- .attribute.names Array(String) — Array of attribute names provided by the dictionary
- .attribute.types Array(String) — Corresponding array of attribute types that are provided by the dictionary
- .has\_hierarchy UInt8 — Whether the dictionary is hierarchical
- .bytes\_allocated UInt64 — The amount of RAM the dictionary uses
- .hit\_rate Float64 — For cache dictionaries, the percentage of uses for which the value was in the cache
- .element\_count UInt64 — The number of items stored in the dictionary
- load\_factor Float64 — The percentage full of the dictionary (for a hashed dictionary, the percentage filled in the .(hash table
- .creation\_time DateTime — The time when the dictionary was created or last successfully reloaded
- last\_exception String — Text of the error that occurs when creating or reloading the dictionary if the dictionary .couldn't be created
- .source String — Text describing the data source for the dictionary

Note that the amount of memory used by the dictionary is not proportional to the number of items stored in it. So for flat and cached dictionaries, all the memory cells are pre-assigned, regardless of how full the dictionary .actually is

## system.events

Contains information about the number of events that have occurred in the system. For example, in the table, you .can find how many **SELECT** queries are processed from the moment of ClickHouse server start

:Columns

- .event (String) — Event name
- .value (UInt64) — Count of events occurred
- .description (String) — Description of an event

### Example

SELECT * FROM system.events LIMIT 5		
t	value	description
Query	12	Number of queries started to be interpreted and maybe executed. Does not include queries that are failed to parse, that are rejected due to AST size limits; rejected due to quota limits or limits on number of simultaneously .running queries. May include internal queries initiated by ClickHouse itself. Does not count subqueries
SelectQuery	8	Same as Query, but only for SELECT queries.
FileOpen	73	Number of files opened.
ReadBufferFromFileDescriptorRead	155	Number of reads (read/pread) from a file descriptor. Does not include sockets.
ReadBufferFromFileDescriptorReadBytes	9931	Number of bytes read from file descriptors. If the file is compressed, this will .show compressed data size

### See Also

- .system.asynchronous\_metrics — Contains periodically calculated metrics
- .system.metrics — Contains instantly calculated metrics
- .Monitoring — Base concepts of ClickHouse monitoring

## system.functions

.Contains information about normal and aggregate functions

:Columns

- .name(String) - The name of the function
- .is\_aggregate(UInt8) — Whether the function is aggregate

## system.graphite\_retentions

.Contains information about parameters `graphite_rollup` which use in tables with `*GraphiteMergeTree` engines

:Columns

- .config\_name (String) - `graphite_rollup` parameter name
- .regex (String) - A pattern for the metric name
- .function (String) - The name of the aggregating function
- .age (UInt64) - The minimum age of the data in seconds
- .precision (UInt64) - How precisely to define the age of the data in seconds
- .priority (UInt16) - Pattern priority
- .is\_default (UInt8) - Is pattern default or not
- .Tables.database (Array(String)) - Array of databases names of tables, which use `config_name` parameter
- .Tables.table (Array(String)) - Array of tables names, which use `config_name` parameter

## system.merges

.Contains information about merges and part mutations currently in process for tables in the MergeTree family

:Columns

- .database String — The name of the database the table is in
- .table String — Table name
- .elapsed Float64 — The time elapsed (in seconds) since the merge started
- .progress Float64 — The percentage of completed work from 0 to 1
- .num\_parts UInt64 — The number of pieces to be merged
- .result\_part\_name String — The name of the part that will be formed as the result of merging
- .is\_mutation UInt8 - 1 if this process is a part mutation
- .total\_size\_bytes\_compressed UInt64 — The total size of the compressed data in the merged chunks
- .total\_size\_marks UInt64 — The total number of marks in the merged parts
- .bytes\_read\_uncompressed UInt64 — Number of bytes read, uncompressed
- .rows\_read UInt64 — Number of rows read
- .bytes\_written\_uncompressed UInt64 — Number of bytes written, uncompressed
- .rows\_written UInt64 — Number of lines rows written

## system.metrics

Contains metrics which can be calculated instantly, or have an current value. For example, a number of .simultaneously processed queries, the current value for replica delay. This table is always up to date

:Columns

- .metric (String) — Metric's name
- .value (Int64) — Metric's value
- .description (String) — Description of the metric

### Example

```
SELECT * FROM system.metrics LIMIT 10
```

metric	value	description
Query	1	Number of executing queries
Merge	0	Number of executing background merges
PartMutation	0	Number of mutations (ALTER DELETE/UPDATE)
ReplicatedFetch	0	Number of data parts fetching from replica
ReplicatedSend	0	Number of data parts sending to replicas
ReplicatedChecks	0	Number of data parts checking for consistency
BackgroundPoolTask	0	Number of active tasks in BackgroundProcessingPool (merges, mutations, fetches or replication queue bookkeeping)
BackgroundSchedulePoolTask	0	Number of active tasks in BackgroundSchedulePool. This pool is used for periodic tasks of ReplicatedMergeTree like cleaning old data parts, altering data parts, replica re-initialization, etc
DiskSpaceReservedForMerge	0	Disk space reserved for currently running background merges. It is slightly more than total size of currently merging parts
DistributedSend	0	Number of connections sending data, that was INSERTed to Distributed tables, to remote servers. Both synchronous and asynchronous mode

## See Also

- [.system.asynchronous\\_metrics](#) — Contains periodically calculated metrics
- [.system.events](#) — Contains a number of happened events
- [.Monitoring](#) — Base concepts of ClickHouse monitoring

## system.numbers

This table contains a single UInt64 column named 'number' that contains almost all the natural numbers starting from zero

You can use this table for tests, or if you need to do a brute force search

Reads from this table are not parallelized

## system.numbers\_mt

The same as 'system.numbers' but reads are parallelized. The numbers can be returned in any order

Used for tests

## system.one

This table contains a single row with a single 'dummy' UInt8 column containing the value 0

This table is used if a SELECT query doesn't specify the FROM clause

This is similar to the DUAL table found in other DBMSs

## system.parts

Contains information about parts of MergeTree tables

Each row describes one part of the data

Columns

- partition (String) – The partition name. To learn what a partition is, see the description of the [ALTER](#) query

:Formats

.YYYYMM for automatic partitioning by month -

.any\_string when partitioning manually -

.name (String) - Name of the data part

active (UInt8) - Indicates whether the part is active. If a part is active, it is used in a table; otherwise, it will be deleted. Inactive data parts remain after merging

marks (UInt64) - The number of marks. To get the approximate number of rows in a data part, multiply marks (by the index granularity (usually 8192

.marks\_size (UInt64) - The size of the file with marks

.rows (UInt64) - The number of rows

.bytes (UInt64) - The number of bytes when compressed

modification\_time (DateTime) - The modification time of the directory with the data part. This usually corresponds to the time of data part creation

.remove\_time (DateTime) - The time when the data part became inactive

refcount (UInt32) - The number of places where the data part is used. A value greater than 2 indicates that the data part is used in queries or merges

.min\_date (Date) - The minimum value of the date key in the data part

.max\_date (Date) - The maximum value of the date key in the data part

min\_block\_number (UInt64) - The minimum number of data parts that make up the current part after merging

max\_block\_number (UInt64) - The maximum number of data parts that make up the current part after merging

.level (UInt32) - Depth of the merge tree. If a merge was not performed, level=0

.primary\_key\_bytes\_in\_memory (UInt64) - The amount of memory (in bytes) used by primary key values

primary\_key\_bytes\_in\_memory\_allocated (UInt64) - The amount of memory (in bytes) reserved for primary key values

.database (String) - Name of the database

.table (String) - Name of the table

.engine (String) - Name of the table engine without parameters

is\_frozen (UInt8) - Flag that shows partition data backup existence. 1, the backup exists. 0, the backup doesn't exist. For more details, see **FREEZE PARTITION**

## system.part\_log

.The **system.part\_log** table is created only if the **part\_log** server setting is specified

This table contains information about the events that occurred with the **data parts** in the **MergeTree** family tables. For instance, adding or merging data

:The **system.part\_log** table contains the following columns

**event\_type** (Enum) — Type of the event that occurred with the data part. Can have one of the following values: **NEW\_PART** — inserting, **MERGE\_PARTS** — merging, **DOWNLOAD\_PART** — downloading, **REMOVE\_PART** — removing or detaching using **DETACH PARTITION**, **MUTATE\_PART** — updating



- `.event_date` (Date) — Event date
- `.event_time` (DateTime) — Event time
- `.duration_ms` (UInt64) — Duration
- `.database` (String) — Name of the database the data part is in
- `.table` (String) — Name of the table the data part is in
- `.part_name` (String) — Name of the data part
- `.partition_id` (String) — ID of the partition that the data part was inserted to. The column takes the 'all' value if the partitioning is by tuple
- `.rows` (UInt64) — The number of rows in the data part
- `.size_in_bytes` (UInt64) — Size of the data part in bytes
- `.merged_from` (Array(String)) — An array of names of the parts which the current part was made up from (after the merge)
- `.bytes_uncompressed` (UInt64) — Size of uncompressed bytes
- `.read_rows` (UInt64) — The number of rows was read during the merge
- `.read_bytes` (UInt64) — The number of bytes was read during the merge
- `.error` (UInt16) — The code number of the occurred error
- `.exception` (String) — Text message of the occurred error

The `system.part_log` table is created after the first inserting data to the `MergeTree` table

## system.processes

This system table is used for implementing the `SHOW PROCESSLIST` query

Columns

<code>user</code> String	- Name of the user who made the request. For distributed query processing, this is the user who helped the requestor server send the query to this server, not the user who made the distributed request on the requestor server
<code>.address</code> String	- The IP address the request was made from. The same for distributed processing
<code>.elapsed</code> Float64	- The time in seconds since request execution started
<code>rows_read</code> UInt64	- The number of rows read from the table. For distributed processing, on the requestor server, this is the total for all remote servers
<code>bytes_read</code> UInt64	- The number of uncompressed bytes read from the table. For distributed processing, on the requestor server, this is the total for all remote servers
<code>total_rows_approx</code> UInt64	- The approximation of the total number of rows that should be read. For distributed processing, on the requestor server, this is the total for all remote servers. It can be updated during request processing, when new sources to process become known
<code>.memory_usage</code> UInt64	- How much memory the request uses. It might not include some types of dedicated memory
<code>.query</code> String	- The query text. For INSERT, it doesn't include the data to insert
<code>.query_id</code> String	- Query ID, if defined

## system.query\_log

Contains information about queries execution. For each query, you can see processing start time, duration of processing, error message and other information

Note

The table doesn't contain input data for `INSERT` queries

ClickHouse creates this table only if the `query_log` server parameter is specified. This parameter sets the logging rules. For example, a logging interval or name of a table the queries will be logged in

To enable query logging, set the parameter `log_queries` to 1. For details, see the `Settings` section

:The `system.query_log` table registers two kinds of queries

- .Initial queries, that were run directly by the client .1
- Child queries that were initiated by other queries (for distributed query execution). For such a kind of queries, .2
- information about the parent queries is shown in the `initial_*` columns

:Columns

- :`type` (UInt8) — Type of event that occurred when executing the query. Possible values
  - .Successful start of query execution — 1 ○
  - .Successful end of query execution — 2 ○
  - .Exception before the start of query execution — 3 ○
  - .Exception during the query execution — 4 ○
- .`event_date` (Date) — Event date •
- .`event_time` (DateTime) — Event time •
- .`query_start_time` (DateTime) — Time of the query processing start •
- .`query_duration_ms` (UInt64) — Duration of the query processing •
- .`read_rows` (UInt64) — Number of read rows •
- .`read_bytes` (UInt64) — Number of read bytes •
- .`written_rows` (UInt64) — For `INSERT` queries, number of written rows. For other queries, the column value is 0 •
- .`written_bytes` (UInt64) — For `INSERT` queries, number of written bytes. For other queries, the column value is 0 •
- .`result_rows` (UInt64) — Number of rows in a result •
- .`result_bytes` (UInt64) — Number of bytes in a result •
- .`memory_usage` (UInt64) — Memory consumption by the query •
- .`query` (String) — Query string •
- .`exception` (String) — Exception message •
- .`stack_trace` (String) — Stack trace (a list of methods called before the error occurred). An empty string, if the query is completed successfully •
- :`is_initial_query` (UInt8) — Kind of query. Possible values
  - .Query was initiated by the client — 1 ○
  - .Query was initiated by another query for distributed query execution — 0 ○
- .`user` (String) — Name of the user initiated the current query •
- .`query_id` (String) — ID of the query •
- .`address` (FixedString(16)) — IP address the query was initiated from •
- .`port` (UInt16) — A server port that was used to receive the query •
- .`initial_user` (String) — Name of the user who run the parent query (for distributed query execution) •
- .`initial_query_id` (String) — ID of the parent query •
- .`initial_address` (FixedString(16)) — IP address that the parent query was launched from •
- .`initial_port` (UInt16) — A server port that was used to receive the parent query from the client •
- :`interface` (UInt8) — Interface that the query was initiated from. Possible values
  - .TCP — 1 ○
  - .HTTP — 2 ○
- .`os_user` (String) — User's OS •
- .`client_hostname` (String) — Server name that the `clickhouse-client` is connected to •
- .`client_name` (String) — The `clickhouse-client` name •
- .`client_revision` (UInt32) — Revision of the `clickhouse-client` •
- .`client_version_major` (UInt32) — Major version of the `clickhouse-client` •
- .`client_version_minor` (UInt32) — Minor version of the `clickhouse-client` •
- .`client_version_patch` (UInt32) — Patch component of the `clickhouse-client` version •
- :`http_method` (UInt8) — HTTP method initiated the query. Possible values
  - .The query was launched from the TCP interface — 0 ○
  - .GET method is used — 1 ○
  - .POST method is used — 2 ○
- .`http_user_agent` (String) — The `UserAgent` header passed in the HTTP request •
- .`quota_key` (String) — The quota key specified in `quotas` setting •
- .`revision` (UInt32) — ClickHouse revision •
- .`thread_numbers` (Array(UInt32)) — Number of threads that are participating in query execution •

- **ProfileEvents.Names** (Array(String)) — Counters that measure the following metrics
  - Time spent on reading and writing over the network
  - Time spent on reading and writing to a disk
  - Number of network errors
  - Time spent on waiting when the network bandwidth is limited
- **ProfileEvents.Values** (Array(UInt64)) — Values of metrics that are listed in the **ProfileEvents.Names** column
- **Settings.Names** (Array(String)) — Names of settings that were changed when the client run a query. To enable logging of settings changing, set the **log\_query\_settings** parameter to 1
- **Settings.Values** (Array(String)) — Values of settings that are listed in the **Settings.Names** column

Each query creates one or two rows in the **query\_log** table, depending on the status of the query

- 1. If the query execution is successful, two events with types 1 and 2 are created (see the **type** column)
- 2. If the error occurred during the query processing, two events with types 1 and 4 are created
- 3. If the error occurred before the query launching, a single event with type 3 is created

By default, logs are added into the table at intervals of 7,5 seconds. You can set this interval in the **query\_log** server setting (see the **flush\_interval\_milliseconds** parameter). To flush the logs forcibly from the memory buffer into the table, use the **SYSTEM FLUSH LOGS** query

When the table is deleted manually, it will be automatically created on the fly. Note that all the previous logs will be deleted

#### Note

The storage period for logs is unlimited; the logs aren't automatically deleted from the table. You need to organize the removing of non-actual logs yourself

You can specify an arbitrary partitioning key for the **system.query\_log** table in the **query\_log** server setting (see the **partition\_by** parameter)

## system.replicas

- Contains information and status for replicated tables residing on the local server
- This table can be used for monitoring. The table contains a row for every Replicated\* table

#### Example

```
* SELECT
FROM system.replicas
WHERE table = 'visits
FORMAT Vertical
```

:Row 1	
database:	merge
table:	visits
engine:	ReplicatedCollapsingMergeTree
is_leader:	1
is_readonly:	0
is_session_expired:	0
future_parts:	1
parts_to_check:	0
zookeeper_path:	/clickhouse/tables/01-06/visits
replica_name:	example01-06-1.yandex.ru
replica_path:	/clickhouse/tables/01-06/visits/replicas/example01-06-1.yandex.ru
columns_version:	9
queue_size:	1
inserts_in_queue:	0
merges_in_queue:	1
log_max_index:	596273
log_pointer:	596274
total_replicas:	2
active_replicas:	2

:Columns

database:	Database name
table:	Table name
engine:	Table engine name
.is_leader:	Whether the replica is the leader
.Only one replica at a time can be the leader. The leader is responsible for selecting background merges to perform .Note that writes can be performed to any replica that is available and has a session in ZK, regardless of whether it is a leader	
.is_readonly:	Whether the replica is in read-only mode
This mode is turned on if the config doesn't have sections with ZooKeeper, if an unknown error occurred when reinitializing sessions in ZooKeeper, and during session reinitialization in ZooKeeper	
.is_session_expired:	Whether the session with ZooKeeper has expired
.Basically the same as 'is_readonly	
.future_parts:	The number of data parts that will appear as the result of INSERTs or merges that haven't been done yet
.parts_to_check:	The number of data parts in the queue for verification
.A part is put in the verification queue if there is suspicion that it might be damaged	
.zookeeper_path:	Path to table data in ZooKeeper
.replica_name:	Replica name in ZooKeeper. Different replicas of the same table have different names
.replica_path:	Path to replica data in ZooKeeper. The same as concatenating 'zookeeper_path/replicas/replica_path
.columns_version:	Version number of the table structure
Indicates how many times ALTER was performed. If replicas have different versions, it means some replicas haven't made all of the ALTERs yet	
.queue_size:	Size of the queue for operations waiting to be performed
.Operations include inserting blocks of data, merges, and certain other actions .It usually coincides with 'future_parts	
.inserts_in_queue:	Number of inserts of blocks of data that need to be made
.Insertions are usually replicated fairly quickly. If this number is large, it means something is wrong	
.merges_in_queue:	The number of merges waiting to be made
.Sometimes merges are lengthy, so this value may be greater than zero for a long time	
.The next 4 columns have a non-zero value only where there is an active session with ZK	
.log_max_index:	Maximum entry number in the log of general activity
.log_pointer:	Maximum entry number in the log of general activity that the replica copied to its execution queue, plus one
.If log_pointer is much smaller than log_max_index, something is wrong	
.total_replicas:	The total number of known replicas of this table
.active_replicas:	The number of replicas of this table that have a session in ZooKeeper (i.e., the number of functioning replicas

If you request all the columns, the table may work a bit slowly, since several reads from ZooKeeper are made for each row

If you don't request the last 4 columns (log\_max\_index, log\_pointer, total\_replicas, active\_replicas), the table works quickly

:For example, you can check that everything is working correctly like this

```
SELECT
,database
,table
,is_leader
,is_readonly
,is_session_expired
,future_parts
,parts_to_check
,columns_version
,queue_size
,inserts_in_queue
,merges_in_queue
,log_max_index
,log_pointer
,total_replicas
active_replicas
FROM system.replicas
WHERE
is_readonly
OR is_session_expired
OR future_parts > 20
OR parts_to_check > 10
OR queue_size > 20
OR inserts_in_queue > 10
OR log_max_index - log_pointer > 10
OR total_replicas < 2
OR active_replicas < total_replicas
```

.If this query doesn't return anything, it means that everything is fine

## system.settings

- .Contains information about settings that are currently in use
- .I.e. used for executing the query you are using to read from the system.settings table

:Columns

.name String	— Setting name
.value String	— Setting value
.changed UInt8	— Whether the setting was explicitly defined in the config or explicitly changed

:Example

```
* SELECT
FROM system.settings
WHERE changed
```

name	value	changed
max_threads	8	1
use_uncompressed_cache	0	1
load_balancing	random	1
max_memory_usage	10000000000	1

## system.tables

- .Contains metadata of each table that the server knows about. Detached tables are not shown in [system.tables](#)

:(This table contains the following columns (the type of the corresponding column is shown in brackets)

- .database (String) — The name of database the table is in
- .name (String) — Table name
- .(engine (String) — Table engine name (without parameters
- .is\_temporary (UInt8) - Flag that indicates whether the table is temporary

- `.data_path` (String) - Path to the table data in the file system
  - `.metadata_path` (String) - Path to the table metadata in the file system
  - `.metadata_modification_time` (DateTime) - Time of latest modification of the table metadata
  - `.dependencies_database` (Array(String)) - Database dependencies
  - `.(dependencies_table` (Array(String)) - Table dependencies (`MaterializedView` tables based on the current table
  - `.create_table_query` (String) - The query that was used to create the table
  - `.engine_full` (String) - Parameters of the table engine
  - `.partition_key` (String) - The partition key expression specified in the table
  - `.sorting_key` (String) - The sorting key expression specified in the table
  - `.primary_key` (String) - The primary key expression specified in the table
  - `.sampling_key` (String) - The sampling key expression specified in the table
- The `system.tables` is used in `SHOW TABLES` query implementation

## system.zookeeper

The table does not exist if ZooKeeper is not configured. Allows reading data from the ZooKeeper cluster defined in the config

The query must have a 'path' equality condition in the WHERE clause. This is the path in ZooKeeper for the children that you want to get data for

The query `SELECT * FROM system.zookeeper WHERE path = '/clickhouse'` outputs data for all children on the `/clickhouse` node

`.'/` = To output data for all root nodes, write path

If the path specified in 'path' doesn't exist, an exception will be thrown

:Columns

- `.name String` — The name of the node
- `.path String` — The path to the node
- `.value String` — Node value
- `.dataLength Int32` — Size of the value
- `.numChildren Int32` — Number of descendants
- `.czxid Int64` — ID of the transaction that created the node
- `.mzxid Int64` — ID of the transaction that last changed the node
- `.pzxid Int64` — ID of the transaction that last deleted or added descendants
- `.ctime DateTime` — Time of node creation
- `.mtime DateTime` — Time of the last modification of the node
- `.version Int32` — Node version: the number of times the node was changed
- `.cversion Int32` — Number of added or removed descendants
- `.aversion Int32` — Number of changes to the ACL
- `.ephemeralOwner Int64` — For ephemeral nodes, the ID of the session that owns this node

:Example

```
* SELECT
FROM system.zookeeper
WHERE path = '/clickhouse/tables/01-08/visits/replicas'
FORMAT Vertical
```

:Row 1	
name:	example01-08-1.yandex.ru
:value	
czxid:	932998691229
mzxid:	932998691229
ctime:	2015-03-27 16:49:51
mtime:	2015-03-27 16:49:51
version:	0
cversion:	47
aversion:	0
ephemeralOwner:	0
dataLength:	0
numChildren:	7
pzxid:	987021031383
path:	/clickhouse/tables/01-08/visits/replicas
:Row 2	
name:	example01-08-2.yandex.ru
:value	
czxid:	933002738135
mzxid:	933002738135
ctime:	2015-03-27 16:57:01
mtime:	2015-03-27 16:57:01
version:	0
cversion:	37
aversion:	0
ephemeralOwner:	0
dataLength:	0
numChildren:	7
pzxid:	987021252247
path:	/clickhouse/tables/01-08/visits/replicas

## system.mutations

The table contains information about **mutations** of MergeTree tables and their progress. Each mutation command is represented by a single row. The table has the following columns

**.database, table** - The name of the database and table to which the mutation was applied

**mutation\_id** - The ID of the mutation. For replicated tables these IDs correspond to znode names in the `<table_path_in_zookeeper>/mutations/` directory in ZooKeeper. For unreplicated tables the IDs correspond to file names in the data directory of the table

**.command** - The mutation command string (the part of the query after `ALTER TABLE [db.]table`

**.create\_time** - When this mutation command was submitted for execution

**block\_numbers.partition\_id, block\_numbers.number** - A Nested column. For mutations of replicated tables contains one record for each partition: the partition ID and the block number that was acquired by the mutation (in each partition only parts that contain blocks with numbers less than the block number acquired by the mutation in that partition will be mutated). Because in non-replicated tables blocks numbers in all partitions form a single sequence, for mutations of non-replicated tables the column will contain one record with a single block number acquired by the mutation

**.parts\_to\_do** - The number of data parts that need to be mutated for the mutation to finish

**is\_done** - Is the mutation done? Note that even if `parts_to_do = 0` it is possible that a mutation of a replicated table is not done yet because of a long-running INSERT that will create a new data part that will need to be mutated

If there were problems with mutating some parts the following columns contain additional information



.**latest\_failed\_part** - The name of the most recent part that could not be mutated

.**latest\_fail\_time** - The time of the most recent part mutation failure

.**latest\_fail\_reason** - The exception message that caused the most recent part mutation failure

## Server configuration parameters

.This section contains descriptions of server settings that cannot be changed at the session or query level

.These settings are stored in the **config.xml** file on the ClickHouse server

.Other settings are described in the "**Settings**" section

Before studying the settings, read the **Configuration files** section and note the use of substitutions (the **incl** and **optional** attributes

## Server settings

### builtin\_dictionaries\_reload\_interval

.The interval in seconds before reloading built-in dictionaries

ClickHouse reloads built-in dictionaries every x seconds. This makes it possible to edit dictionaries "on the fly" without restarting the server

.Default value: 3600

#### Example

```
<builtin_dictionaries_reload_interval>3600</builtin_dictionaries_reload_interval>
```

## compression

.Data compression settings

Warning

.Don't use it if you have just started using ClickHouse

:The configuration looks like this

```
<compression>
<case>
</parameters>
<case/>
...
</compression/>
```

.<You can configure multiple sections **<case**

:<Block field **<case**

.**min\_part\_size** - The minimum size of a table part

.**min\_part\_size\_ratio** - The ratio of the minimum size of a table part to the full size of the table

.(**method** - Compression method. Acceptable values : **lz4** or **zstd**(experimental)

ClickHouse checks **min\_part\_size** and **min\_part\_size\_ratio** and processes the **case** blocks that match these conditions. If none of the **<case>** matches, ClickHouse applies the **lz4** compression algorithm

#### Example

```
<"compression incl="clickhouse_compression>
<case>
<min_part_size>10000000000</min_part_size>
<min_part_size_ratio>0.01</min_part_size_ratio>
<method>zstd</method>
<case/>
<compression/>
```

## default\_database

.The default database

.To get a list of databases, use the **SHOW DATABASES** query

### Example

```
<default_database>default</default_database>
```

## default\_profile

.Default settings profile

.Settings profiles are located in the file specified in the parameter **user\_config**

### Example

```
<default_profile>default</default_profile>
```

## dictionaries\_config

.The path to the config file for external dictionaries

:Path

.Specify the absolute path or the path relative to the server config file

.? The path can contain wildcards \* and

."See also "**External dictionaries**

### Example

```
<dictionaries_config>*_dictionary.xml</dictionaries_config>
```

## dictionaries\_lazy\_load

.Lazy loading of dictionaries

If **true**, then each dictionary is created on first use. If dictionary creation failed, the function that was using the dictionary throws an exception

.If **false**, all dictionaries are created when the server starts, and if there is an error, the server shuts down

.The default is **true**

### Example

```
<dictionaries_lazy_load>true</dictionaries_lazy_load>
```

## format\_schema\_path

.The path to the directory with the schemes for the input data, such as schemas for the **CapnProto** format

### Example

```
<-- .Directory containing schema files for various input formats --!>
<format_schema_path>format_schemas</format_schema_path>
```

# graphite

.Sending data to [Graphite](#)

:Settings

.host – The Graphite server

.port – The port on the Graphite server

.interval – The interval for sending, in seconds

.timeout – The timeout for sending data, in seconds

.root\_path – Prefix for keys

.metrics – Sending data from a :ref:system\_tables-system.metrics table

.events – Sending data from a :ref:system\_tables-system.events table

.asynchronous\_metrics – Sending data from a :ref:system\_tables-system.asynchronous\_metrics table

- 
- 
- 
- 
- 
- 
- 
- 

You can configure multiple `<graphite>` clauses. For instance, you can use this for sending different data at different intervals

## Example

```
<graphite>
<host>localhost</host>
<port>42000</port>
<timeout>0.1</timeout>
<interval>60</interval>
<root_path>one_min</root_path>
<metrics>true</metrics>
<events>true</events>
<asynchronous_metrics>true</asynchronous_metrics>
</graphite/>
```

# graphite\_rollup

.Settings for thinning data for Graphite

.For more details, see [GraphiteMergeTree](#)

## Example

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

# http\_port/https\_port

.(The port for connecting to the server over HTTP(s

.If `https_port` is specified, [openSSL](#) must be configured

.If `http_port` is specified, the openSSL configuration is ignored even if it is set

### Example

```
<https>0000</https>
```

## http\_server\_default\_response

.The page that is shown by default when you access the ClickHouse HTTP(s) server

### Example

.Opens <https://tabix.io/> when accessing [http://localhost: http\\_port](http://localhost: http_port)

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

## include\_from

.The path to the file with substitutions

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

### Example

```
<include_from>/etc/metrica.xml</include_from>
```

## interserver\_http\_port

.Port for exchanging data between ClickHouse servers

### Example

```
<interserver_http_port>9009</interserver_http_port>
```

## interserver\_http\_host

.The host name that can be used by other servers to access this server

.If omitted, it is defined in the same way as the [hostname-f](#) command

.Useful for breaking away from a specific network interface

### Example

```
<interserver_http_host>example.yandex.ru</interserver_http_host>
```

## keep\_alive\_timeout

The number of seconds that ClickHouse waits for incoming requests before closing the connection. Defaults to 3 seconds

### Example

```
<keep_alive_timeout>3</keep_alive_timeout>
```

## listen\_host

.:: Restriction on hosts that requests can come from. If you want the server to answer all of them, specify

:Examples

```
<listen_host>::1</listen_host>
<listen_host>127.0.0.1</listen_host>
```

## logger

.Logging settings

:Keys

- .level – Logging level. Acceptable values: `trace`, `debug`, `information`, `warning`, `error`
- .log – The log file. Contains all the entries according to `level`
- .errorlog – Error log file
- size – Size of the file. Applies to `log` and `errorlog`. Once the file reaches `size`, ClickHouse archives and renames it, and creates a new log file in its place
- .count – The number of archived log files that ClickHouse stores

### Example

```
<logger>
<level>trace</level>
<log>/var/log/clickhouse-server/clickhouse-server.log</log>
<errorlog>/var/log/clickhouse-server/clickhouse-server.err.log</errorlog>
<size>1000M</size>
<count>10</count>
</logger>
```

:Writing to the syslog is also supported. Config example

```
<logger>
<use_syslog>1</use_syslog>
<syslog>
<address>syslog.remote:10514</address>
<hostname>myhost.local</hostname>
<facility>LOG_LOCAL6</facility>
<format>syslog</format>
</syslog>
</logger>
```

:Keys

- .user\_syslog — Required setting if you want to write to the syslog
- .address — The host[:port] of syslogd. If omitted, the local daemon is used
- .hostname — Optional. The name of the host that logs are sent from
- facility — **The syslog facility keyword** in uppercase letters with the "LOG\_" prefix: (`LOG_USER`, `LOG_DAEMON`, `LOG_LOCAL3`, and so on
- .Default value: `LOG_USER` if `address` is specified, `LOG_DAEMON` otherwise
- .format – Message format. Possible values: `bsd` and `syslog`

## macros

.Parameter substitutions for replicated tables

.Can be omitted if replicated tables are not used

."For more information, see the section "**Creating replicated tables**"

### Example

```
</ "macros incl="macros" optional="true>
```

## mark\_cache\_size

.Approximate size (in bytes) of the cache of "marks" used by **MergeTree**

The cache is shared for the server and memory is allocated as needed. The cache size must be at least 5368709120

### Example

```
<mark_cache_size>5368709120</mark_cache_size>
```

## max\_concurrent\_queries

.The maximum number of simultaneously processed requests

### Example

```
<max_concurrent_queries>100</max_concurrent_queries>
```

## max\_connections

.The maximum number of inbound connections

### Example

```
<max_connections>4096</max_connections>
```

## max\_open\_files

.The maximum number of open files

.By default: **maximum**

.We recommend using this option in Mac OS X, since the **getrlimit()** function returns an incorrect value

### Example

```
<max_open_files>262144</max_open_files>
```

## max\_table\_size\_to\_drop

.Restriction on deleting tables

.If the size of a **MergeTree** table exceeds **max\_table\_size\_to\_drop** (in bytes), you can't delete it using a DROP query

If you still need to delete the table without restarting the ClickHouse server, create the **<clickhouse-path>/flags/force\_drop\_table** file and run the DROP query

.Default value: 50 GB

.The value 0 means that you can delete all tables without any restrictions

### Example

```
<max_table_size_to_drop>0</max_table_size_to_drop>
```

## merge\_tree

.Fine tuning for tables in the **MergeTree**

.For more information, see the MergeTreeSettings.h header file

### Example

```
<merge_tree>  
<max_suspicious_broken_parts>5</max_suspicious_broken_parts>  
</merge_tree/>
```

## openSSL

.SSL client/server configuration

Support for SSL is provided by the **libpoco** library. The interface is described in the file **SSLManager.h**

:Keys for server/client settings

- privateKeyFile – The path to the file with the secret key of the PEM certificate. The file may contain a key and certificate at the same time
- certificateFile – The path to the client/server certificate file in PEM format. You can omit it if privateKeyFile contains the certificate
- caConfig – The path to the file or directory that contains trusted root certificates
- verificationMode – The method for checking the node's certificates. Details are in the description of the Context class. Possible values: none, relaxed, strict, once
- verificationDepth – The maximum length of the verification chain. Verification will fail if the certificate chain length exceeds the set value
- loadDefaultCAFile – Indicates that built-in CA certificates for OpenSSL will be used. Acceptable values: true, false
- cipherList – Supported OpenSSL encryptions. For example: ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH
- cacheSessions – Enables or disables caching sessions. Must be used in combination with sessionIdContext. Acceptable values: true, false
- sessionIdContext – A unique set of random characters that the server appends to each generated identifier. The length of the string must not exceed SSL\_MAX\_SSL\_SESSION\_ID\_LENGTH. This parameter is always recommended, since it helps avoid problems both if the server caches the session and if the client requested {caching. Default value: \${application.name}
- sessionCacheSize – The maximum number of sessions that the server caches. Default value: 1024\*20. 0 – Unlimited sessions
- sessionTimeout – Time for caching the session on the server
- extendedVerification – Automatically extended verification of certificates after the session ends. Acceptable values: true, false
- requireTLSv1 – Require a TLSv1 connection. Acceptable values: true, false
- requireTLSv1\_1 – Require a TLSv1.1 connection. Acceptable values: true, false
- requireTLSv1\_2 – Require a TLSv1.2 connection. Acceptable values: true, false
- fips – Activates OpenSSL FIPS mode. Supported if the library's OpenSSL version supports FIPS
- privateKeyPassphraseHandler – Class (PrivateKeyPassphraseHandler subclass) that requests the passphrase for accessing the private key. For example: <privateKeyPassphraseHandler>, <name>KeyFileHandler</name>, <<options><password>test</password></options>>, </privateKeyPassphraseHandler>
- invalidCertificateHandler – Class (subclass of CertificateHandler) for verifying invalid certificates. For example: <invalidCertificateHandler> <name>ConsoleCertificateHandler</name> </invalidCertificateHandler>
- disableProtocols – Protocols that are not allowed to use
- preferServerCiphers – Preferred server ciphers on the client

## Example of settings

```

<openssl>
<server>
openssl req -subj "/CN=localhost" -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout /etc/clickhouse- --!>
<-- server/server.key -out /etc/clickhouse-server/server.crt
<certificateFile>/etc/clickhouse-server/server.crt</certificateFile>
<privateKeyFile>/etc/clickhouse-server/server.key</privateKeyFile>
<-- openssl dhparam -out /etc/clickhouse-server/dhparam.pem 4096 --!>
<dhParamsFile>/etc/clickhouse-server/dhparam.pem</dhParamsFile>
<verificationMode>none</verificationMode>
<loadDefaultCAFile>true</loadDefaultCAFile>
<cacheSessions>true</cacheSessions>
<disableProtocols>ssl2,ssl3</disableProtocols>
<preferServerCiphers>true</preferServerCiphers>
<server/>
<client>
<loadDefaultCAFile>true</loadDefaultCAFile>
<cacheSessions>true</cacheSessions>
<disableProtocols>ssl2,ssl3</disableProtocols>
<preferServerCiphers>true</preferServerCiphers>
<-- <Use for self-signed: <verificationMode>none</verificationMode --!>
<invalidCertificateHandler>
<-- <Use for self-signed: <name>AcceptCertificateHandler</name --!>
<name>RejectCertificateHandler</name>
<invalidCertificateHandler/>
<client/>
</openssl>

```

## part\_log

Logging events that are associated with **MergeTree**. For instance, adding or merging data. You can use the log to .simulate merge algorithms and compare their characteristics. You can visualize the merge process

Queries are logged in the **system.part\_log** table, not in a separate file. You can configure the name of this table in .(the **table** parameter (see below

:Use the following parameters to configure logging

- .**database** – Name of the database
- .**table** – Name of the system table
- .**partition\_by** – Sets a **custom partitioning key**
- .**flush\_interval\_milliseconds** – Interval for flushing data from the buffer in memory to the table

### Example

```

<part_log>
<database>system</database>
<table>part_log</table>
<partition_by>toMonday(event_date)</partition_by>
<flush_interval_milliseconds>7500</flush_interval_milliseconds>
</part_log>

```

## path

.The path to the directory containing data

Note

.The trailing slash is mandatory

### Example

```

<path>/var/lib/clickhouse/</path>

```

## query\_log



.Setting for logging queries received with the `log_queries=1` setting

Queries are logged in the `system.query_log` table, not in a separate file. You can change the name of the table in `.(the table parameter (see below`

:Use the following parameters to configure logging

`.database` – Name of the database

`.table` – Name of the system table the queries will be logged in

`.partition_by` – Sets a `custom partitioning key` for a system table

`.flush_interval_milliseconds` – Interval for flushing data from the buffer in memory to the table

- 
- 
- 
- 

If the table doesn't exist, ClickHouse will create it. If the structure of the query log changed when the ClickHouse `.server` was updated, the table with the old structure is renamed, and a new table is created automatically

### Example

```
<query_log>
<database>system</database>
<table>query_log</table>
<partition_by>toMonday(event_date)</partition_by>
<flush_interval_milliseconds>7500</flush_interval_milliseconds>
</query_log>
```

## remote\_servers

.Configuration of clusters used by the Distributed table engine

."For more information, see the section "`Table engines/Distributed`

### Example

```
</ "remote_servers incl="clickhouse_remote_servers>
```

."For the value of the `incl` attribute, see the section "`Configuration files`

## timezone

.The server's time zone

.(Specified as an IANA identifier for the UTC time zone or geographic location (for example, `Africa/Abidjan`

The time zone is necessary for conversions between String and DateTime formats when DateTime fields are output to text format (printed on the screen or in a file), and when getting DateTime from a string. In addition, the time zone is used in functions that work with the time and date if they didn't receive the time zone in the input `.parameters`

### Example

```
<timezone>Europe/Moscow</timezone>
```

## tcp\_port

.Port for communicating with clients over the TCP protocol

### Example

```
<tcp_port>9000</tcp_port>
```

## tcp\_port\_secure

.TCP port for secure communication with clients. Use it with `OpenSSL` settings

### Possible values

.Positive integer

#### Default value

```
<tcp_port_secure>9440</tcp_port_secure>
```

## tmp\_path

.Path to temporary data for processing large queries

Note

.The trailing slash is mandatory

#### Example

```
<tmp_path>/var/lib/clickhouse/tmp/</tmp_path>
```

## uncompressed\_cache\_size

.Cache size (in bytes) for uncompressed data used by table engines from the [MergeTree](#)

There is one shared cache for the server. Memory is allocated on demand. The cache is used if the option [.use\\_uncompressed\\_cache](#) is enabled

.The uncompressed cache is advantageous for very short queries in individual cases

#### Example

```
<uncompressed_cache_size>8589934592</uncompressed_cache_size>
```

## user\_files\_path

.[\(\)](#)The directory with user files. Used in the table function [file](#)

#### Example

```
<user_files_path>/var/lib/clickhouse/user_files/</user_files_path>
```

## users\_config

:Path to the file that contains

- .User configurations
- .Access rights
- .Settings profiles
- .Quota settings

- 
- 
- 
- 

#### Example

```
<users_config>users.xml</users_config>
```

## zookeeper

.Contains settings that allow ClickHouse to interact with a [ZooKeeper](#) cluster

ClickHouse uses ZooKeeper for storing metadata of replicas when using replicated tables. If replicated tables are not used, this section of parameters can be omitted

:This section contains the following parameters

.**node** — ZooKeeper endpoint. You can set multiple endpoints

:For example

```
<xml <node index="1"> <host>example_host</host> <port>2181</port> </node>
```

.The **index** attribute specifies the node order when trying to connect to the ZooKeeper cluster

.**session\_timeout** — Maximum timeout for the client session in milliseconds

.**root** — The **znode** that is used as the root for znodes used by the ClickHouse server. Optional

**identity** — User and password, that can be required by ZooKeeper to give access to requested znodes.

.Optional

### Example configuration

```
<zookeeper>
<node>
<host>example1</host>
<port>2181</port>
</node>
<node>
<host>example2</host>
<port>2181</port>
</node>
<session_timeout_ms>30000</session_timeout_ms>
<operation_timeout_ms>10000</operation_timeout_ms>
<-- .Optional. Chroot suffix. Should exist --!>
<root>/path/to/zookeeper/node</root>
<-- .Optional. Zookeeper digest ACL string --!>
<identity>user:password</identity>
</zookeeper>
```

### See Also

[Replication](#)

[ZooKeeper Programmer's Guide](#)

## use\_minimalistic\_part\_header\_in\_zookeeper

.Storage method for data part headers in ZooKeeper

:This setting only applies to the **MergeTree** family. It can be specified

.Globally in the **merge\_tree** section of the **config.xml** file

ClickHouse uses the setting for all the tables on the server. You can change the setting at any time. Existing tables change their behavior when the setting changes

.For each individual table

When creating a table, specify the corresponding **engine setting**. The behavior of an existing table with this setting does not change, even if the global setting changes

### Possible values

.Functionality is turned off — 0

.Functionality is turned on — 1

If **use\_minimalistic\_part\_header\_in\_zookeeper = 1**, then **replicated** tables store the headers of the data parts compactly using a single **znode**. If the table contains many columns, this storage method significantly reduces the volume of the data stored in Zookeeper

Attention

After applying `use_minimalistic_part_header_in_zookeeper = 1`, you can't downgrade the ClickHouse server to a version that doesn't support this setting. Be careful when upgrading ClickHouse on servers in a cluster. Don't upgrade all the servers at once. It is safer to test new versions of ClickHouse in a test environment, or on just a few servers of a cluster

Data part headers already stored with this setting can't be restored to their previous (non-compact) representation

**Default value:** 0

## Settings

There are multiple ways to make all the settings described below

Settings are configured in layers, so each subsequent layer redefines the previous settings

Ways to configure settings, in order of priority

Settings in the `users.xml` server configuration file

Set in the element `<profiles`

Session settings

Send `SET setting=value` from the ClickHouse console client in interactive mode

Similarly, you can use ClickHouse sessions in the HTTP protocol. To do this, you need to specify the `session_id`

HTTP parameter

Query settings

When starting the ClickHouse console client in non-interactive mode, set the startup parameter `--`

`.setting=value`

(...When using the HTTP API, pass CGI parameters (`URL?setting_1=value&setting_2=value`

Settings that can only be made in the server config file are not covered in this section

## Permissions for queries

Queries in ClickHouse can be divided into several types

Read data queries: `SELECT`, `SHOW`, `DESCRIBE`, `EXISTS`

Write data queries: `INSERT`, `OPTIMIZE`

Change settings queries: `SET`, `USE`

DDL queries: `CREATE`, `ALTER`, `RENAME`, `ATTACH`, `DETACH`, `DROP TRUNCATE`

`KILL QUERY`

The following settings regulate user permissions by the type of query

`readonly` — Restricts permissions for all types of queries except DDL queries

`allow_ddl` — Restricts permissions for DDL queries

`KILL QUERY` can be performed with any settings

## readonly

Restricts permissions for read data, write data and change settings queries

See how the queries are divided into types `above`

### Possible values

All queries are allowed — 0

Only read data queries are allowed — 1

Read data and change settings queries are allowed — 2

.After setting `readonly = 1`, the user can't change `readonly` and `allow_ddl` settings in the current session

When using the `GET` method in the `HTTP interface`, `readonly = 1` is set automatically. To modify data, use the `POST` method

Setting `readonly = 1` prohibit the user from changing all the settings. There is a way to prohibit the user from changing only specific settings, for details see [constraints on settings](#)

### Default value

0

## allow\_ddl

.Allows/denies `DDL` queries

.See how the queries are divided into types [above](#)

### Possible values

.DDL queries are not allowed — 0

.DDL queries are allowed — 1

- 
- 

.You cannot execute `SET allow_ddl = 1` if `allow_ddl = 0` for the current session

### Default value

1

## Restrictions on query complexity

.Restrictions on query complexity are part of the settings

.They are used in order to provide safer execution from the user interface

Almost all the restrictions only apply to `SELECT`. For distributed query processing, restrictions are applied on each server separately

ClickHouse checks the restrictions for data parts, not for each row. It means that you can exceed the value of restriction with a size of the data part

."Restrictions on the "maximum amount of something" can take the value 0, which means "unrestricted"

.Most restrictions also have an 'overflow\_mode' setting, meaning what to do when the limit is exceeded

It can take one of two values: `throw` or `break`. Restrictions on aggregation (`group_by_overflow_mode`) also have the value `any`

.(`throw` – Throw an exception (default

.`break` – Stop executing the query and return the partial result, as if the source data ran out

`any` (only for `group_by_overflow_mode`) – Continuing aggregation for the keys that got into the set, but don't add new keys to the set

## max\_memory\_usage

.The maximum amount of RAM to use for running a query on a single server

.In the default configuration file, the maximum is 10 GB

.The setting doesn't consider the volume of available memory or the total volume of memory on the machine

.The restriction applies to a single query within a single server

.You can use `SHOW PROCESSLIST` to see the current memory consumption for each query

.In addition, the peak memory consumption is tracked for each query and written to the log

.Memory usage is not monitored for the states of certain aggregate functions

Memory usage is not fully tracked for states of the aggregate functions `min`, `max`, `any`, `anyLast`, `argMin`, `argMax` from `.String` and `.Array` arguments

Memory consumption is also restricted by the parameters `max_memory_usage_for_user` and `max_memory_usage_for_all_queries`

## max\_memory\_usage\_for\_user

.The maximum amount of RAM to use for running a user's queries on a single server

.(Default values are defined in `Settings.h`. By default, the amount is not restricted (`max_memory_usage_for_user = 0`

.See also the description of `max_memory_usage`

## max\_memory\_usage\_for\_all\_queries

.The maximum amount of RAM to use for running all queries on a single server

Default values are defined in `Settings.h`. By default, the amount is not restricted (`max_memory_usage_for_all_queries = 0`

.See also the description of `max_memory_usage`

## max\_rows\_to\_read

The following restrictions can be checked on each block (instead of on each row). That is, the restrictions can be broken a little

.When running a query in multiple threads, the following restrictions apply to each thread separately

.Maximum number of rows that can be read from a table when running a query

## max\_bytes\_to\_read

.Maximum number of bytes (uncompressed data) that can be read from a table when running a query

## read\_overflow\_mode

.What to do when the volume of data read exceeds one of the limits: 'throw' or 'break'. By default, throw

## max\_rows\_to\_group\_by

Maximum number of unique keys received from aggregation. This setting lets you limit memory consumption when aggregating

## group\_by\_overflow\_mode

What to do when the number of unique keys for aggregation exceeds the limit: 'throw', 'break', or 'any'. By default, throw

Using the 'any' value lets you run an approximation of GROUP BY. The quality of this approximation depends on the statistical nature of the data

## max\_bytes\_before\_external\_group\_by

.Enables or disables execution of `GROUP BY` clause in external memory. See `GROUP BY in external memory`

:Possible values

.Maximum volume or RAM (in bytes) that can be used by `GROUP BY` operation  
. `GROUP BY` in external memory disabled — 0

.Default value: 0

## max\_rows\_to\_sort

.Maximum number of rows before sorting. This allows you to limit memory consumption when sorting

## max\_bytes\_to\_sort

.Maximum number of bytes before sorting

## sort\_overflow\_mode

What to do if the number of rows received before sorting exceeds one of the limits: 'throw' or 'break'. By default, .throw

## max\_result\_rows

Limit on the number of rows in the result. Also checked for subqueries, and on remote servers when running parts .of a distributed query

## max\_result\_bytes

.Limit on the number of bytes in the result. The same as the previous setting

## result\_overflow\_mode

.What to do if the volume of the result exceeds one of the limits: 'throw' or 'break'. By default, throw  
.Using 'break' is similar to using LIMIT

## max\_execution\_time

.Maximum query execution time in seconds  
.At this time, it is not checked for one of the sorting stages, or when merging and finalizing aggregate functions

## timeout\_overflow\_mode

.What to do if the query is run longer than 'max\_execution\_time': 'throw' or 'break'. By default, throw

## min\_execution\_speed

Minimal execution speed in rows per second. Checked on every data block when  
'timeout\_before\_checking\_execution\_speed' expires. If the execution speed is lower, an exception is thrown

## min\_execution\_speed\_bytes

Minimum number of execution bytes per second. Checked on every data block when  
'timeout\_before\_checking\_execution\_speed' expires. If the execution speed is lower, an exception is thrown

## max\_execution\_speed

Maximum number of execution rows per second. Checked on every data block when  
'timeout\_before\_checking\_execution\_speed' expires. If the execution speed is high, the execution speed will be .reduced

## max\_execution\_speed\_bytes

Maximum number of execution bytes per second. Checked on every data block when  
'timeout\_before\_checking\_execution\_speed' expires. If the execution speed is high, the execution speed will be .reduced

## timeout\_before\_checking\_execution\_speed

Checks that execution speed is not too slow (no less than 'min\_execution\_speed'), after the specified time in .seconds has expired

## max\_columns\_to\_read

Maximum number of columns that can be read from a table in a single query. If a query requires reading a greater number of columns, it throws an exception

## max\_temporary\_columns

Maximum number of temporary columns that must be kept in RAM at the same time when running a query, including constant columns. If there are more temporary columns than this, it throws an exception

## max\_temporary\_non\_const\_columns

The same thing as 'max\_temporary\_columns', but without counting constant columns

Note that constant columns are formed fairly often when running a query, but they require approximately zero computing resources

## max\_subquery\_depth

Maximum nesting depth of subqueries. If subqueries are deeper, an exception is thrown. By default, 100

## max\_pipeline\_depth

Maximum pipeline depth. Corresponds to the number of transformations that each data block goes through during query processing. Counted within the limits of a single server. If the pipeline depth is greater, an exception is thrown. By default, 1000

## max\_ast\_depth

Maximum nesting depth of a query syntactic tree. If exceeded, an exception is thrown

At this time, it isn't checked during parsing, but only after parsing the query. That is, a syntactic tree that is too deep can be created during parsing, but the query will fail. By default, 1000

## max\_ast\_elements

Maximum number of elements in a query syntactic tree. If exceeded, an exception is thrown

In the same way as the previous setting, it is checked only after parsing the query. By default, 50,000

## max\_rows\_in\_set

Maximum number of rows for a data set in the IN clause created from a subquery

## max\_bytes\_in\_set

Maximum number of bytes (uncompressed data) used by a set in the IN clause created from a subquery

## set\_overflow\_mode

What to do when the amount of data exceeds one of the limits: 'throw' or 'break'. By default, throw

## max\_rows\_in\_distinct

Maximum number of different rows when using DISTINCT

## max\_bytes\_in\_distinct

Maximum number of bytes used by a hash table when using DISTINCT

## distinct\_overflow\_mode

What to do when the amount of data exceeds one of the limits: 'throw' or 'break'. By default, throw

## max\_rows\_to\_transfer

Maximum number of rows that can be passed to a remote server or saved in a temporary table when using GLOBAL IN



## max\_bytes\_to\_transfer

Maximum number of bytes (uncompressed data) that can be passed to a remote server or saved in a temporary table when using GLOBAL IN

## transfer\_overflow\_mode

.What to do when the amount of data exceeds one of the limits: 'throw' or 'break'. By default, throw

## max\_rows\_in\_join

.Limits the number of rows in the hash table that is used when joining tables

.This settings applies to **SELECT ... JOIN** operations and the **Join table engine**

.If a query contains multiple joins, ClickHouse checks this setting for every intermediate result

ClickHouse can proceed with different actions when the limit is reached. Use the **join\_overflow\_mode** settings to choose the action

:Possible values

- .Positive integer
- .Unlimited number of rows — 0

.Default value: 0

## max\_bytes\_in\_join

.Limits hash table size in bytes that is used when joining tables

.This settings applies to **SELECT ... JOIN** operations and the **Join table engine**

.If query contains some joins, ClickHouse checks this setting for every intermediate result

ClickHouse can proceed with different actions when the limit is reached. Use the **join\_overflow\_mode** settings to choose the action

:Possible values

- .Positive integer
- .Memory control is disabled — 0

.Default value: 0

## join\_overflow\_mode

:Defines an action that ClickHouse performs, when any of the following join limits is reached

- max\_bytes\_in\_join**
- max\_rows\_in\_join**

:Possible values

- .**THROW** — ClickHouse throws an exception and breaks operation
- .**BREAK** — ClickHouse breaks operation and doesn't throw an exception

.Default value: **THROW**

### See Also

- JOIN clause**
- Join table engine**

# max\_partitions\_per\_insert\_block

.Limits the maximum number of partitions in a single inserted block

.Positive integer

.Unlimited number of partitions — 0

.Default value: 100

## Details

When inserting data, ClickHouse calculates the number of partitions in the inserted block. If the number of partitions is more than `max_partitions_per_insert_block`, ClickHouse throws an exception with the following text

Too many partitions for single INSERT block (more than " + toString(max\_parts) + "). The limit is controlled by " 'max\_partitions\_per\_insert\_block' setting. Large number of partitions is a common misconception. It will lead to severe negative performance impact, including slow server startup, slow INSERT queries and slow SELECT queries. Recommended total number of partitions for a table is under 1000..10000. Please note, that partitioning is not intended to speed up SELECT queries (ORDER BY key is sufficient to make range queries fast). Partitions are ".(intended for data manipulation (DROP PARTITION, etc

## Settings

### distributed\_product\_mode

.Changes the behavior of `distributed subqueries`

ClickHouse applies this setting when the query contains the product of distributed tables, i.e. when the query for a distributed table contains a non-GLOBAL subquery for the distributed table

:Restrictions

.Only applied for IN and JOIN subqueries

.Only if the FROM section uses a distributed table containing more than one shard

.If the subquery concerns a distributed table containing more than one shard

.Not used for a table-valued `remote` function

:Possible values

`deny` — Default value. Prohibits using these types of subqueries (returns the "Double-distributed in/JOIN (subqueries is denied" exception

`local` — Replaces the database and table in the subquery with local ones for the destination server (shard), leaving the normal `IN/JOIN`

`global` — Replaces the `IN/JOIN` query with `GLOBAL IN/GLOBAL JOIN`

`allow` — Allows the use of these types of subqueries

### enable\_optimize\_predicate\_expression

.Turns on predicate pushdown in `SELECT` queries

.Predicate pushdown may significantly reduce network traffic for distributed queries

:Possible values

.Disabled — 0

.Enabled — 1

.Default value: 0

## Usage

:Consider the following queries

'SELECT count() FROM test\_table WHERE date = '2018-10-10' .1  
'SELECT count() FROM (SELECT \* FROM test\_table) WHERE date = '2018-10-10' .2

If `enable_optimize_predicate_expression = 1`, then the execution time of these queries is equal, because ClickHouse applies `WHERE` to the subquery when processing it

If `enable_optimize_predicate_expression = 0`, then the execution time of the second query is much longer, because the `WHERE` clause applies to all the data after the subquery finishes

## fallback\_to\_stale\_replicas\_for\_distributed\_queries

."Forces a query to an out-of-date replica if updated data is not available. See "[Replication](#)"

.ClickHouse selects the most relevant from the outdated replicas of the table

.Used when performing `SELECT` from a distributed table that points to replicated tables

.(By default, 1 (enabled)

## force\_index\_by\_date

.Disables query execution if the index can't be used by date

.Works with tables in the MergeTree family

If `force_index_by_date=1`, ClickHouse checks whether the query has a date key condition that can be used for restricting data ranges. If there is no suitable condition, it throws an exception. However, it does not check whether the condition actually reduces the amount of data to read. For example, the condition `Date != '2000-01-01'` is acceptable even when it matches all the data in the table (i.e., running the query requires a full scan). For more "information about ranges of data in MergeTree tables, see "[MergeTree](#)"

## force\_primary\_key

.Disables query execution if indexing by the primary key is not possible

.Works with tables in the MergeTree family

If `force_primary_key=1`, ClickHouse checks to see if the query has a primary key condition that can be used for restricting data ranges. If there is no suitable condition, it throws an exception. However, it does not check whether the condition actually reduces the amount of data to read. For more information about data ranges in "MergeTree tables, see "[MergeTree](#)"

## fsync\_metadata

.Enables or disables `fsync` when writing `.sql` files. Enabled by default

It makes sense to disable it if the server has millions of tiny table chunks that are constantly being created and destroyed

## enable\_http\_compression

.Enables or disables data compression in the response to an HTTP request

.For more information, read the [HTTP interface description](#)

:Possible values

.Disabled — 0

.Enabled — 1

.Default value: 0

## http\_zlib\_compression\_level

.Sets the level of data compression in the response to an HTTP request if `enable_http_compression = 1`

.Possible values: Numbers from 1 to 9

.Default value: 3

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

Enables or disables checksum verification when decompressing the HTTP POST data from the client. Used only for `ClickHouse native compression format` (not used with `gzip` or `deflate`)

.For more information, read the [HTTP interface description](#)

:Possible values

.Disabled — 0

•

.Enabled — 1

•

.Default value: 0

## send\_progress\_in\_http\_headers

.Enables or disables sending of the `X-ClickHouse-Progress` HTTP response headers in an answer from `clickhouse-server`

.For more information, read the [HTTP interface description](#)

:Possible values

.Disabled — 0

•

.Enabled — 1

•

.Default value: 0

## input\_format\_allow\_errors\_num

.(.Sets the maximum number of acceptable errors when reading from text formats (CSV, TSV, etc

.The default value is 0

.Always pair it with `input_format_allow_errors_ratio`. To skip errors, both settings must be greater than 0

If an error occurred while reading rows but the error counter is still less than `input_format_allow_errors_num`, ClickHouse ignores the row and moves on to the next one

.If `input_format_allow_errors_num` is exceeded, ClickHouse throws an exception

## input\_format\_allow\_errors\_ratio

.(.Sets the maximum percentage of errors allowed when reading from text formats (CSV, TSV, etc

.The percentage of errors is set as a floating-point number between 0 and 1

.The default value is 0

.Always pair it with `input_format_allow_errors_num`. To skip errors, both settings must be greater than 0

If an error occurred while reading rows but the error counter is still less than `input_format_allow_errors_ratio`, ClickHouse ignores the row and moves on to the next one

.If `input_format_allow_errors_ratio` is exceeded, ClickHouse throws an exception

## input\_format\_values\_interpret\_expressions

Enables or disables the full SQL parser if the fast stream parser can't parse the data. This setting is used only for the `Values` format at the data insertion. For more information about syntax parsing, see the [Syntax](#) section

:Possible values

.Disabled — 0

.In this case, you must provide formatted data. See the [Formats](#) section

.Enabled — 1

In this case, you can use an SQL expression as a value, but data insertion is much slower this way. If you .insert only formatted data, then ClickHouse behaves as if the setting value is 0

.Default value: 1

### Example of Use

.Insert the [DateTime](#) type value with the different settings

```
;SET input_format_values_interpret_expressions = 0
((()INSERT INTO datetime_t VALUES (now
```

```
:Exception on client
(Code: 27. DB::Exception: Cannot parse input: expected ) before: now()): (at row 1
```

```
;SET input_format_values_interpret_expressions = 1
((()INSERT INTO datetime_t VALUES (now
```

```
.Ok
```

:The last query is equivalent to the following

```
;SET input_format_values_interpret_expressions = 0
()INSERT INTO datetime_t SELECT now
```

```
.Ok
```

## input\_format\_defaults\_for\_omitted\_fields

Turns on/off the extended data exchange between a ClickHouse client and a ClickHouse server. This setting .applies for [INSERT](#) queries

When executing the [INSERT](#) query, the ClickHouse client prepares data and sends it to the server for writing. The client gets the table structure from the server when preparing the data. In some cases, the client needs more information than the server sends by default. Turn on the extended data exchange with

[.input\\_format\\_defaults\\_for\\_omitted\\_fields = 1](#)

When the extended data exchange is enabled, the server sends the additional metadata along with the table .structure. The composition of the metadata depends on the operation

:Operations where you may need the extended data exchange enabled

.Inserting data in [JSONEachRow](#) format

.For all other operations, ClickHouse doesn't apply the setting

Note

The extended data exchange functionality consumes additional computing resources on the server and can .reduce performance

:Possible values

.Disabled — 0

.Enabled — 1

.Default value: 0

# input\_format\_skip\_unknown\_fields

.Enables or disables skipping insertion of extra data

When writing data, ClickHouse throws an exception if input data contain columns that do not exist in the target table. If skipping is enabled, ClickHouse doesn't insert extra data and doesn't throw an exception

:Supported formats

- JSONEachRow
- CSVWithNames
- TabSeparatedWithNames
- TSKV

:Possible values

- .Disabled — 0
- .Enabled — 1

.Default value: 0

# input\_format\_with\_names\_use\_header

.Enables or disables checking the column order when inserting data

To improve insert performance, we recommend disabling this check if you are sure that the column order of the input data is the same as in the target table

:Supported formats

- CSVWithNames
- TabSeparatedWithNames

:Possible values

- .Disabled — 0
- .Enabled — 1

.Default value: 1

# join\_default\_strictness

.Sets default strictness for JOIN clauses

:Possible values

- ALL — If the right table has several matching rows, ClickHouse creates a Cartesian product from matching rows. This is the normal JOIN behavior from standard SQL
- ANY — If the right table has several matching rows, only the first one found is joined. If the right table has only one matching row, the results of ANY and ALL are the same
- ASOF — For joining sequences with an uncertain match
- Empty string — If ALL or ANY is not specified in the query, ClickHouse throws an exception

.Default value: ALL

# join\_any\_take\_last\_row

.Changes behavior of join operations with ANY strictness

Attention

.This setting applies only for the Join table engine

:Possible values

- .If the right table has more than one matching row, only the first one found is joined — 0
- .If the right table has more than one matching row, only the last one found is joined — 1

.Default value: 0

### See Also

- [JOIN clause](#)
- [Join table engine](#)
- [join\\_default\\_strictness](#)

## join\_use\_nulls

Sets the type of **JOIN** behavior. When merging tables, empty cells may appear. ClickHouse fills them differently .based on this setting

:Possible values

- .The empty cells are filled with the default value of the corresponding field type — 0
- JOIN** behaves the same way as in standard SQL. The type of the corresponding field is converted to — 1
- .**Nullable**, and empty cells are filled with **NULL**

.Default value: 0

## join\_any\_take\_last\_row

Changes the behavior of **ANY JOIN**. When disabled, **ANY JOIN** takes the first row found for a key. When enabled, **ANY JOIN** takes the last matched row, if there are multiple rows for the same key. The setting is used only in **Join table engine**

:Possible values

- .Disabled — 0
- .Enabled — 1

.Default value: 1

## max\_block\_size

In ClickHouse, data is processed by blocks (sets of column parts). The internal processing cycles for a single block are efficient enough, but there are noticeable expenditures on each block. The **max\_block\_size** setting is a recommendation for what size of block (in number of rows) to load from tables. The block size shouldn't be too small, so that the expenditures on each block are still noticeable, but not too large, so that the query with LIMIT that is completed after the first block is processed quickly. The goal is to avoid consuming too much memory .when extracting a large number of columns in multiple threads, and to preserve at least some cache locality

.Default value: 65,536

Blocks the size of **max\_block\_size** are not always loaded from the table. If it is obvious that less data needs to be .retrieved, a smaller block is processed

## preferred\_block\_size\_bytes

Used for the same purpose as **max\_block\_size**, but it sets the recommended block size in bytes by adapting it to the .number of rows in the block

.However, the block size cannot be more than **max\_block\_size** rows

.By default: 1,000,000. It only works when reading from MergeTree engines

## merge\_tree\_uniform\_read\_distribution

ClickHouse uses multiple threads when reading from **MergeTree\*** tables. This setting turns on/off the uniform distribution of reading tasks over the working threads. The algorithm of the uniform distribution aims to make .execution time for all the threads approximately equal in a **SELECT** query

### Possible values

- .Do not use uniform read distribution — 0
- .Use uniform read distribution — 1

- 
- 

**.Default value:** 1

## merge\_tree\_min\_rows\_for\_concurrent\_read

If the number of rows to be read from a file of a **MergeTree\*** table exceeds **merge\_tree\_min\_rows\_for\_concurrent\_read**, then ClickHouse tries to perform a concurrent reading from this file on several threads

### Possible values

.Any positive integer

**.Default value:** 163840

## merge\_tree\_min\_rows\_for\_seek

If the distance between two data blocks to be read in one file is less than **merge\_tree\_min\_rows\_for\_seek** rows, then ClickHouse does not seek through the file, but reads the data sequentially

### Possible values

.Any positive integer

**.Default value:** 0

## merge\_tree\_coarse\_index\_granularity

When searching data, ClickHouse checks the data marks in the index file. If ClickHouse finds that required keys are in some range, it divides this range into **merge\_tree\_coarse\_index\_granularity** subranges and searches the required keys there recursively

### Possible values

.Any positive even integer

**.Default value:** 8

## merge\_tree\_max\_rows\_to\_use\_cache

If ClickHouse should read more than **merge\_tree\_max\_rows\_to\_use\_cache** rows in one query, it does not use the cash of uncompressed blocks. The **uncompressed\_cache\_size** server setting defines the size of the cache of uncompressed blocks

### Possible values

.Any positive integer

**.Default value:** 1048576

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

.The minimum data volume required for using direct I/O access to the storage disk

ClickHouse uses this setting when reading data from tables. If the total storage volume of all the data to be read exceeds **min\_bytes\_to\_use\_direct\_io** bytes, then ClickHouse reads the data from the storage disk with the **O\_DIRECT** option

### Possible values

- .Direct I/O is disabled — 0
- .Positive integer

- 
-



**.Default value:** 0

## log\_queries

.Setting up query logging

Queries sent to ClickHouse with this setup are logged according to the rules in the [query\\_log](#) server configuration parameter

### :Example

```
log_queries=1
```

## max\_insert\_block\_size

.The size of blocks to form for insertion into a table

.This setting only applies in cases when the server forms the blocks

For example, for an INSERT via the HTTP interface, the server parses the data format and forms blocks of the specified size

But when using clickhouse-client, the client parses the data itself, and the 'max\_insert\_block\_size' setting on the server doesn't affect the size of the inserted blocks

The setting also doesn't have a purpose when using INSERT SELECT, since data is inserted using the same blocks that are formed after SELECT

.Default value: 1,048,576

The default is slightly more than [max\\_block\\_size](#). The reason for this is because certain table engines ([\\*MergeTree](#)) form a data part on the disk for each inserted block, which is a fairly large entity. Similarly, [\\*MergeTree](#) tables sort data during insertion, and a large enough block size allows sorting more data in RAM

## max\_replica\_delay\_for\_distributed\_queries

."Disables lagging replicas for distributed queries. See "[Replication](#)"

.Sets the time in seconds. If a replica lags more than the set value, this replica is not used

.Default value: 300

.Used when performing [SELECT](#) from a distributed table that points to replicated tables

## max\_threads

The maximum number of query processing threads, excluding threads for retrieving data from remote servers ((see the 'max\_distributed\_connections' parameter

.This parameter applies to threads that perform the same stages of the query processing pipeline in parallel

For example, when reading from a table, if it is possible to evaluate expressions with functions, filter with WHERE and pre-aggregate for GROUP BY in parallel using at least 'max\_threads' number of threads, then 'max\_threads' are used

.Default value: 2

If less than one SELECT query is normally run on a server at a time, set this parameter to a value slightly less than the actual number of processor cores

For queries that are completed quickly because of a LIMIT, you can set a lower 'max\_threads'. For example, if the necessary number of entries are located in every block and max\_threads = 8, then 8 blocks are retrieved, although it would have been enough to read just one

.The smaller the [max\\_threads](#) value, the less memory is consumed

## max\_compress\_block\_size

The maximum size of blocks of uncompressed data before compressing for writing to a table. By default, 1,048,576 (1 MiB). If the size is reduced, the compression rate is significantly reduced, the compression and decompression speed increases slightly due to cache locality, and memory consumption is reduced. There usually isn't any reason to change this setting

Don't confuse blocks for compression (a chunk of memory consisting of bytes) with blocks for query processing (a set of rows from a table)

## min\_compress\_block\_size

For **MergeTree** tables. In order to reduce latency when processing queries, a block is compressed when writing the next mark if its size is at least 'min\_compress\_block\_size'. By default, 65,536

The actual size of the block, if the uncompressed data is less than 'max\_compress\_block\_size', is no less than this value and no less than the volume of data for one mark

Let's look at an example. Assume that 'index\_granularity' was set to 8192 during table creation

We are writing a UInt32-type column (4 bytes per value). When writing 8192 rows, the total will be 32 KB of data. Since min\_compress\_block\_size = 65,536, a compressed block will be formed for every two marks

We are writing a URL column with the String type (average size of 60 bytes per value). When writing 8192 rows, the average will be slightly less than 500 KB of data. Since this is more than 65,536, a compressed block will be formed for each mark. In this case, when reading data from the disk in the range of a single mark, extra data won't be decompressed

There usually isn't any reason to change this setting

## max\_query\_size

The maximum part of a query that can be taken to RAM for parsing with the SQL parser

The INSERT query also contains data for INSERT that is processed by a separate stream parser (that consumes O(1) RAM), which is not included in this restriction

Default value: 256 KiB

## interactive\_delay

The interval in microseconds for checking whether request execution has been canceled and sending the progress

(Default value: 100,000 (checks for canceling and sends the progress ten times per second)

## connect\_timeout, receive\_timeout, send\_timeout

Timeouts in seconds on the socket used for communicating with the client

Default value: 10, 300, 300

## poll\_interval

Lock in a wait loop for the specified number of seconds

Default value: 10

## max\_distributed\_connections

The maximum number of simultaneous connections with remote servers for distributed processing of a single query to a single Distributed table. We recommend setting a value no less than the number of servers in the cluster

Default value: 1024

The following parameters are only used when creating Distributed tables (and when launching a server), so there is no reason to change them at runtime

## distributed\_connections\_pool\_size

The maximum number of simultaneous connections with remote servers for distributed processing of all queries to a single Distributed table. We recommend setting a value no less than the number of servers in the cluster

.Default value: 1024

## connect\_timeout\_with\_failover\_ms

The timeout in milliseconds for connecting to a remote server for a Distributed table engine, if the 'shard' and 'replica' sections are used in the cluster definition

.If unsuccessful, several attempts are made to connect to various replicas

.Default value: 50

## connections\_with\_failover\_max\_tries

.The maximum number of connection attempts with each replica for the Distributed table engine

.Default value: 3

## extremes

Whether to count extreme values (the minimums and maximums in columns of a query result). Accepts 0 or 1. By default, 0 (disabled)

."For more information, see the section "Extreme values"

## use\_uncompressed\_cache

.(Whether to use a cache of uncompressed blocks. Accepts 0 or 1. By default, 0 (disabled)

Using the uncompressed cache (only for tables in the MergeTree family) can significantly reduce latency and increase throughput when working with a large number of short queries. Enable this setting for users who send frequent short requests. Also pay attention to the [uncompressed\\_cache\\_size](#) configuration parameter (only set in the config file) – the size of uncompressed cache blocks. By default, it is 8 GiB. The uncompressed cache is filled in as needed and the least-used data is automatically deleted

For queries that read at least a somewhat large volume of data (one million rows or more), the uncompressed cache is disabled automatically in order to save space for truly small queries. This means that you can keep the 'use\_uncompressed\_cache' setting always set to 1

## replace\_running\_query

When using the HTTP interface, the 'query\_id' parameter can be passed. This is any string that serves as the query identifier

If a query from the same user with the same 'query\_id' already exists at this time, the behavior depends on the 'replace\_running\_query' parameter

default) – Throw an exception (don't allow the query to run if a query with the same 'query\_id' is already) 0  
.(running

.Cancel the old query and start running the new one – 1

Yandex.Metrica uses this parameter set to 1 for implementing suggestions for segmentation conditions. After entering the next character, if the old query hasn't finished yet, it should be canceled

## schema

This parameter is useful when you are using formats that require a schema definition, such as [Cap'n Proto](#). The value depends on the format

# stream\_flush\_interval\_ms

.Works for tables with streaming in the case of a timeout, or when a thread generates **max\_insert\_block\_size** rows

.The default value is 7500

The smaller the value, the more often data is flushed into the table. Setting the value too low leads to poor performance

## load\_balancing

.Specifies the algorithm of replicas selection that is used for distributed query processing

:ClickHouse supports the following algorithms of choosing replicas

(**Random** (by default

**Nearest hostname**

**In order**

**First or random**

- 
- 
- 
- 

### (Random (by default

```
load_balancing = random
```

The number of errors is counted for each replica. The query is sent to the replica with the fewest errors, and if there are several of these, to any one of them

Disadvantages: Server proximity is not accounted for; if the replicas have different data, you will also get different data

### Nearest Hostname

```
load_balancing = nearest_hostname
```

The number of errors is counted for each replica. Every 5 minutes, the number of errors is integrally divided by 2. Thus, the number of errors is calculated for a recent time with exponential smoothing. If there is one replica with a minimal number of errors (i.e. errors occurred recently on the other replicas), the query is sent to it. If there are multiple replicas with the same minimal number of errors, the query is sent to the replica with a host name that is most similar to the server's host name in the config file (for the number of different characters in identical positions, up to the minimum length of both host names)

For instance, example01-01-1 and example01-01-2.yandex.ru are different in one position, while example01-01-1 and example01-02-2 differ in two places

This method might seem primitive, but it doesn't require external data about network topology, and it doesn't compare IP addresses, which would be complicated for our IPv6 addresses

.Thus, if there are equivalent replicas, the closest one by name is preferred

We can also assume that when sending a query to the same server, in the absence of failures, a distributed query will also go to the same servers. So even if different data is placed on the replicas, the query will return mostly the same results

### In Order

```
load_balancing = in_order
```

.Replicas with the same number of errors are accessed in the same order as they are specified in configuration  
.This method is appropriate when you know exactly which replica is preferable

### First or Random

```
load_balancing = first_or_random
```

This algorithm chooses the first replica in the set or a random replica if the first is unavailable. It's effective in cross-replication topology setups, but useless in other configurations

The `first_or_random` algorithm solves the problem of the `in_order` algorithm. With `in_order`, if one replica goes down, the next one gets a double load while the remaining replicas handle the usual amount of traffic. When using the `first_or_random` algorithm, load is evenly distributed among replicas that are still available

## prefer\_localhost\_replica

.Enables/disables preferable using the localhost replica when processing distributed queries

:Possible values

.ClickHouse always sends a query to the localhost replica if it exists — 1

.ClickHouse uses the balancing strategy specified by the `load_balancing` setting — 0

.Default value: 1

Warning

.Disable this setting if you use `max_parallel_replicas`

## totals\_mode

How to calculate TOTALS when HAVING is present, as well as when `max_rows_to_group_by` and

`group_by_overflow_mode = 'any'` are present

."See the section "WITH TOTALS modifier

## totals\_auto\_threshold

.'The threshold for `totals_mode = 'auto`

."See the section "WITH TOTALS modifier

## max\_parallel\_replicas

.The maximum number of replicas for each shard when executing a query

.For consistency (to get different parts of the same data split), this option only works when the sampling key is set

.Replica lag is not controlled

## compile

.(Enable compilation of queries. By default, 0 (disabled

.(Compilation is only used for part of the query-processing pipeline: for the first stage of aggregation (GROUP BY  
If this portion of the pipeline was compiled, the query may run faster due to deployment of short cycles and  
inlining aggregate function calls. The maximum performance improvement (up to four times faster in rare cases)  
is seen for queries with multiple simple aggregate functions. Typically, the performance gain is insignificant. In  
.very rare cases, it may slow down query execution

## min\_count\_to\_compile

.How many times to potentially use a compiled chunk of code before running compilation. By default, 3

For testing, the value can be set to 0: compilation runs synchronously and the query waits for the end of the  
compilation process before continuing execution. For all other cases, use values starting with 1. Compilation  
.normally takes about 5-10 seconds

If the value is 1 or more, compilation occurs asynchronously in a separate thread. The result will be used as soon  
.as it is ready, including queries that are currently running

Compiled code is required for each different combination of aggregate functions used in the query and the type of  
.keys in the GROUP BY clause

The results of compilation are saved in the build directory in the form of .so files. There is no restriction on the  
number of compilation results, since they don't use very much space. Old results will be used after server restarts,  
.except in the case of a server upgrade – in this case, the old results are deleted

## output\_format\_json\_quote\_64bit\_integers

If the value is true, integers appear in quotes when using JSON\* Int64 and UInt64 formats (for compatibility with .most JavaScript implementations); otherwise, integers are output without the quotes

## format\_csv\_delimiter

., The character interpreted as a delimiter in the CSV data. By default, the delimiter is

## insert\_quorum

.Enables quorum writes

.If `insert_quorum < 2`, the quorum writes are disabled

.If `insert_quorum >= 2`, the quorum writes are enabled

.Default value: 0

### Quorum writes

`INSERT` succeeds only when ClickHouse manages to correctly write data to the `insert_quorum` of replicas during the `insert_quorum_timeout`. If for any reason the number of replicas with successful writes does not reach the `insert_quorum`, the write is considered failed and ClickHouse will delete the inserted block from all the replicas .where data has already been written

All the replicas in the quorum are consistent, i.e., they contain data from all previous `INSERT` queries. The `INSERT` .sequence is linearized

.When reading the data written from the `insert_quorum`, you can use the `select_sequential_consistency` option

### ClickHouse generates an exception

.If the number of available replicas at the time of the query is less than the `insert_quorum`

At an attempt to write data when the previous block has not yet been inserted in the `insert_quorum` of replicas.

This situation may occur if the user tries to perform an `INSERT` before the previous one with the `insert_quorum`

.is completed

### :See also the following parameters

`insert_quorum_timeout`

`select_sequential_consistency`

## insert\_quorum\_timeout

Quorum write timeout in seconds. If the timeout has passed and no write has taken place yet, ClickHouse will generate an exception and the client must repeat the query to write the same block to the same or any other .replica

.Default value: 60 seconds

### :See also the following parameters

`insert_quorum`

`select_sequential_consistency`

## select\_sequential\_consistency

:Enables or disables sequential consistency for `SELECT` queries

:Possible values

.Disabled — 0

.Enabled — 1

.Default value: 0

## Usage

When sequential consistency is enabled, ClickHouse allows the client to execute the **SELECT** query only for those replicas that contain data from all previous **INSERT** queries executed with **insert\_quorum**. If the client refers to a partial replica, ClickHouse will generate an exception. The **SELECT** query will not include data that has not yet been written to the quorum of replicas

## See Also

[insert\\_quorum](#)

[insert\\_quorum\\_timeout](#)

- 
- 

## max\_network\_bytes

Limits the data volume (in bytes) that is received or transmitted over the network when executing a query. This setting applies for every individual query

:Possible values

.Positive integer

.Data volume control is disabled — 0

- 
- 

.Default value: 0

## max\_network\_bandwidth

Limits speed of data exchange over the network in bytes per second. This setting applies for every individual query

:Possible values

.Positive integer

.Bandwidth control is disabled — 0

- 
- 

.Default value: 0

## max\_network\_bandwidth\_for\_user

Limits speed of data exchange over the network in bytes per second. This setting applies for all concurrently running queries performed by a single user

:Possible values

.Positive integer

.Control of the data speed is disabled — 0

- 
- 

.Default value: 0

## max\_network\_bandwidth\_for\_all\_users

Limits speed of data exchange over the network in bytes per second. This setting applies for all concurrently running queries on the server

:Possible values

.Positive integer

.Control of the data speed is disabled — 0

- 
- 

.Default value: 0

## allow\_experimental\_cross\_to\_join\_conversion

:Enables or disables

- Rewriting of queries with multiple **JOIN clauses** from the syntax with commas to the **JOIN ON/USING** syntax. If the setting value is 0, ClickHouse doesn't process queries with the syntax with commas, and throws an exception. .1
- Converting of **CROSS JOIN** into **INNER JOIN** if conditions of join allow it .2

:Possible values

.Disabled — 0

.Enabled — 1

.Default value: 1

## count\_distinct\_implementation

.Specifies which of the **uniq\*** functions should be used for performing the **COUNT(DISTINCT ...)** construction

:Possible values

**uniq**

**uniqCombined**

**uniqHLL12**

**uniqExact**

.Default value: **uniqExact**

## Settings profiles

.A settings profile is a collection of settings grouped under the same name. Each ClickHouse user has a profile

.To apply all the settings in a profile, set the **profile** setting

:Example

.Install the **web** profile

```
SET profile = 'web'
```

.Settings profiles are declared in the user config file. This is usually **users.xml**

:Example



```

<-- Settings profiles --!>
<profiles>
<-- Default settings --!>
<default>
<-- .The maximum number of threads when running a single query --!>
<max_threads>8</max_threads>
<default/>

<-- Settings for queries from the user interface --!>
<web>
<max_rows_to_read>1000000000</max_rows_to_read>
<max_bytes_to_read>100000000000</max_bytes_to_read>

<max_rows_to_group_by>1000000</max_rows_to_group_by>
<group_by_overflow_mode>any</group_by_overflow_mode>

<max_rows_to_sort>1000000</max_rows_to_sort>
<max_bytes_to_sort>10000000000</max_bytes_to_sort>

<max_result_rows>100000</max_result_rows>
<max_result_bytes>100000000</max_result_bytes>
<result_overflow_mode>break</result_overflow_mode>

<max_execution_time>600</max_execution_time>
<min_execution_speed>1000000</min_execution_speed>
<timeout_before_checking_execution_speed>15</timeout_before_checking_execution_speed>

<max_columns_to_read>25</max_columns_to_read>
<max_temporary_columns>100</max_temporary_columns>
<max_temporary_non_const_columns>50</max_temporary_non_const_columns>

<max_subquery_depth>2</max_subquery_depth>
<max_pipeline_depth>25</max_pipeline_depth>
<max_ast_depth>50</max_ast_depth>
<max_ast_elements>100</max_ast_elements>

<readonly>1</readonly>
<web/>
</profiles>

```

The example specifies two profiles: **default** and **web**. The **default** profile has a special purpose: it must always be present and is applied when starting the server. In other words, the **default** profile contains default settings. The **.web** profile is a regular profile that can be set using the **SET** query or using a URL parameter in an HTTP query

Settings profiles can inherit from each other. To use inheritance, indicate the **profile** setting before the other .settings that are listed in the profile

## Constraints on Settings

The constraints on settings can be defined in the **users** section of the **user.xml** configuration file and prohibit users .from changing some of the settings with the **SET** query

:The constraints are defined as following

```

<profiles>
<user_name>
<constraints>
<setting_name_1>
<min>lower_boundary</min>
<setting_name_1/>
<setting_name_2>
<max>upper_boundary</max>
<setting_name_2/>
<setting_name_3>
<min>lower_boundary</min>
<max>upper_boundary</max>
<setting_name_3/>
<setting_name_4>
</readonly>
<setting_name_4/>
<constraints/>
<user_name/>
</profiles>

```

.If user tries to violate the constraints an exception is thrown and the setting isn't actually changed  
There are supported three types of constraints: **min**, **max**, **readonly**. The **min** and **max** constraints specify upper and lower boundaries for a numeric setting and can be used in combination. The **readonly** constraint specify that the .user cannot change the corresponding setting at all

**:Example:** Let **users.xml** includes lines

```

<profiles>
<default>
<max_memory_usage>10000000000</max_memory_usage>
<force_index_by_date>0</force_index_by_date>
...
<constraints>
<max_memory_usage>
<min>5000000000</min>
<max>20000000000</max>
<max_memory_usage/>
<force_index_by_date>
</readonly>
<force_index_by_date/>
<constraints/>
<default/>
</profiles>

```

:The following queries all throw exceptions

```

;SET max_memory_usage=20000000001
;SET max_memory_usage=4999999999
;SET force_index_by_date=1

```

```

.Code: 452, e.displayText() = DB::Exception: Setting max_memory_usage should not be greater than 20000000000
.Code: 452, e.displayText() = DB::Exception: Setting max_memory_usage should not be less than 5000000000
.Code: 452, e.displayText() = DB::Exception: Setting force_index_by_date should not be changed

```

**Note:** the **default** profile has a special handling: all the constraints defined for the **default** profile become the default .constraints, so they restrict all the users until they're overridden explicitly for these users

## ClickHouse Utility

- **clickhouse-local** — Allows running SQL queries on data without stopping the ClickHouse server, similar to how **.awk** does this
- **clickhouse-copier** — Copies (and reshard) data from one cluster to another cluster

# clickhouse-copier

.Copies data from the tables in one cluster to tables in another (or the same) cluster

You can run multiple `clickhouse-copier` instances on different servers to perform the same job. ZooKeeper is used for .syncing the processes

:After starting, `clickhouse-copier`

:Connects to ZooKeeper and receives

.Copying jobs

.The state of the copying jobs

.It performs the jobs

Each running process chooses the "closest" shard of the source cluster and copies the data into the .destination cluster, resharding the data if necessary

.`clickhouse-copier` tracks the changes in ZooKeeper and applies them on the fly

To reduce network traffic, we recommend running `clickhouse-copier` on the same server where the source data is .located

## Running clickhouse-copier

:The utility should be run manually

```
clickhouse-copier copier --daemon --config zookeeper.xml --task-path /task/path --base-dir /path/to/dir
```

:Parameters

.`daemon` — Starts `clickhouse-copier` in daemon mode

.`config` — The path to the `zookeeper.xml` file with the parameters for the connection to ZooKeeper

`task-path` — The path to the ZooKeeper node. This node is used for syncing `clickhouse-copier` processes and .storing tasks. Tasks are stored in `$task-path/description`

.`task-file` — Optional path to file with task configuration for initial upload to ZooKeeper

.`task-upload-force` — Force upload `task-file` even if node already exists

`base-dir` — The path to logs and auxiliary files. When it starts, `clickhouse-copier` creates `clickhouse-copier_YYYYMMHHSS_<PID>` subdirectories in `$base-dir`. If this parameter is omitted, the directories are created in .the directory where `clickhouse-copier` was launched

## Format of zookeeper.xml

```
<yandex>
<logger>
<level>trace</level>
<size>100M</size>
<count>3</count>
</logger>

<zookeeper>
<"node index="1>
<host>127.0.0.1</host>
<port>2181</port>
</node>
</zookeeper>
</yandex>
```

## Configuration of copying tasks

```
<yandex>
<!-- Configuration of clusters as in an ordinary server config --!>
<remote_servers>
<source cluster>
```

```

<source_cluster>
<shard>
<internal_replication>false</internal_replication>
<replica>
<host>127.0.0.1</host>
<port>9000</port>
<replica/>
<shard/>
...
<source_cluster/>

<destination_cluster>
...
<destination_cluster/>
<remote_servers/>

<-- .How many simultaneously active workers are possible. If you run more workers superfluous workers will sleep --!>
<max_workers>2</max_workers>

<-- Setting used to fetch (pull) data from source cluster tables --!>
<settings_pull>
<readonly>1</readonly>
<settings_pull/>

<-- Setting used to insert (push) data to destination cluster tables --!>
<settings_push>
<readonly>0</readonly>
<settings_push/>

.Common setting for fetch (pull) and insert (push) operations. Also, copier process context uses it --!>
<-- .They are overlaid by <settings_pull/> and <settings_push/> respectively
<settings>
<connect_timeout>3</connect_timeout>
<-- .Sync insert is set forcibly, leave it here just in case --!>
<insert_distributed_sync>1</insert_distributed_sync>
<settings/>

.Copying tasks description --!>
You could specify several table task in the same task description (in the same ZooKeeper node), they will be performed sequentially
<--
<tables>
<-- .A table task, copies one table --!>
<table_hits>
<-- Source cluster name (from <remote_servers/> section) and tables in it that should be copied --!>
<cluster_pull>source_cluster</cluster_pull>
<database_pull>test</database_pull>
<table_pull>hits</table_pull>

<-- Destination cluster name and tables in which the data should be inserted --!>
<cluster_push>destination_cluster</cluster_push>
<database_push>test</database_push>
<table_push>hits2</table_push>

.Engine of destination tables --!>
If destination tables have not be created, workers create them using columns definition from source tables and engine definition from here

NOTE: If the first worker starts insert data and detects that destination partition is not empty then the partition will be dropped and refilled, take it into account if you already have some data in destination tables. You could directly specify partitions that should be copied in <enabled_partitions/>, they should be in quoted format like partition column of
.system.parts table
<--
<engine>

```

```
{ENGINE=ReplicatedMergeTree('/clickhouse/tables/{cluster}/{shard}/hits2', '{replica
(PARTITION BY toMonday(date
(ORDER BY (CounterID, EventDate
<engine/>

<-- Sharding key used to insert data to destination cluster --!>
<sharding_key>jumpConsistentHash(intHash64(UserID), 2)</sharding_key>

<-- Optional expression that filter data while pull them from source servers --!>
<where_condition>CounterID != 0</where_condition>

.This section specifies partitions that should be copied, other partition will be ignored --!>
Partition names should have the same format as
.(partition column of system.parts table (i.e. a quoted text
,Since partition key of source and destination cluster could be different
.these partition names specify destination partitions

,(NOTE: In spite of this section is optional (if it is not specified, all partitions will be copied
.it is strictly recommended to specify them explicitly
If you already have some ready paritions on destination cluster they
will be removed at the start of the copying since they will be interpeted
!!!as unfinished data from the previous copying
<--
<enabled_partitions>
<partition>'2018-02-26'</partition>
<partition>'2018-03-05'</partition>
...
<enabled_partitions/>
<table_hits/>

<-- .Next table to copy. It is not copied until previous table is copying --!>
<table_visits/>
...
<table_visits/>
...
<tables/>
<yandex/>
```

**clickhouse-copier** tracks the changes in **/task/path/description** and applies them on the fly. For instance, if you change .the value of **max\_workers**, the number of processes running tasks will also change

## clickhouse-local

The **clickhouse-local** program enables you to perform fast processing on local files, without having to deploy and .configure the ClickHouse server

.Accepts data that represent tables and queries them using **ClickHouse SQL dialect**

**clickhouse-local** uses the same core as ClickHouse server, so it supports most of the features and the same set of .formats and table engines

By default **clickhouse-local** does not have access to data on the same host, but it supports loading server .configuration using **--config-file** argument

### Warning

It is not recommended to load production server configuration into **clickhouse-local** because data can be damaged in .case of human error

## Usage

:Basic usage

```
"clickhouse-local --structure "table_structure" --input-format "format_of_incoming_data" -q "query"
```

:Arguments

- `.S, --structure` — table structure for input data-
- `.if, --input-format` — input format, `TSV` by default-
- `.f, --file` — path to data, `stdin` by default-
- `.q --query` — queries to execute with `;` as delimiter-
- `.N, --table` — table name where to put output data, `table` by default-
- `.of, --format, --output-format` — output format, `TSV` by default-
- `.stacktrace` — whether to dump debug output in case of exception--
- `.verbose` — more details on query execution--
- `.s` — disables `stderr` logging-
- `config-file` — path to configuration file in same format as for ClickHouse server, by default the configuration--
- `.empty`
- `.help` — arguments references for `clickhouse-local--`

Also there are arguments for each ClickHouse configuration variable which are more commonly used instead of `--config-file`

## Examples

```
echo -e "1,2\n3,4" | clickhouse-local -S "a Int64, b Int64" -if "CSV" -q "SELECT * FROM table"
.Read 2 rows, 32.00 B in 0.000 sec., 5182 rows/sec., 80.97 KiB/sec
2 1
4 3
```

:Previous example is the same as

```
echo -e "1,2\n3,4" | clickhouse-local -q "CREATE TABLE table (a Int64, b Int64) ENGINE = File(CSV, stdin); SELECT a, b FROM $"
"table; DROP TABLE table"
.Read 2 rows, 32.00 B in 0.000 sec., 4987 rows/sec., 77.93 KiB/sec
2 1
4 3
```

:Now let's output memory user for each Unix user

```
ps aux | tail -n +2 | awk '{ printf("%s\t%s\n", $1, $4) }' | clickhouse-local -S "user String, mem Float64" -q "SELECT user, $"
"round(sum(mem), 2) as memTotal FROM table GROUP BY user ORDER BY memTotal DESC FORMAT Pretty"
.Read 186 rows, 4.15 KiB in 0.035 sec., 5302 rows/sec., 118.34 KiB/sec
+-----+
| user   | memTotal |
+-----+
| bayonet | 113.5    |
+-----+
| root   | 8.8      |
+-----+
...
```

## General Questions

### ?Why Not Use Something Like MapReduce

We can refer to systems like MapReduce as distributed computing systems in which the reduce operation is based on distributed sorting. The most common open source solution in this class is [Apache Hadoop](#). Yandex uses their in-house solution, YT

These systems aren't appropriate for online queries due to their high latency. In other words, they can't be used as the back-end for a web interface

These types of systems aren't useful for real-time data updates

Distributed sorting isn't the best way to perform reduce operations if the result of the operation and all the intermediate results (if there are any) are located in the RAM of a single server, which is usually the case for online queries. In such a case, a hash table is the optimal way to perform reduce operations. A common approach to optimizing map-reduce tasks is pre-aggregation (partial reduce) using a hash table in RAM. The user performs this optimization manually

Distributed sorting is one of the main causes of reduced performance when running simple map-reduce tasks

Most MapReduce implementations allow you to execute arbitrary code on a cluster. But a declarative query language is better suited to OLAP in order to run experiments quickly. For example, Hadoop has Hive and Pig. Also consider Cloudera Impala or Shark (outdated) for Spark, as well as Spark SQL, Presto, and Apache Drill. Performance when running such tasks is highly sub-optimal compared to specialized systems, but relatively high latency makes it unrealistic to use these systems as the backend for a web interface

## What If I Have a Problem with Encodings When Using Oracle Through ODBC

If you use Oracle through the ODBC driver as a source of external dictionaries, you need to set the correct value for the `NLS_LANG` environment variable in `/etc/default/clickhouse`. For more information, see the [Oracle NLS\\_LANG FAQ](#)

### Example

```
NLS_LANG=RUSSIAN_RUSSIA.UTF8
```

## ClickHouse Development

### Overview of ClickHouse Architecture

ClickHouse is a true column-oriented DBMS. Data is stored by columns, and during the execution of arrays (vectors or chunks of columns). Whenever possible, operations are dispatched on arrays, rather than on individual values. This is called "vectorized query execution," and it helps lower the cost of actual data processing

This idea is nothing new. It dates back to the `APL` programming language and its descendants: `A +`, `J`, `K`, and `Q`. Array programming is used in scientific data processing. Neither is this idea something new in relational databases: for example, it is used in the `Vectorwise` system

There are two different approaches for speeding up the query processing: vectorized query execution and runtime code generation. In the latter, the code is generated for every kind of query on the fly, removing all indirection and dynamic dispatch. Neither of these approaches is strictly better than the other. Runtime code generation can be better when it fuses many operations together, thus fully utilizing CPU execution units and the pipeline. Vectorized query execution can be less practical, because it involves temporary vectors that must be written to the cache and read back. If the temporary data does not fit in the L2 cache, this becomes an issue. But vectorized query execution more easily utilizes the SIMD capabilities of the CPU. A [research paper](#) written by our friends shows that it is better to combine both approaches. ClickHouse uses vectorized query execution and has limited initial support for runtime code generation

## Columns

To represent columns in memory (actually, chunks of columns), the `IColumn` interface is used. This interface provides helper methods for implementation of various relational operators. Almost all operations are immutable: they do not modify the original column, but create a new modified one. For example, the `IColumn::filter` method accepts a filter byte mask. It is used for the `WHERE` and `HAVING` relational operators. Additional examples: the `IColumn::permute` method to support `ORDER BY`, the `IColumn::cut` method to support `LIMIT`, and so on

Various `IColumn` implementations (`ColumnUInt8`, `ColumnString` and so on) are responsible for the memory layout of columns. Memory layout is usually a contiguous array. For the integer type of columns it is just one contiguous array, like `std::vector`. For `String` and `Array` columns, it is two vectors: one for all array elements, placed contiguously, and a second one for offsets to the beginning of each array. There is also `ColumnConst` that stores just one value in memory, but looks like a column

## Field

Nevertheless, it is possible to work with individual values as well. To represent an individual value, the `Field` is used. `Field` is just a discriminated union of `UInt64`, `Int64`, `Float64`, `String` and `Array`. `IColumn` has the `operator[]` method to get the n-th value as a `Field`, and the `insert` method to append a `Field` to the end of a column. These methods are not very efficient, because they require dealing with temporary `Field` objects representing an individual value. There are more efficient methods, such as `insertFrom`, `insertRangeFrom`, and so on

`Field` doesn't have enough information about a specific data type for a table. For example, `UInt8`, `UInt16`, `UInt32`, and `UInt64` are all represented as `UInt64` in a `Field`

## Leaky Abstractions

`IColumn` has methods for common relational transformations of data, but they don't meet all needs. For example, `ColumnUInt64` doesn't have a method to calculate the sum of two columns, and `ColumnString` doesn't have a method to run a substring search. These countless routines are implemented outside of `IColumn`

Various functions on columns can be implemented in a generic, non-efficient way using `IColumn` methods to extract `Field` values, or in a specialized way using knowledge of inner memory layout of data in a specific `IColumn` implementation. To do this, functions are cast to a specific `IColumn` type and deal with internal representation directly. For example, `ColumnUInt64` has the `getData` method that returns a reference to an internal array, then a separate routine reads or fills that array directly. In fact, we have "leaky abstractions" to allow efficient specializations of various routines

## Data Types

`IDataType` is responsible for serialization and deserialization: for reading and writing chunks of columns or individual values in binary or text form

`IDataType` directly corresponds to data types in tables. For example, there are `DataTypeUInt32`, `DataTypeDateTime`, `DataTypeString` and so on

`IDataType` and `IColumn` are only loosely related to each other. Different data types can be represented in memory by the same `IColumn` implementations. For example, `DataTypeUInt32` and `DataTypeDateTime` are both represented by `ColumnUInt32` or `ColumnConstUInt32`. In addition, the same data type can be represented by different `IColumn` implementations. For example, `DataTypeUInt8` can be represented by `ColumnUInt8` or `ColumnConstUInt8`

`IDataType` only stores metadata. For instance, `DataTypeUInt8` doesn't store anything at all (except `vptr`) and `DataTypeFixedString` stores just `N` (the size of fixed-size strings)

`IDataType` has helper methods for various data formats. Examples are methods to serialize a value with possible quoting, to serialize a value for JSON, and to serialize a value as part of XML format. There is no direct correspondence to data formats. For example, the different data formats `Pretty` and `TabSeparated` can use the same `serializeTextEscaped` helper method from the `IDataType` interface

## Block

A `Block` is a container that represents a subset (chunk) of a table in memory. It is just a set of triples: (`IColumn`, `IDataType`, `column name`). During query execution, data is processed by `Blocks`. If we have a `Block`, we have data (in the `IColumn` object), we have information about its type (in `IDataType`) that tells us how to deal with that column, and we have the column name (either the original column name from the table, or some artificial name assigned for getting temporary results of calculations)



When we calculate some function over columns in a block, we add another column with its result to the block, and we don't touch columns for arguments of the function because operations are immutable. Later, unneeded columns can be removed from the block, but not modified. This is convenient for elimination of common .subexpressions

Blocks are created for every processed chunk of data. Note that for the same type of calculation, the column names and types remain the same for different blocks, and only column data changes. It is better to split block data from the block header, because small block sizes will have a high overhead of temporary strings for copying .shared\_ptrs and column names

## Block Streams

Block streams are for processing data. We use streams of blocks to read data from somewhere, perform data transformations, or write data to somewhere. `IBlockInputStream` has the `read` method to fetch the next block while .available. `IBlockOutputStream` has the `write` method to push the block somewhere

:Streams are responsible for

- .1 Reading or writing to a table. The table just returns a stream for reading or writing blocks
- .2 Implementing data formats. For example, if you want to output data to a terminal in `Pretty` format, you create .a block output stream where you push blocks, and it formats them
- .3 Performing data transformations. Let's say you have `IBlockInputStream` and want to create a filtered stream. You create `FilterBlockInputStream` and initialize it with your stream. Then when you pull a block from `FilterBlockInputStream`, it pulls a block from your stream, filters it, and returns the filtered block to you. Query .execution pipelines are represented this way

There are more sophisticated transformations. For example, when you pull from `AggregatingBlockInputStream`, it reads all data from its source, aggregates it, and then returns a stream of aggregated data for you. Another example: `UnionBlockInputStream` accepts many input sources in the constructor and also a number of threads. It .launches multiple threads and reads from multiple sources in parallel

Block streams use the "pull" approach to control flow: when you pull a block from the first stream, it consequently pulls the required blocks from nested streams, and the entire execution pipeline will work. Neither "pull" nor "push" is the best solution, because control flow is implicit, and that limits implementation of various features like simultaneous execution of multiple queries (merging many pipelines together). This limitation could be overcome with coroutines or just running extra threads that wait for each other. We may have more possibilities if we make control flow explicit: if we locate the logic for passing data from one calculation unit to another outside of those .calculation units. Read this [article](#) for more thoughts

We should note that the query execution pipeline creates temporary data at each step. We try to keep block size small enough so that temporary data fits in the CPU cache. With that assumption, writing and reading temporary data is almost free in comparison with other calculations. We could consider an alternative, which is to fuse many operations in the pipeline together, to make the pipeline as short as possible and remove much of the temporary data. This could be an advantage, but it also has drawbacks. For example, a split pipeline makes it easy to implement caching intermediate data, stealing intermediate data from similar queries running at the same time, .and merging pipelines for similar queries

## Formats

Data formats are implemented with block streams. There are "presentational" formats only suitable for output of data to the client, such as `Pretty` format, which provides only `IBlockOutputStream`. And there are input/output .formats, such as `TabSeparated` or `JSONEachRow`

There are also row streams: `IRowInputStream` and `IRowOutputStream`. They allow you to pull/push data by individual rows, not by blocks. And they are only needed to simplify implementation of row-oriented formats. The wrappers `BlockInputStreamFromRowInputStream` and `BlockOutputStreamFromRowOutputStream` allow you to convert row-oriented .streams to regular block-oriented streams

## I/O

For byte-oriented input/output, there are `ReadBuffer` and `WriteBuffer` abstract classes. They are used instead of C++ `iostreams`. Don't worry: every mature C++ project is using something other than `iostreams` for good reasons

`ReadBuffer` and `WriteBuffer` are just a contiguous buffer and a cursor pointing to the position in that buffer. Implementations may own or not own the memory for the buffer. There is a virtual method to fill the buffer with the following data (for `ReadBuffer`) or to flush the buffer somewhere (for `WriteBuffer`). The virtual methods are rarely called

Implementations of `ReadBuffer/WriteBuffer` are used for working with files and file descriptors and network sockets, for implementing compression (`CompressedWriteBuffer` is initialized with another `WriteBuffer` and performs compression before writing data to it), and for other purposes – the names `ConcatReadBuffer`, `LimitReadBuffer`, and `HashingWriteBuffer` speak for themselves

`Read/WriteBuffers` only deal with bytes. To help with formatted input/output (for instance, to write a number in decimal format), there are functions from `ReadHelpers` and `WriteHelpers` header files

Let's look at what happens when you want to write a result set in `JSON` format to stdout. You have a result set ready to be fetched from `IBlockInputStream`. You create `WriteBufferFromFileDescriptor(STDOUT_FILENO)` to write bytes to stdout. You create `JSONRowOutputStream`, initialized with that `WriteBuffer`, to write rows in `JSON` to stdout. You create `BlockOutputStreamFromRowOutputStream` on top of it, to represent it as `IBlockOutputStream`. Then you call `copyData` to transfer data from `IBlockInputStream` to `IBlockOutputStream`, and everything works. Internally, `JSONRowOutputStream` will write various `JSON` delimiters and call the `IDataType::serializeTextJSON` method with a reference to `IColumn` and the row number as arguments. Consequently, `IDataType::serializeTextJSON` will call a method from `WriteHelpers.h`: for example, `writeText` for numeric types and `writeJSONString` for `DataTypeString`

## Tables

Tables are represented by the `IStorage` interface. Different implementations of that interface are different table engines. Examples are `StorageMergeTree`, `StorageMemory`, and so on. Instances of these classes are just tables

The most important `IStorage` methods are `read` and `write`. There are also `alter`, `rename`, `drop`, and so on. The `read` method accepts the following arguments: the set of columns to read from a table, the `AST` query to consider, and the desired number of streams to return. It returns one or multiple `IBlockInputStream` objects and information about the stage of data processing that was completed inside a table engine during query execution

In most cases, the `read` method is only responsible for reading the specified columns from a table, not for any further data processing. All further data processing is done by the query interpreter and is outside the responsibility of `IStorage`

:But there are notable exceptions

The `AST` query is passed to the `read` method and the table engine can use it to derive index usage and to read •  
less data from a table

Sometimes the table engine can process data itself to a specific stage. For example, `StorageDistributed` can •  
send a query to remote servers, ask them to process data to a stage where data from different remote  
servers can be merged, and return that preprocessed data  
The query interpreter then finishes processing the data

The table's `read` method can return multiple `IBlockInputStream` objects to allow parallel data processing. These multiple block input streams can read from a table in parallel. Then you can wrap these streams with various transformations (such as expression evaluation or filtering) that can be calculated independently and create a `UnionBlockInputStream` on top of them, to read from multiple streams in parallel

There are also `TableFunctions`. These are functions that return a temporary `IStorage` object to use in the `FROM` clause of a query

To get a quick idea of how to implement your own table engine, look at something simple, like `StorageMemory` or `StorageTinyLog`

As the result of the `read` method, `IStorage` returns `QueryProcessingStage` – information about what parts of the query were already calculated inside storage. Currently we have only very coarse granularity for that information. There is no way for the storage to say "I have already processed this part of the expression in WHERE, for this range of .data". We need to work on that

## Parsers

A query is parsed by a hand-written recursive descent parser. For example, `ParserSelectQuery` just recursively calls the underlying parsers for various parts of the query. Parsers create an `AST`. The `AST` is represented by nodes, which are instances of `IAST`

.Parser generators are not used for historical reasons

## Interpreters

Interpreters are responsible for creating the query execution pipeline from an `AST`. There are simple interpreters, such as `InterpreterExistsQuery` and `InterpreterDropQuery`, or the more sophisticated `InterpreterSelectQuery`. The query execution pipeline is a combination of block input or output streams. For example, the result of interpreting the `SELECT` query is the `IBlockInputStream` to read the result set from; the result of the `INSERT` query is the `IBlockOutputStream` to write data for insertion to; and the result of interpreting the `INSERT SELECT` query is the `IBlockInputStream` that returns an empty result set on the first read, but that copies data from `SELECT` to `INSERT` at the same time

`InterpreterSelectQuery` uses `ExpressionAnalyzer` and `ExpressionActions` machinery for query analysis and transformations. This is where most rule-based query optimizations are done. `ExpressionAnalyzer` is quite messy and should be rewritten: various query transformations and optimizations should be extracted to separate classes to allow modular transformations or query

## Functions

.There are ordinary functions and aggregate functions. For aggregate functions, see the next section

Ordinary functions don't change the number of rows – they work as if they are processing each row independently. In fact, functions are not called for individual rows, but for `Block`'s of data to implement vectorized .query execution

There are some miscellaneous functions, like `blockSize`, `rowNumberInBlock`, and `runningAccumulate`, that exploit .block processing and violate the independence of rows

ClickHouse has strong typing, so implicit type conversion doesn't occur. If a function doesn't support a specific combination of types, an exception will be thrown. But functions can work (be overloaded) for many different combinations of types. For example, the `plus` function (to implement the `+` operator) works for any combination of numeric types: `UInt8 + Float32`, `UInt16 + Int8`, and so on. Also, some variadic functions can accept any number of arguments, such as the `concat` function

Implementing a function may be slightly inconvenient because a function explicitly dispatches supported data types and supported `IColumns`. For example, the `plus` function has code generated by instantiation of a C++ .template for each combination of numeric types, and for constant or non-constant left and right arguments

This is a nice place to implement runtime code generation to avoid template code bloat. Also, it will make it .possible to add fused functions like fused multiply-add, or to make multiple comparisons in one loop iteration

Due to vectorized query execution, functions are not short-circuit. For example, if you write `WHERE f(x) AND g(y)`, both sides will be calculated, even for rows, when `f(x)` is zero (except when `f(x)` is a zero constant expression). But if selectivity of the `f(x)` condition is high, and calculation of `f(x)` is much cheaper than `g(y)`, it's better to implement multi-pass calculation: first calculate `f(x)`, then filter columns by the result, and then calculate `g(y)` only for smaller, .filtered chunks of data

# Aggregate Functions

Aggregate functions are stateful functions. They accumulate passed values into some state, and allow you to get results from that state. They are managed with the `IAggregateFunction` interface. States can be rather simple (the state for `AggregateFunctionCount` is just a single `UInt64` value) or quite complex (the state of `AggregateFunctionUniqCombined` is a combination of a linear array, a hash table and a `HyperLogLog` probabilistic data structure).

To deal with multiple states while executing a high-cardinality `GROUP BY` query, states are allocated in `Arena` (a memory pool), or they could be allocated in any suitable piece of memory. States can have a non-trivial constructor and destructor: for example, complex aggregation states can allocate additional memory themselves. This requires some attention to creating and destroying states and properly passing their ownership, to keep track of who and when will destroy states.

Aggregation states can be serialized and deserialized to pass over the network during distributed query execution or to write them on disk where there is not enough RAM. They can even be stored in a table with the `DataTypeAggregateFunction` to allow incremental aggregation of data.

The serialized data format for aggregate function states is not versioned right now. This is ok if aggregate states are only stored temporarily. But we have the `AggregatingMergeTree` table engine for incremental aggregation, and people are already using it in production. This is why we should add support for backward compatibility when changing the serialized format for any aggregate function in the future.

## Server

The server implements several different interfaces:

- An HTTP interface for any foreign clients
- A TCP interface for the native ClickHouse client and for cross-server communication during distributed query execution
- An interface for transferring data for replication

Internally, it is just a basic multithreaded server without coroutines, fibers, etc. Since the server is not designed to process a high rate of simple queries but is intended to process a relatively low rate of complex queries, each of them can process a vast amount of data for analytics.

The server initializes the `Context` class with the necessary environment for query execution: the list of available databases, users and access rights, settings, clusters, the process list, the query log, and so on. This environment is used by interpreters.

We maintain full backward and forward compatibility for the server TCP protocol: old clients can talk to new servers and new clients can talk to old servers. But we don't want to maintain it eternally, and we are removing support for old versions after about one year.

For all external applications, we recommend using the HTTP interface because it is simple and easy to use. The TCP protocol is more tightly linked to internal data structures: it uses an internal format for passing blocks of data and it uses custom framing for compressed data. We haven't released a C library for that protocol because it requires linking most of the ClickHouse codebase, which is not practical.

## Distributed Query Execution

Servers in a cluster setup are mostly independent. You can create a `Distributed` table on one or all servers in a cluster. The `Distributed` table does not store data itself – it only provides a "view" to all local tables on multiple nodes of a cluster. When you `SELECT` from a `Distributed` table, it rewrites that query, chooses remote nodes according to load balancing settings, and sends the query to them. The `Distributed` table requests remote servers to process a query just up to a stage where intermediate results from different servers can be merged. Then it receives the intermediate results and merges them. The distributed table tries to distribute as much work as possible to remote servers, and does not send much intermediate data over the network.

Things become more complicated when you have subqueries in IN or JOIN clauses and each of them uses a **Distributed** table. We have different strategies for execution of these queries

There is no global query plan for distributed query execution. Each node has its own local query plan for its part of the job. We only have simple one-pass distributed query execution: we send queries for remote nodes and then merge the results. But this is not feasible for difficult queries with high cardinality GROUP BYs or with a large amount of temporary data for JOIN: in such cases, we need to "reshuffle" data between servers, which requires additional coordination. ClickHouse does not support that kind of query execution, and we need to work on it

## Merge Tree

**MergeTree** is a family of storage engines that supports indexing by primary key. The primary key can be an arbitrary tuple of columns or expressions. Data in a **MergeTree** table is stored in "parts". Each part stores data in the primary key order (data is ordered lexicographically by the primary key tuple). All the table columns are stored in separate **column.bin** files in these parts. The files consist of compressed blocks. Each block is usually from 64 KB to 1 MB of uncompressed data, depending on the average value size. The blocks consist of column values placed contiguously one after the other. Column values are in the same order for each column (the order is defined by the primary key), so when you iterate by many columns, you get values for the corresponding rows

The primary key itself is "sparse". It doesn't address each single row, but only some ranges of data. A separate **primary.idx** file has the value of the primary key for each N-th row, where N is called **index\_granularity** (usually, N = 8192). Also, for each column, we have **column.mrk** files with "marks," which are offsets to each N-th row in the data file. Each mark is a pair: the offset in the file to the beginning of the compressed block, and the offset in the decompressed block to the beginning of data. Usually compressed blocks are aligned by marks, and the offset in the decompressed block is zero. Data for **primary.idx** always resides in memory and data for **column.mrk** files is cached

When we are going to read something from a part in **MergeTree**, we look at **primary.idx** data and locate ranges that could possibly contain requested data, then look at **column.mrk** data and calculate offsets for where to start reading those ranges. Because of sparseness, excess data may be read. ClickHouse is not suitable for a high load of simple point queries, because the entire range with **index\_granularity** rows must be read for each key, and the entire compressed block must be decompressed for each column. We made the index sparse because we must be able to maintain trillions of rows per single server without noticeable memory consumption for the index. Also, because the primary key is sparse, it is not unique: it cannot check the existence of the key in the table at INSERT time. You could have many rows with the same key in a table

When you **INSERT** a bunch of data into **MergeTree**, that bunch is sorted by primary key order and forms a new part. To keep the number of parts relatively low, there are background threads that periodically select some parts and merge them to a single sorted part. That's why it is called **MergeTree**. Of course, merging leads to "write amplification". All parts are immutable: they are only created and deleted, but not modified. When SELECT is run, it holds a snapshot of the table (a set of parts). After merging, we also keep old parts for some time to make recovery after failure easier, so if we see that some merged part is probably broken, we can replace it with its source parts

**MergeTree** is not an LSM tree because it doesn't contain "memtable" and "log": inserted data is written directly to the filesystem. This makes it suitable only to INSERT data in batches, not by individual row and not very frequently – about once per second is ok, but a thousand times a second is not. We did it this way for simplicity's sake, and because we are already inserting data in batches in our applications

MergeTree tables can only have one (primary) index: there aren't any secondary indices. It would be nice to allow multiple physical representations under one logical table, for example, to store data in more than one physical order or even to allow representations with pre-aggregated data along with original data

There are MergeTree engines that are doing additional work during background merges. Examples are [CollapsingMergeTree](#) and [AggregatingMergeTree](#). This could be treated as special support for updates. Keep in mind that these are not real updates because users usually have no control over the time when background merges will be executed, and data in a [MergeTree](#) table is almost always stored in more than one part, not in completely merged form

## Replication

Replication in ClickHouse is implemented on a per-table basis. You could have some replicated and some non-replicated tables on the same server. You could also have tables replicated in different ways, such as one table with two-factor replication and another with three-factor

Replication is implemented in the [ReplicatedMergeTree](#) storage engine. The path in [ZooKeeper](#) is specified as a parameter for the storage engine. All tables with the same path in [ZooKeeper](#) become replicas of each other: they synchronize their data and maintain consistency. Replicas can be added and removed dynamically simply by creating or dropping a table

Replication uses an asynchronous multi-master scheme. You can insert data into any replica that has a session with [ZooKeeper](#), and data is replicated to all other replicas asynchronously. Because ClickHouse doesn't support UPDATES, replication is conflict-free. As there is no quorum acknowledgment of inserts, just-inserted data might be lost if one node fails

Metadata for replication is stored in ZooKeeper. There is a replication log that lists what actions to do. Actions are: get part; merge parts; drop partition, etc. Each replica copies the replication log to its queue and then executes the actions from the queue. For example, on insertion, the "get part" action is created in the log, and every replica downloads that part. Merges are coordinated between replicas to get byte-identical results. All parts are merged in the same way on all replicas. To achieve this, one replica is elected as the leader, and that replica initiates merges and writes "merge parts" actions to the log

Replication is physical: only compressed parts are transferred between nodes, not queries. To lower the network cost (to avoid network amplification), merges are processed on each replica independently in most cases. Large merged parts are sent over the network only in cases of significant replication lag

In addition, each replica stores its state in ZooKeeper as the set of parts and its checksums. When the state on the local filesystem diverges from the reference state in ZooKeeper, the replica restores its consistency by downloading missing and broken parts from other replicas. When there is some unexpected or broken data in the local filesystem, ClickHouse does not remove it, but moves it to a separate directory and forgets it

The ClickHouse cluster consists of independent shards, and each shard consists of replicas. The cluster is not elastic, so after adding a new shard, data is not rebalanced between shards automatically. Instead, the cluster load will be uneven. This implementation gives you more control, and it is fine for relatively small clusters such as tens of nodes. But for clusters with hundreds of nodes that we are using in production, this approach becomes a significant drawback. We should implement a table engine that will span its data across the cluster with dynamically replicated regions that could be split and balanced between clusters automatically

## How to Build ClickHouse Release Package

### Install Git and Pbuilder

```
sudo apt-get update
sudo apt-get install git pbuilder debhelper lsb-release fakeroot sudo debian-archive-keyring debian-keyring
```

### Checkout ClickHouse Sources

```
git clone --recursive --branch stable https://github.com/yandex/ClickHouse.git
cd ClickHouse
```

### Run Release Script

```
release/.
```



# How to Build ClickHouse for Development

.The following tutorial is based on the Ubuntu Linux system  
.With appropriate changes, it should also work on any other Linux distribution  
.Only x86\_64 with SSE 4.2 is supported. Support for AArch64 is experimental

To test for SSE 4.2, do

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

## Install Git and CMake

```
sudo apt-get install git cmake ninja-build
```

.Or cmake3 instead of cmake on older systems

## Install GCC 7

.There are several ways to do this

### Install from a PPA Package

```
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-7 g++-7
```

### Install from Sources

Look at [ci/build-gcc-from-sources.sh](#)

## Use GCC 7 for Builds

```
export CC=gcc-7
export CXX=g++-7
```

## Install Required Libraries from Packages

```
sudo apt-get install libicu-dev libreadline-dev
```

## Checkout ClickHouse Sources

```
git clone --recursive git@github.com:yandex/ClickHouse.git
or: git clone --recursive https://github.com/yandex/ClickHouse.git ##

cd ClickHouse
```

.For the latest stable version, switch to the [stable](#) branch

## Build ClickHouse

```
mkdir build
cd build
.. cmake
ninja
.. cd
```

.To create an executable, run [ninja clickhouse](#)  
.This will create the [dbms/programs/clickhouse](#) executable, which can be used with [client](#) or [server](#) arguments

## How to Build ClickHouse on Mac OS X

.Build should work on Mac OS X 10.12

## Install Homebrew

```
"(usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install/
```

## Install Required Compilers, Tools, and Libraries

```
brew install cmake ninja gcc icu4c openssl libtool gettext readline
```

## Checkout ClickHouse Sources

```
git clone --recursive git@github.com:yandex/ClickHouse.git  
or: git clone --recursive https://github.com/yandex/ClickHouse.git ##
```

```
cd ClickHouse
```

.For the latest stable version, switch to the **stable** branch

## Build ClickHouse

```
mkdir build  
cd build  
`cmake .. -DCMAKE_CXX_COMPILER=`which g++-8` -DCMAKE_C_COMPILER=`which gcc-8`  
ninja  
.. cd
```

## Caveats

.If you intend to run clickhouse-server, make sure to increase the system's maxfiles variable

### Note

.You'll need to use sudo

:To do so, create the following file

:Library/LaunchDaemons/limit.maxfiles.plist/

```
<?xml version="1.0" encoding="UTF-8?">  
"DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN!">  
<"http://www.apple.com/DTDs/PropertyList-1.0.dtd"  
<"plist version="1.0">  
<dict>  
<key>Label</key>  
<string>limit.maxfiles</string>  
<key>ProgramArguments</key>  
<array>  
<string>launchctl</string>  
<string>limit</string>  
<string>maxfiles</string>  
<string>524288</string>  
<string>524288</string>  
</array>  
<key>RunAtLoad</key>  
</true>  
<key>ServiceIPC</key>  
</false>  
</dict>  
</plist>
```

:Execute the following command

```
sudo chown root:wheel /Library/LaunchDaemons/limit.maxfiles.plist $
```

.Reboot

.To check if it's working, you can use **ulimit -n** command



# How to Write C++ Code

## General Recommendations

.The following are recommendations, not requirements **.1**

.If you are editing code, it makes sense to follow the formatting of the existing code **.2**

Code style is needed for consistency. Consistency makes it easier to read the code, and it also makes it easier **.3**  
.to search the code

.Many of the rules do not have logical reasons; they are dictated by established practices **.4**

## Formatting

.Most of the formatting will be done automatically by **clang-format** **.1**

.Indents are 4 spaces. Configure your development environment so that a tab adds four spaces **.2**

.Opening and closing curly brackets must be on a separate line **.3**

```
(inline void readBoolText(bool & x, ReadBuffer & buf
)
;char tmp = '0
;(readChar(tmp, buf
; 'x = tmp != '0
{
```

If the entire function body is a single **statement**, it can be placed on a single line. Place spaces around curly **.4**  
(braces (besides the space at the end of the line

```
{ ;inline size_t mask() const { return buf_size() - 1
{ ;()inline size_t place(HashValue x) const { return x & mask
```

.For functions. Don't put spaces around brackets **.5**

```
(void reinsert(const Value & x
```

```
;((memcpy(&buf[place_value], &x, sizeof(x
```

In **if**, **for**, **while** and other expressions, a space is inserted in front of the opening bracket (as opposed to function **.6**  
(calls

```
(for (size_t i = 0; i < rows; i += storage.index_granularity
```

**.7** Add spaces around binary operators (**+**, **-**, **\***, **/**, **%**, ...) and the ternary operator **.7**

```
;('UInt16 year = (s[0] - '0') * 1000 + (s[1] - '0') * 100 + (s[2] - '0') * 10 + (s[3] - '0
;('UInt8 month = (s[5] - '0') * 10 + (s[6] - '0
;('UInt8 day = (s[8] - '0') * 10 + (s[9] - '0
```

.If a line feed is entered, put the operator on a new line and increase the indent before it **.8**

```
(if (elapsed_ns
") " >> message
" ,.rows_read_on_server * 1000000000 / elapsed_ns << " rows/s >>
;" (.bytes_read_on_server * 1000.0 / elapsed_ns << " MB/s >>
```

.You can use spaces for alignment within a line, if desired **.9**

```
;dst.ClickLogID = click.LogID
;dst.ClickEventID = click.EventID
;dst.ClickGoodEvent = click.GoodEvent
```

**.10** Don't use spaces around the operators **.10**

.If necessary, the operator can be wrapped to the next line. In this case, the offset in front of it is increased

.Do not use a space to separate unary operators (--, ++, \*, &, ...) from the argument **.11**

.Put a space after a comma, but not before it. The same rule goes for a semicolon inside a **for** expression **.12**

.Do not use spaces to separate the **[]** operator **.13**

.< In a **template <...>** expression, use a space between **template** and **<**; no spaces after **<** or before **.14**

```
<template <typename TKey, typename TValue
struct AggregatedStatElement
{}
```

In classes and structures, write **public**, **private**, and **protected** on the same level as **class/struct**, and indent the rest **.15**  
.of the code

```
<template <typename T
class MultiVersion
}
:public
.Version of object for usage. shared_ptr manage lifetime of version ///
; <using Version = std::shared_ptr<const T
...
{
```

If the same **namespace** is used for the entire file, and there isn't anything else significant, an offset is not **.16**  
.necessary inside **namespace**

If the block for an **if**, **for**, **while**, or other expression consists of a single **statement**, the curly brackets are optional. **.17**  
... ,Place the **statement** on a separate line, instead. This rule is also valid for nested **if**, **for**, **while**

.But if the inner **statement** contains curly brackets or **else**, the external block should be written in curly brackets

```
.Finish write ///
(for (auto & stream : streams
;())stream.second->finalize
```

.There shouldn't be any spaces at the ends of lines **.18**

.Source files are UTF-8 encoded **.19**

.Non-ASCII characters can be used in string literals **.20**

```
,".timer.elapsed() / chunks_stats.hits) << " μsec/hit >> " , " >>
```

.Do not write multiple expressions in a single line **21**

.Group sections of code inside functions and separate them with no more than one empty line **.22**

.Separate functions, classes, and so on with one or two empty lines **.23**

.A **const** (related to a value) must be written before the type name **.24**

```
correct//
const char * pos
const std::string & s
incorrect//
char const * pos
```

.When declaring a pointer or reference, the **\*** and **&** symbols should be separated by spaces on both sides **.25**

```
correct//
const char * pos
incorrect//
const char* pos
const char *pos
```

.(When using template types, alias them with the **using** keyword (except in the simplest cases **.26**

.In other words, the template parameters are specified only in **using** and aren't repeated in the code

.**using** can be declared locally, such as inside a function

```
correct//
;<<using FileStreams = std::map<std::string, std::shared_ptr<Stream
;FileStreams streams
incorrect//
;std::map<std::string, std::shared_ptr<Stream>> streams
```

.Do not declare several variables of different types in one statement **.27**

```
incorrect//
;int x, *y
```

.Do not use C-style casts **.28**

```
incorrect//
;std::cerr << (int)c <<; std::endl
correct//
;std::cerr << static_cast<int>(c) << std::endl
```

.In classes and structs, group members and functions separately inside each visibility scope **.29**

.For small classes and structs, it is not necessary to separate the method declaration from the implementation **.30**

.The same is true for small methods in any classes or structs

For templated classes and structs, don't separate the method declarations from the implementation (because  
.(otherwise they must be defined in the same translation unit

.You can wrap lines at 140 characters, instead of 80 **.31**

.Always use the prefix increment/decrement operators if postfix is not required **.32**

```
(for (Names::const_iterator it = column_names.begin(); it != column_names.end(); ++it
```

## Comments

.Be sure to add comments for all non-trivial parts of code **.1**

This is very important. Writing the comment might help you realize that the code isn't necessary, or that it is  
.(designed wrong

```
.(Part of piece of memory, that can be used **/
.(For example, if internal_buffer is 1MB, and there was only 10 bytes loaded to buffer from file for reading *
then working_buffer will have size of only 10 bytes *
.(working_buffer.end() will point to position right after those 10 bytes available for read) *
/*
```

.Comments can be as detailed as necessary **.2**

Place comments before the code they describe. In rare cases, comments can come after the code, on the same **.3**  
.line

```
.Parses and executes the query **/
/*
void executeQuery
(ReadBuffer & istr, /// Where to read the query from (and data for INSERT, if applicable
WriteBuffer & ostr, /// Where to write the result
...Context & context, /// DB, tables, data types, engines, functions, aggregate functions
BlockInputStreamPtr & query_plan, /// Here could be written the description on how query was executed
QueryProcessingStage::Enum stage = QueryProcessingStage::Complete /// Up to which stage process the SELECT query
(
```

.Comments should be written in English only **.4**

.If you are writing a library, include detailed comments explaining it in the main header file **.5**

Do not add comments that do not provide additional information. In particular, do not leave empty comments **.6**  
:like this

```
*/
:Procedure Name *
:Original procedure name *
:Author *
:Date of creation *
:Dates of modification *
:Modification authors *
:Original file name *
:Purpose *
:Intent *
:Designation *
:Classes used *
:Constants *
:Local variables *
:Parameters *
:Date of creation *
:Purpose *
/*
```

The example is borrowed from the resource <http://home.tamk.fi/~jaalto/course/coding-style/doc/unmaintainable-./code>

.Do not write garbage comments (author, creation date ..) at the beginning of each file **.7**

Single-line comments begin with three slashes: `///` and multi-line comments begin with `/**`. These comments are **.8**  
."considered "documentation

Note: You can use Doxygen to generate documentation from these comments. But Doxygen is not generally used  
.because it is more convenient to navigate the code in the IDE

Multi-line comments must not have empty lines at the beginning and end (except the line that closes a multi- **.9**  
.(line comment

.For commenting out code, use basic comments, not “documenting” comments **.10**

.Delete the commented out parts of the code before committing **.11**

.Do not use profanity in comments or code **.12**

.Do not use uppercase letters. Do not use excessive punctuation **.13**

```
???WHAT THE FAIL ///
```

.Do not use comments to make delimiters **.14**

```
*****///
```

.Do not start discussions in comments **.15**

```
?Why did you do this stuff ///
```

.There's no need to write a comment at the end of a block describing what it was about **.16**

```
for ///
```

## Names

.Use lowercase letters with underscores in the names of variables and class members **.1**

```
;size_t max_block_size
```

.For the names of functions (methods), use camelCase beginning with a lowercase letter **.2**

```
{ ;"std::string getName() const override { return "Memory
```

For the names of classes (structs), use CamelCase beginning with an uppercase letter. Prefixes other than I are **.3**  
.not used for interfaces

```
class StorageMemory : public IStorage
```

.**using** are named the same way as classes, or with **\_t** on the end **.4**

.Names of template type arguments: in simple cases, use **T**; **T**, **U**; **T1**, **T2** **.5**

.For more complex cases, either follow the rules for class names, or add the prefix **T**

```
<template <typename TKey, typename TValue  
struct AggregatedStatElement
```

.Names of template constant arguments: either follow the rules for variable names, or use **N** in simple cases **.6**

```
<template <bool without_www  
struct ExtractDomain
```

.For abstract classes (interfaces) you can add the **I** prefix **.7**

```
class IBlockInputStream
```

.If you use a variable locally, you can use the short name **.8**

.In all other cases, use a name that describes the meaning

```
;bool info_successfully_loaded = false
```

.Names of **defines** and global constants use ALL\_CAPS with underscores **.9**

```
define MAX_SRC_TABLE_NAMES_TO_STORE 1000##
```

.File names should use the same style as their contents **.10**

.(If a file contains a single class, name the file the same way as the class (CamelCase

.(If the file contains a single function, name the file the same way as the function (camelCase

:If the name contains an abbreviation, then **.11**

.(For variable names, the abbreviation should use lowercase letters **mysql\_connection** (not **mySQL\_connection** •  
For names of classes and functions, keep the uppercase letters in the abbreviation **MySQLConnection** (not •  
.(**MySQLConnection**

Constructor arguments that are used just to initialize the class members should be named the same way as **.12**  
.the class members, but with an underscore at the end

```

)FileQueueProcessor
, _const std::string & path
, _const std::string & prefix
( _std::shared_ptr<FileHandler> handler
, (_path(path :
, (_prefix(prefix
, (_handler(handler
(("log(&Logger::get("FileQueueProcessor
}
{

```

.The underscore suffix can be omitted if the argument is not used in the constructor body

.(There is no difference in the names of local variables and class members (no prefixes required **.13**

```

(timer (not m_timer

```

For the constants in an **enum**, use CamelCase with a capital letter. ALL\_CAPS is also acceptable. If the **enum** is **.14**  
 .non-local, use an **enum class**

```

enum class CompressionMethod
}
,QuickLZ = 0
,LZ4     = 1
;{

```

.All names must be in English. Transliteration of Russian words is not allowed **.15**

```

not Stroka

```

Abbreviations are acceptable if they are well known (when you can easily find the meaning of the abbreviation **.16**  
 .(in Wikipedia or in a search engine

```

.`AST`, `SQL`

```

```

(Not `NVDH` (some random letters

```

.Incomplete words are acceptable if the shortened version is common use

.You can also use an abbreviation if the full name is included next to it in the comments

.File names with C++ source code must have the **.cpp** extension. Header files must have the **.h** extension **.17**

## How to Write Code

.Memory management **.1**

.Manual memory deallocation (**delete**) can only be used in library code

.In library code, the **delete** operator can only be used in destructors

.In application code, memory must be freed by the object that owns it

:Examples

.The easiest way is to place an object on the stack, or make it a member of another class

.For a large number of small objects, use containers

.For automatic deallocation of a small number of objects that reside in the heap, use **shared\_ptr/unique\_ptr**

.Resource management **.2**

.Use **RAII** and see above

.Error handling **.3**

.(Use exceptions. In most cases, you only need to throw an exception, and don't need to catch it (because of **RAII**

.In offline data processing applications, it's often acceptable to not catch exceptions

In servers that handle user requests, it's usually enough to catch exceptions at the top level of the connection handler

.In thread functions, you should catch and keep all exceptions to rethrow them in the main thread after **join**

```
If there weren't any calculations yet, calculate the first block synchronously ///
(if (!started
}
;())calculate
;started = true
{
else /// If calculations are already in progress, wait for the result
;()pool.wait

(if (exception
;()exception->rethrow
```

.Never hide exceptions without handling. Never just blindly put all exceptions to log

```
Not correct//
{} (...) catch
```

.If you need to ignore some exceptions, do so only for specific ones and rethrow the rest

```
{catch (const DB::Exception & e
}
(if (e.code() == ErrorCodes::UNKNOWN_AGGREGATE_FUNCTION
;return nullptr
else
;throw
{
```

When using functions with response codes or **errno**, always check the result and throw an exception in case of error

```
((if (0 != close(fd
;(throwFromErrno("Cannot close file " + file_name, ErrorCodes::CANNOT_CLOSE_FILE
```

.**Do not use assert**

.Exception types **.4**

There is no need to use complex exception hierarchy in application code. The exception text should be understandable to a system administrator

.Throwing exceptions from destructors **.5**

.This is not recommended, but it is allowed

:Use the following options

Create a function (**done()** or **finalize()**) that will do all the work in advance that might lead to an exception. If •  
.that function was called, there should be no exceptions in the destructor later  
Tasks that are too complex (such as sending messages over the network) can be put in separate method that •  
.the class user will have to call before destruction  
(If there is an exception in the destructor, it's better to log it than to hide it (if the logger is available •  
In simple applications, it is acceptable to rely on **std::terminate** (for cases of **noexcept** by default in C++11) to •  
.handle exceptions

.Anonymous code blocks **.6**

You can create a separate code block inside a single function in order to make certain variables local, so that the destructors are called when exiting the block

```
;()Block block = data.in->read  
  
}  
;std::lock_guard<std::mutex> lock(mutex  
;data.ready = true  
;data.block = block  
{  
  
;()ready_any.set
```

## .Multithreading .7

:In offline data processing programs

Try to get the best possible performance on a single CPU core. You can then parallelize your code if necessary

•

:In server applications

Use the thread pool to process requests. At this point, we haven't had any tasks that required userspace context switching

•

.Fork is not used for parallelization

## .Syncing threads .8

Often it is possible to make different threads use different memory cells (even better: different cache lines,) and (to not use any thread synchronization (except `joinAll`

.If synchronization is required, in most cases, it is sufficient to use mutex under `lock_guard`

.In other cases use system synchronization primitives. Do not use busy wait

.Atomic operations should be used only in the simplest cases

.Do not try to implement lock-free data structures unless it is your primary area of expertise

## .Pointers vs references .9

.In most cases, prefer references

## .const .10

.Use constant references, pointers to constants, `const_iterator`, and const methods

.Consider `const` to be default and use non-`const` only when necessary

.When passing variables by value, using `const` usually does not make sense

## .unsigned .11

.Use `unsigned` if necessary

## .Numeric types .12

.Use the types `UInt8`, `UInt16`, `UInt32`, `UInt64`, `Int8`, `Int16`, `Int32`, and `Int64`, as well as `size_t`, `ssize_t`, and `ptrdiff_t`

.Don't use these types for numbers: `signed/unsigned long`, `long long`, `short`, `signed/unsigned char`, `char`

## .Passing arguments .13

.(Pass complex values by reference (including `std::string`



.If a function captures ownership of an object created in the heap, make the argument type `shared_ptr` or `unique_ptr`

.Return values **.14**

.{In most cases, just use `return`. Do not write `[return std::move(res)]{}.strike`

.If the function allocates an object on heap and returns it, use `shared_ptr` or `unique_ptr`

In rare cases you might need to return the value via an argument. In this case, the argument should be a reference

```
<using AggregateFunctionPtr = std::shared_ptr<IAggregateFunction
;
Allows creating an aggregate function by its name */
/*
class AggregateFunctionFactory
{
public
;()AggregateFunctionFactory
;AggregateFunctionPtr get(const String & name, const DataTypes & argument_types) const
```

.namespace **.15**

.There is no need to use a separate `namespace` for application code

.Small libraries don't need this, either

.For medium to large libraries, put everything in a `namespace`

In the library's `.h` file, you can use `namespace detail` to hide implementation details not needed for the application code

.In a `.cpp` file, you can use a `static` or anonymous namespace to hide symbols

Also, a `namespace` can be used for an `enum` to prevent the corresponding names from falling into an external `(namespace` (but it's better to use an `enum class`

.Deferred initialization **.16**

.If arguments are required for initialization, then you normally shouldn't write a default constructor

If later you'll need to delay initialization, you can add a default constructor that will create an invalid object. Or, for a small number of objects, you can use `shared_ptr/unique_ptr`

```
(_Loader(DB::Connection * connection_, const std::string & query, size_t max_block_size
For deferred initialization ///
{ } )Loader
```

.Virtual functions **.17**

If the class is not intended for polymorphic use, you do not need to make functions virtual. This also applies to the destructor

.Encodings **.18**

.Use UTF-8 everywhere. Use `std::string` and `char *`. Do not use `std::wstring` and `wchar_t`

.Logging **.19**

.See the examples everywhere in the code

.Before committing, delete all meaningless and debug logging, and any other types of debug output

.Logging in cycles should be avoided, even on the Trace level

.Logs must be readable at any logging level

.Logging should only be used in application code, for the most part

.Log messages must be written in English

.The log should preferably be understandable for the system administrator

.Do not use profanity in the log

.Use UTF-8 encoding in the log. In rare cases you can use non-ASCII characters in the log

.Input-output **.20**

.(Don't use **iostreams** in internal cycles that are critical for application performance (and never use **stringstream**

.Use the **DB/IO** library instead

.Date and time **.21**

.See the **DateLUT** library

.include **.22**

.Always use **#pragma once** instead of include guards

.using **.23**

**.using namespace** is not used. You can use **using** with something specific. But make it local inside a class or function

.Do not use **trailing return type** for functions unless necessary **.24**

```
{auto f() -&gt; void;}{.strike}
```

.Declaration and initialization of variables **.25**

```
right way//
;std::string s = "Hello
;{"std::string s{"Hello

wrong way//
;{"auto s = std::string{"Hello
```

.For virtual functions, write **virtual** in the base class, but write **override** instead of **virtual** in descendent classes **.26**

## **++Unused Features of C**

.Virtual inheritance is not used **.1**

.Exception specifiers from C++03 are not used **.2**

## **Platform**

.We write code for a specific platform **.1**

.But other things being equal, cross-platform or portable code is preferred

.Language: C++17 **.2**

Compiler: **gcc**. At this time (December 2017), the code is compiled using version 7.2. (It can also be compiled **.3**  
(.using **clang 4**

.(++The standard library is used (**libstdc++** or **libc**

.OS: Linux Ubuntu, not older than Precise.**.4**

.Code is written for x86\_64 CPU architecture.**.5**

.The CPU instruction set is the minimum supported set among our servers. Currently, it is SSE 4.2

.Use **-Wall -Wextra -Werror** compilation flags **.6**

Use static linking with all libraries except those that are difficult to connect to statically (see the output of the **.7** **ldd** command

.Code is developed and debugged with release settings **.8**

## Tools

.KDevelop is a good IDE **.1**

.For debugging, use **gdb**, **valgrind** (**memcheck**), **strace**, **-fsanitize=...**, or **tcmalloc\_minimal\_debug** **.2**

.For profiling, use **Linux Perf**, **valgrind** (**callgrind**), or **strace -cf** **.3**

.Sources are in Git **.4**

.Assembly uses **CMake** **.5**

.Programs are released using **deb** packages **.6**

.Commits to master must not break the build **.7**

.Though only selected revisions are considered workable

.Make commits as often as possible, even if the code is only partially ready **.8**

.Use branches for this purpose

If your code in the **master** branch is not buildable yet, exclude it from the build before the **push**. You'll need to finish it or remove it within a few days

.For non-trivial changes, use branches and publish them on the server **.9**

.Unused code is removed from the repository **.10**

## Libraries

The C++14 standard library is used (experimental extensions are allowed), as well as **boost** and **Poco** **.1**  
.frameworks

.If necessary, you can use any well-known libraries available in the OS package **.2**

.If there is a good solution already available, then use it, even if it means you have to install another library

(.But be prepared to remove bad libraries from code)

You can install a library that isn't in the packages, if the packages don't have what you need or have an **.3**  
.outdated version or the wrong type of compilation

.If the library is small and doesn't have its own complex build system, put the source files in the **contrib** folder **.4**

.Preference is always given to libraries that are already in use **.5**

## General Recommendations

.Write as little code as possible **.1**

.Try the simplest solution **.2**

.Don't write code until you know how it's going to work and how the inner loop will function **.3**

.In the simplest cases, use `using` instead of classes or structs **.4**

If possible, do not write copy constructors, assignment operators, destructors (other than a virtual one, if the **.5** class contains at least one virtual function), move constructors or move assignment operators. In other words, the `.compiler-generated` functions must work correctly. You can use `default`

.Code simplification is encouraged. Reduce the size of your code where possible **.6**

## Additional Recommendations

Explicitly specifying `std::` for types from `stddef.h` **.1**

.is not recommended. In other words, we recommend writing `size_t` instead `std::size_t`, because it's shorter

.:It is acceptable to add `std`

Explicitly specifying `std::` for functions from the standard C library **.2**

.is not recommended. In other words, write `memcpy` instead of `std::memcpy`

The reason is that there are similar non-standard functions, such as `memmem`. We do use these functions on `.occasion`. These functions do not exist in `namespace std`

.If you write `std::memcpy` instead of `memcpy` everywhere, then `memmem` without `std::` will look strange

.Nevertheless, you can still use `std::` if you prefer it

.Using functions from C when the same ones are available in the standard C++ library **.3**

.This is acceptable if it is more efficient

.For example, use `memcpy` instead of `std::copy` for copying large chunks of memory

.Multiline function arguments **.4**

:Any of the following wrapping styles are allowed

```
}function  
,T1 x1  
(T2 x2
```

```
}function  
,size_t left, size_t right  
,const & RangesInDataParts ranges  
(size_t limit
```

```
,function(size_t left, size_t right  
,const & RangesInDataParts ranges  
(size_t limit
```

```
,function(size_t left, size_t right  
,const & RangesInDataParts ranges  
(size_t limit
```

```
}function  
,size_t left  
,size_t right  
,const & RangesInDataParts ranges  
(size_t limit
```

## ClickHouse Testing Functional Tests

Functional tests are the most simple and convenient to use. Most of ClickHouse features can be tested with functional tests and they are mandatory to use for every change in ClickHouse code that can be tested that way

Each functional test sends one or multiple queries to the running ClickHouse server and compares the result with reference

Tests are located in `dbms/tests/queries` directory. There are two subdirectories: `stateless` and `stateful`. Stateless tests run queries without any preloaded test data - they often create small synthetic datasets on the fly, within the test itself. Stateful tests require preloaded test data from Yandex.Metrica and not available to general public. We tend to use only `stateless` tests and avoid adding new `stateful` tests

Each test can be one of two types: `.sql` and `.sh`. `.sql` test is the simple SQL script that is piped to `clickhouse-client --multiquery --testmode`. `.sh` test is a script that is run by itself

To run all tests, use `dbms/tests/clickhouse-test` tool. Look `--help` for the list of possible options. You can simply run all tests or run subset of tests filtered by substring in test name: `./clickhouse-test substring`

The most simple way to invoke functional tests is to copy `clickhouse-client` to `/usr/bin/`, run `clickhouse-server` and then run `./clickhouse-test` from its own directory

To add new test, create a `.sql` or `.sh` file in `dbms/tests/queries/0_stateless` directory, check it manually and then generate `.reference` file in the following way: `clickhouse-client -n --testmode < 00000_test.sql > 00000_test.reference` or `./00000_test.sh > ./00000_test.reference`

Tests should use (create, drop, etc) only tables in `test` database that is assumed to be created beforehand; also tests can use temporary tables

If you want to use distributed queries in functional tests, you can leverage `remote` table function with `127.0.0.{1..2}` addresses for the server to query itself; or you can use predefined test clusters in server configuration file like `test_shard_localhost`

Some tests are marked with `zookeeper`, `shard` or `long` in their names  
`zookeeper` is for tests that are using ZooKeeper. `shard` is for tests that requires server to listen `127.0.0.*`; `distributed` or `global` have the same meaning. `long` is for tests that run slightly longer than one second. You can disable these groups of tests using `--no-zookeeper`, `--no-shard` and `--no-long` options, respectively--

## Known bugs

If we know some bugs that can be easily reproduced by functional tests, we place prepared functional tests in `dbms/tests/queries/bugs` directory. These tests will be moved to `dbms/tests/queries/0_stateless` when bugs are fixed

## Integration Tests

Integration tests allow to test ClickHouse in clustered configuration and ClickHouse interaction with other servers like MySQL, Postgres, MongoDB. They are useful to emulate network splits, packet drops, etc. These tests are run under Docker and create multiple containers with various software

See `dbms/tests/integration/README.md` on how to run these tests

Note that integration of ClickHouse with third-party drivers is not tested. Also we currently don't have integration tests with our JDBC and ODBC drivers

## Unit Tests

Unit tests are useful when you want to test not the ClickHouse as a whole, but a single isolated library or class. You can enable or disable build of tests with `ENABLE_TESTS` CMake option. Unit tests (and other test programs) are located in `tests` subdirectories across the code. To run unit tests, type `ninja test`. Some tests use `gtest`, but some are just programs that return non-zero exit code on test failure

It's not necessarily to have unit tests if the code is already covered by functional tests (and functional tests are usually much more simple to use)

## Performance Tests

Performance tests allow to measure and compare performance of some isolated part of ClickHouse on synthetic queries. Tests are located at `dbms/tests/performance`. Each test is represented by `.xml` file with description of test case. Tests are run with `clickhouse performance-test` tool (that is embedded in `clickhouse` binary). See `--help` for invocation

Each test run one or multiple queries (possibly with combinations of parameters) in a loop with some conditions for stop (like "maximum execution speed is not changing in three seconds") and measure some metrics about query performance (like "maximum execution speed"). Some tests can contain preconditions on preloaded test dataset

If you want to improve performance of ClickHouse in some scenario, and if improvements can be observed on simple queries, it is highly recommended to write a performance test. It always makes sense to use `perf top` or other perf tools during your tests

## Test Tools And Scripts

Some programs in `tests` directory are not prepared tests, but are test tools. For example, for `Lexer` there is a tool `dbms/src/Parsers/tests/lexer` that just do tokenization of stdin and writes colorized result to stdout. You can use these kind of tools as a code examples and for exploration and manual testing

You can also place pair of files `.sh` and `.reference` along with the tool to run it on some predefined input - then script result can be compared to `.reference` file. These kind of tests are not automated

## Miscellaneous Tests

There are tests for external dictionaries located at `dbms/tests/external_dictionaries` and for machine learned models in `dbms/tests/external_models`. These tests are not updated and must be transferred to integration tests

There is separate test for quorum inserts. This test run ClickHouse cluster on separate servers and emulate various failure cases: network split, packet drop (between ClickHouse nodes, between ClickHouse and ZooKeeper, between ClickHouse server and client, etc.), `kill -9`, `kill -STOP` and `kill -CONT`, like `Jepsen`. Then the test checks that all acknowledged inserts was written and all rejected inserts was not

Quorum test was written by separate team before ClickHouse was open-sourced. This team no longer work with ClickHouse. Test was accidentally written in Java. For these reasons, quorum test must be rewritten and moved to integration tests

## Manual Testing

:When you develop a new feature, it is reasonable to also test it manually. You can do it with the following steps

Build ClickHouse. Run ClickHouse from the terminal: change directory to `dbms/src/programs/clickhouse-server` and run it with `./clickhouse-server`. It will use configuration (`config.xml`, `users.xml` and files within `config.d` and `users.d` directories) from the current directory by default. To connect to ClickHouse server, run `dbms/src/programs/clickhouse-client/clickhouse-client`

Note that all clickhouse tools (server, client, etc) are just symlinks to a single binary named `clickhouse`. You can find this binary at `dbms/src/programs/clickhouse`. All tools can also be invoked as `clickhouse tool` instead of `clickhouse-tool`

Alternatively you can install ClickHouse package: either stable release from Yandex repository or you can build package for yourself with `./release` in ClickHouse sources root. Then start the server with `sudo service clickhouse-server start` (or stop to stop the server). Look for logs at `/etc/clickhouse-server/clickhouse-server.log`

When ClickHouse is already installed on your system, you can build a new `clickhouse` binary and replace the existing binary

```
sudo service clickhouse-server stop
/sudo cp ./clickhouse /usr/bin
sudo service clickhouse-server start
```

Also you can stop system clickhouse-server and run your own with the same configuration but with logging to :terminal

```
sudo service clickhouse-server stop
sudo -u clickhouse /usr/bin/clickhouse server --config-file /etc/clickhouse-server/config.xml
```

:Example with gdb

```
sudo -u clickhouse gdb --args /usr/bin/clickhouse server --config-file /etc/clickhouse-server/config.xml
```

If the system clickhouse-server is already running and you don't want to stop it, you can change port numbers in .your `config.xml` (or override them in a file in `config.d` directory), provide appropriate data path, and run it

`clickhouse` binary has almost no dependencies and works across wide range of Linux distributions. To quick and dirty test your changes on a server, you can simply `scp` your fresh built `clickhouse` binary to your server and then .run it as in examples above

## Testing Environment

Before publishing release as stable we deploy it on testing environment. Testing environment is a cluster that process 1/39 part of `Yandex.Metrica` data. We share our testing environment with Yandex.Metrica team. ClickHouse is upgraded without downtime on top of existing data. We look at first that data is processed successfully without lagging from realtime, the replication continue to work and there is no issues visible to :Yandex.Metrica team. First check can be done in the following way

```
SELECT hostName() AS h, any(version()), any(uptime()), max(UTCEventTime), count() FROM remote('example01-01-{1..3}'t',
;merge, hits) WHERE EventDate >= today() - 2 GROUP BY h ORDER BY h
```

In some cases we also deploy to testing environment of our friend teams in Yandex: Market, Cloud, etc. Also we .have some hardware servers that are used for development purposes

## Load Testing

After deploying to testing environment we run load testing with queries from production cluster. This is done .manually

.Make sure you have enabled `query_log` on your production cluster

:Collect query log for a day or more

```
clickhouse-client --query="SELECT DISTINCT query FROM system.query_log WHERE event_date = today() AND query LIKE '%ym:%'
AND query NOT LIKE '%system.query_log%' AND type = 2 AND is_initial_query" > queries.tsv
```

This is a way complicated example. `type = 2` will filter queries that are executed successfully. `query LIKE '%ym:%'` is to select relevant queries from Yandex.Metrica. `is_initial_query` is to select only queries that are initiated by client, .(not by ClickHouse itself (as parts of distributed query processing

:`scp` this log to your testing cluster and run it as following

```
clickhouse benchmark --concurrency 16 < queries.tsv
```

(probably you also want to specify a `--user`)

.Then leave it for a night or weekend and go take a rest

You should check that `clickhouse-server` doesn't crash, memory footprint is bounded and performance not degrading .over time

Precise query execution timings are not recorded and not compared due to high variability of queries and .environment

# Build Tests

Build tests allow to check that build is not broken on various alternative configurations and on some foreign systems. Tests are located at `ci` directory. They run build from source inside Docker, Vagrant, and sometimes with `.qemu-user-static` inside Docker. These tests are under development and test runs are not automated

:Motivation

Normally we release and run all tests on a single variant of ClickHouse build. But there are alternative build variants that are not thoroughly tested. Examples

- ;build on FreeBSD
- ;build on Debian with libraries from system packages
- ;build with shared linking of libraries
- ;build on AArch64 platform
- .build on PowerPc platform

For example, build with system packages is bad practice, because we cannot guarantee what exact version of packages a system will have. But this is really needed by Debian maintainers. For this reason we at least have to support this variant of build. Another example: shared linking is a common source of trouble, but it is needed for some enthusiasts

Though we cannot run all tests on all variant of builds, we want to check at least that various build variants are not broken. For this purpose we use build tests

## Testing For Protocol Compatibility

When we extend ClickHouse network protocol, we test manually that old clickhouse-client works with new clickhouse-server and new clickhouse-client works with old clickhouse-server (simply by running binaries from corresponding packages)

## Help From The Compiler

Main ClickHouse code (that is located in `dbms` directory) is built with `-Wall -Wextra -Werror` and with some additional enabled warnings. Although these options are not enabled for third-party libraries

Clang has even more useful warnings - you can look for them with `-Weverything` and pick something to default build

For production builds, gcc is used (it still generates slightly more efficient code than clang). For development, clang is usually more convenient to use. You can build on your own machine with debug mode (to save battery of your laptop), but please note that compiler is able to generate more warnings with `-O3` due to better control flow and inter-procedure analysis. When building with clang, `libc++` is used instead of `libstdc++` and when building with debug mode, debug version of `libc++` is used that allows to catch more errors at runtime

## Sanitizers

### .Address sanitizer

We run functional and integration tests under ASan on per-commit basis

### .(Valgrind (Memcheck

We run functional tests under Valgrind overnight. It takes multiple hours. Currently there is one known false positive in `re2` library, see [this article](#)

### .Undefined behaviour sanitizer

We run functional and integration tests under ASan on per-commit basis

### .Thread sanitizer

We run functional tests under TSan on per-commit basis. We still don't run integration tests under TSan on per-commit basis



## **.Memory sanitizer**

.Currently we still don't use MSan

## **.Debug allocator**

.Debug version of [jemalloc](#) is used for debug build

# Fuzzing

We use simple fuzz test to generate random SQL queries and to check that the server doesn't die. Fuzz testing is performed with Address sanitizer. You can find it in [00746\\_sql\\_fuzzy.pl](#). This test should be run continuously

.(overnight and longer

.As of December 2018, we still don't use isolated fuzz testing of library code

# Security Audit

People from Yandex Cloud department do some basic overview of ClickHouse capabilities from the security

.standpoint

# Static Analyzers

We run [PVS-Studio](#) on per-commit basis. We have evaluated [clang-tidy](#), [Coverity](#), [cppcheck](#), [PVS-Studio](#), [tscancode](#). You will find instructions for usage in [dbms/tests/instructions/](#) directory. Also you can read [the article in russian](#)

.If you use [CLion](#) as an IDE, you can leverage some [clang-tidy](#) checks out of the box

# Hardening

.[FORTIFY\\_SOURCE](#) is used by default. It is almost useless, but still makes sense in rare cases and we don't disable it

# Code Style

.Code style rules are described [here](#)

.To check for some common style violations, you can use [utils/check-style](#) script

To force proper style of your code, you can use [clang-format](#). File [.clang-format](#) is located at the sources root. It mostly corresponding with our actual code style. But it's not recommended to apply [clang-format](#) to existing files because it makes formatting worse. You can use [clang-format-diff](#) tool that you can find in clang source repository

Alternatively you can try [uncrustify](#) tool to reformat your code. Configuration is in [uncrustify.cfg](#) in the sources root. It is less tested than [clang-format](#)

.[CLion](#) has its own code formatter that has to be tuned for our code style

# Metrika B2B Tests

Each ClickHouse release is tested with Yandex Metrika and AppMetrika engines. Testing and stable versions of ClickHouse are deployed on VMs and run with a small copy of Metrika engine that is processing fixed sample of input data. Then results of two instances of Metrika engine are compared together

These tests are automated by separate team. Due to high number of moving parts, tests are fail most of the time by completely unrelated reasons, that are very difficult to figure out. Most likely these tests have negative value for us. Nevertheless these tests was proved to be useful in about one or two times out of hundreds

# Test Coverage

.As of July 2018 we don't track test coverage

# Test Automation

."We run tests with Yandex internal CI and job automation system named "Sandbox

Build jobs and tests are run in Sandbox on per commit basis. Resulting packages and test results are published in GitHub and can be downloaded by direct links. Artifacts are stored eternally. When you send a pull request on GitHub, we tag it as "can be tested" and our CI system will build ClickHouse packages (release, debug, with .address sanitizer, etc) for you

.We don't use Travis CI due to the limit on time and computational power

.We don't use Jenkins. It was used before and now we are happy we are not using Jenkins

## Third-Party Libraries Used

License	Library
BSD 2-Clause License	base64
Boost Software License 1.0	boost
MIT	brtli
MIT	capnproto
Apache License 2.0	cctz
BSD 3-Clause License	double-conversion
MIT	FastMemcpy
BSD 3-Clause License	googletest
BSD 3-Clause License	hyperscan
BSD 2-Clause License	libbtrie
BSD + MIT	libcxxabi
Zlib License	libdivide
LGPL v2.1	libgsasl
Apache License 2.0	libhdfs3
Apache License 2.0	libmetrohash
Apache License 2.0	libpcg-random
OpenSSL License	libressl
BSD 2-Clause License	librdkafka
CC0 1.0 Universal	libwidechar_width
BSD 3-Clause License	llvm
BSD 2-Clause License	lz4
LGPL v2.1	mariadb-connector-c
Public Domain	murmurhash
Zlib License	pdqsort
Boost Software License - Version 1.0	poco
BSD 3-Clause License	protobuf
BSD 3-Clause License	re2
LGPL v2.1	UnixODBC
Zlib License	zlib-ng
BSD 3-Clause License	zstd
	Roadmap ##

## Q2 2019

DDL for dictionaries

Integration with S3-like object stores

Multiple storages for hot/cold data, JBOD support

- 
- 
- 

## Q3 2019

:JOIN execution improvements

Distributed join not limited by memory

Resource pools for more precise distribution of cluster capacity between users

Fine-grained authorization

- 
- 
- 
-

Integration with external authentication services

## ClickHouse release 19.7.3.9, 2019-05-30

### Новые возможности

Добавлена возможность ограничить значения конфигурационных параметров которые может задать пользователь. Эти ограничения устанавливаются в профиле настроек пользователя. [#4931](#)

([Vitaly Baranov](#))

Добавлен вариант функции `groupUniqArray` с дополнительным параметром `max_size`, который ограничивает размер результирующего массива, аналогично

функции `groupArray(max_size)(x`

`Guillaume)` [#5026](#)

([Tassery](#))

Для входных файлов формата TSVWithNames и CSVWithNames появилась возможность определить порядок колонок в файле исходя из его заголовка. Это поведение

управляется конфигурационным параметром `input_format_with_names_use_header`

[#5081](#)

([Alexander](#))

### Исправления ошибок

Падение в процессе слияния при использовании `uncompressed_cache` и `JOIN`

[Danila\)](#) [#5133](#) [\(#5197\)](#)

([Kutenin](#))

Segmentation fault на запросе к системным таблицам [\(#5066](#)

[#5127](#)

([Ivan](#))

Потеря загружаемых данных при больших потоках загрузки через KafkaEngine

[#5080](#) [\(#4736\)](#)

([Ivan](#))

Исправлен очень редкий data race condition который мог произойти при выполнении запроса с `UNION`

`ALL` включающего минимум два `SELECT` из таблиц `system.columns`, `system.tables`, `system.parts`,

`system.parts_tables` или таблиц семейства `Merge` и одновременно выполняющихся запросов `ALTER`

(столбцов соответствующих таблиц. [#5189](#) ([alexey-milovidov](#))

### Улучшения производительности

Используется поразрядная сортировка числовых колонок для `ORDER BY` без

`LIMIT`. [#5106](#)

[Evgenii\)](#) [#4439](#)

[,Pravda](#)

[alexey-milovidov](#)

### Документация

Документация для некоторых табличных движков переведена на китайский

[,#5107](#)

[,#5094](#)

[#5087](#)

[,张风嘯\)](#)

`never)` [#5068](#)

([lee](#))

### Улучшения сборки, тестирования и пакетирования

Правильно отображаются символы в кодировке UTF-8 в `clickhouse-test`

[#5084](#)

([alexey-milovidov](#))

Добавлен параметр командной строки для [clickhouse-client](#), позволяющий  
. всегда загружать данные подсказок

[#5102](#)

([alexey-milovidov](#))

. Исправлены некоторые предупреждения PVS-Studio

[#5082](#)

([alexey-milovidov](#))

. Обновлена библиотека LZ4

[Danila](#)) [#5040](#)

([Kutenin](#))

. В зависимости сборки добавлен gperf для поддержки готовящегося PR [#5030](#)

[#5110](#)

([proller](#))

## ClickHouse release 19.6.2.11, 2019-05-13

### Новые возможности

TTL выражения, позволяющие настроить время жизни и автоматическую очистку данных в таблице (или в отдельных её столбцах. [#4212](#) ([Anton Popov](#))

Добавлена поддержка алгоритма сжатия [brotli](#) в HTTP ответах ([Accept-Encoding: br](#)). Для тела POST (запросов, эта возможность уже существовала. [#4388](#) ([Mikhail](#))

Добавлена функция [isValidUTF8](#) для проверки, содержит ли строка валидные данные в кодировке UTF-8. [#4934](#) ([Danila Kutenin](#))

Добавлены новое правило балансировки ([load\\_balancing](#)) [first\\_or\\_random](#) по которому запросы посылаются на первый заданный хост и если он недоступен - на случайные хосты шарда. Полезно для топологий с (кросс-репликацией. [#5012](#) ([nvartolomei](#))

### Экспериментальные возможности

Добавлена настройка [index\\_granularity\\_bytes](#) (адаптивная гранулярность индекса) для таблиц семейства (MergeTree\* . [#4826](#) ([alesapin](#))

### Улучшения

Добавлена поддержка для не константных и отрицательных значений аргументов смещения и длины (для функции [substringUTF8](#). [#4989](#) ([alexey-milovidov](#))

Отключение push-down в правую таблицы в left join, левую таблицу в right join, и в обе таблицы в full (join. Это исправляет неправильные JOIN результаты в некоторых случаях. [#4846](#) ([Ivan](#))

[clickhouse-copier](#): Автоматическая загрузка конфигурации задачи в zookeeper из --task-file опции [#4876](#) (([proller](#))

Добавлены подсказки с учётом опечаток для имён движков таблиц и табличных функций. [#4891](#) (([Danila Kutenin](#))

Поддержка выражений [select \\*](#) и [select tablename.\\*](#) для множественных join без подзапросов [#4898](#) (([Artem Zuikov](#))

Сообщения об ошибках об отсутствующих столбцах стали более понятными. [#4915](#) ([Artem Zuikov](#))

### Улучшение производительности

(Существенное ускорение ASOF JOIN [#4924](#) ([Martijn Bakker](#))

### Обратно несовместимые изменения

(HTTP заголовок [Query-Id](#) переименован в [X-ClickHouse-Query-Id](#) для соответствия. [#4972](#) ([Mikhail](#))

### Исправления ошибок

(Исправлены возможные разыменования нулевого указателя в [clickhouse-copier](#). [#4900](#) ([proller](#))

(Исправлены ошибки в запросах с JOIN + ARRAY JOIN [#4938](#) ([Artem Zuikov](#))

Исправлено зависание на старте сервера если внешний словарь зависит от другого словаря через (использование таблицы из БД с движком [Dictionary](#). [#4962](#) ([Vitaly Baranov](#))

При использовании `distributed_product_mode = 'local'` корректно работает использование столбцов локальных таблиц в `where/having/order by/...` через табличные алиасы. Выкидывает исключение если (таблица не имеет алиас. Доступ к столбцам без алиасов пока не возможен. [#4986](#) (Artem Zuikov) (Исправлен потенциально некорректный результат для `SELECT DISTINCT` с `JOIN` [#5001](#) (Artem Zuikov) Исправлен очень редкий data race condition который мог произойти при выполнении запроса с `UNION ALL` включающего минимум два `SELECT` из таблиц `system.columns`, `system.tables`, `system.parts`, `system.parts_tables` или таблиц семейства Merge и одновременно выполняющихся запросов `ALTER` (столбцов соответствующих таблиц. [#5189](#) (alexey-milovidov)

## Улучшения сборки/тестирования/пакетирования

Исправлена неработоспособность тестов, если `clickhouse-server` запущен на удалённом хосте [#4713](#) ((Vasily Nemkov) `clickhouse-test`: Отключена раскраска результата, если команда запускается не в терминале. [#4937](#) ((alesapin) (`clickhouse-test`: Возможность использования не только базы данных `test` [#5008](#) (proller) (Исправлены ошибки при запуске тестов под UBSan [#5037](#) (Vitaly Baranov) Добавлен аллокатор Yandex LFAalloc для аллоцирования MarkCache и UncompressedCache данных (разными способами для более надежного отлавливания проездов по памяти [#4995](#) (Danila Kutenin) Утилита для упрощения бэкпортирования изменений в старые релизы и составления changelogs. [#4949](#) (Ivan)

## ClickHouse release 19.5.4.22, 2019-05-13

### Исправления ошибок

(Исправлены возможные падения в `bitmap*` функциях [#5220](#) [#5228](#) (Andy Yang) Исправлен очень редкий data race condition который мог произойти при выполнении запроса с `UNION ALL` включающего минимум два `SELECT` из таблиц `system.columns`, `system.tables`, `system.parts`, `system.parts_tables` или таблиц семейства Merge и одновременно выполняющихся запросов `ALTER` (столбцов соответствующих таблиц. [#5189](#) (alexey-milovidov) Исправлена ошибка `Set for IN is not created yet in case of using single LowCardinality column in the left part of IN` Эта ошибка возникала когда LowCardinality столбец была частью primary key. [#5031](#) [#5154](#) (Nikolai Kochetov) Исправление функции `retention`: только первое соответствующее условие добавлялось в состояние данных. Сейчас все условия которые удовлетворяют в строке данных добавляются в состояние. [#5119](#) (小路)

## ClickHouse release 19.5.3.8, 2019-04-18

### Исправления ошибок

Исправлен тип настройки `max_partitions_per_insert_block` с булевого на UInt64. [#5028](#) (Mohammad Hossein Sekhavat)

## ClickHouse release 19.5.2.6, 2019-04-15

### Новые возможности

Добавлены функции для работы с несколькими регулярными выражениями с помощью библиотеки `Hyperscan`. (`multiMatchAny`, `multiMatchAnyIndex`, `multiFuzzyMatchAny`, `multiFuzzyMatchAnyIndex`). [#4780](#), [#4841](#) ((Danila Kutenin) (Добавлена функция `multiSearchFirstPosition`. [#4780](#) (Danila Kutenin) (Реализована возможность указания построчного ограничения доступа к таблицам. [#4792](#) (Ivan) Добавлен новый тип вторичного индекса на базе фильтра Блума (используется в функциях `equal`, `in` и `like`). [#4499](#) (Nikita Vasilev) Добавлен `ASOF JOIN` которые позволяет джойнить строки по наиболее близкому известному значению. [#4774](#) [#4867](#) [#4863](#) [#4875](#) (Martijn Bakker, Artem Zuikov) Теперь запрос `COMMA JOIN` переписывается `CROSS JOIN`. И затем оба переписываются в `INNER JOIN`, если (это возможно. [#4661](#) (Artem Zuikov)

## Улучшения

Функции `topK` и `topKWeighted` теперь поддерживают произвольный `loadFactor` (исправляет issue [#4252](#)). [\(#4634 \(Kirill Danshin](#)

Добавлена возможность использования настройки `parallel_replicas_count > 1` для таблиц без (семплирования (ранее настройка просто игнорировалась). [#4637 \(Alexey Elymanov](#)

Поддержан запрос `CREATE OR REPLACE VIEW`. Позволяет создать `VIEW` или изменить запрос в одном (выражении. [#4654 \(Boris Granveaud](#)

(Движок таблиц `Buffer` теперь поддерживает `PREWHERE`. [#4671 \(Yangkuan Liu](#)

Теперь реплицируемые таблицы могут стартовать в `readonly` режиме даже при отсутствии `zookeeper`. [\(#4691 \(alesapin](#)

Исправлено мигание прогресс-бара в `clickhouse-client`. Проблема была наиболее заметна при (использовании `FORMAT Null` в потоковых запросах. [#4811 \(alexey-milovidov](#)

Добавлена возможность отключения функций, использующих библиотеку `hyperscan`, для пользователей, чтобы ограничить возможное неконтролируемое потребление ресурсов. [#4816 \(\(alexey-milovidov](#)

(Добавлено логирование номера версии во все исключения. [#4824 \(proller](#)

Добавлено ограничение на размер строк и количество параметров в функции `multiMatch`. Теперь они (принимают строки уместающиеся в `unsigned int`. [#4834 \(Danila Kutenin](#)

(Улучшено использование памяти и обработка ошибок в `Hyperscan`. [#4866 \(Danila Kutenin](#)

Теперь системная таблица `system.graphite_detentions` заполняется из конфигурационного файла для (таблиц семейства `*GraphiteMergeTree`. [#4584 \(Mikhail f. Shiryayev](#)

Функция `trigramDistance` переименована в функцию `ngramDistance`. Добавлено несколько функций с (`CaseInsensitive` и `UTF`. [#4602 \(Danila Kutenin](#)

(Улучшено вычисление вторичных индексов. [#4640 \(Nikita Vasilev](#)

Теперь обычные колонки, а также колонки `DEFAULT`, `MATERIALIZED` и `ALIAS` хранятся в одном списке ((исправляет issue [#2867](#)). [#4707 \(Alex Zatelepin](#)

## Исправления ошибок

В случае невозможности выделить память вместо вызова `std::terminate` бросается исключение (`std::bad_alloc`. [#4665 \(alexey-milovidov](#)

Исправлены ошибки чтения `capnproto` из буфера. Иногда файлы не загружались по HTTP. [#4674 \(\(Vladislav](#)

(Исправлена ошибка `Unknown log entry type: 0` после запроса `OPTIMIZE TABLE FINAL`. [#4683 \(Amos Bird](#)

При передаче неправильных аргументов в `hasAny` и `hasAll` могла происходить ошибка сегментирования. [\(#4698 \(alexey-milovidov](#)

Исправлен дедлок, который мог происходить при запросе `DROP DATABASE dictionary`. [#4701 \(alexey-milovidov](#)

(Исправлено неопределенное поведение в функциях `median` и `quantile`. [#4702 \(hcz](#)

Исправлено определение уровня сжатия при указании настройки `network_compression_method` в нижнем (регистре. Было сломано в `v19.1`. [#4706 \(proller](#)

(Настройка `<timezone>UTC</timezone>` больше не игнорируется (исправляет issue [#4658](#)). [#4718 \(proller](#)

(Исправлено поведение функции `histogram` с `Distributed` таблицами. [#4741 \(olegkv](#)

Исправлено срабатывание `thread-санитайзера` с ошибкой `destroy of a locked mutex`. [#4742 \(alexey-milovidov](#)

Исправлено срабатывание `thread-санитайзера` при завершении сервера, вызванное гонкой при использовании системных логов. Также исправлена потенциальная ошибка `use-after-free` при (завершении сервера в котором был включен `part_log`. [#4758 \(alexey-milovidov](#)

Исправлена перепроверка кусков в `ReplicatedMergeTreeAlterThread` при появлении ошибок. [#4772 \(Nikolai \(Kochetov](#)

Исправлена работа арифметических операций с промежуточными состояниями агрегатных функций (для константных аргументов (таких как результаты подзапросов). [#4776 \(alexey-milovidov](#)

Теперь имена колонок всегда экранируются в файлах с метаданными. В противном случае было (невозможно создать таблицу с колонкой с именем `index`. [#4782 \(alexey-milovidov](#)

(Исправлено падение в запросе `ALTER ... MODIFY ORDER BY` к `Distributed` таблице. [#4790 \(TCeason](#)



- Исправлена ошибка сегментирования при запросах с `JOIN ON` и включенной настройкой (`enable_optimize_predicate_expression`. #4794 (Winter Zhang
- Исправлено добавление лишней строки после чтения protobuf-сообщения из таблицы с движком `Kafka`. (#4808 (Vitaly Baranov
- Исправлено падение при запросе с `JOIN ON` с не `nullable` и `nullable` колонкой. Также исправлено (поведение при появлении `NULLs` среди ключей справа в `ANY JOIN` + `join_use_nulls`. #4815 (Artem Zuikov
- Исправлена ошибка сегментирования в `clickhouse-copier`. #4835 (proller
- Исправлена гонка при `SELECT` запросе из `system.tables` если таблица была конкурентно переименована (или к ней был применен `ALTER` запрос. #4836 (alexey-milovidov
- Исправлена гонка при скачивании куска, который уже является устаревшим. #4839 (alexey-milovidov
- Исправлена редкая гонка при `RENAME` запросах к таблицам семейства MergeTree. #4844 (alexey-milovidov
- Исправлена ошибка сегментирования в функции `arrayIntersect`. Ошибка возникала при вызове функции (с константными и не константными аргументами. #4847 (Lixiang Qian
- Исправлена редкая ошибка при чтении из колонки типа `Array(LowCardinality)`, которая возникала, если в (колонке содержалось большее количество подряд идущих пустых массивов. #4850 (Nikolai Kochetov
- Исправлено падение в запроса с `FULL/RIGHT JOIN` когда объединение происходило по `nullable` и не (`nullable` колонке. #4855 (Artem Zuikov
- Исправлена ошибка `No message received`, возникавшая при скачивании кусков между репликами. #4856 ((alesapin
- Исправлена ошибка в функции `arrayIntersect` приводившая к неправильным результатам в случае (нескольких повторяющихся значений в массиве. #4871 (Nikolai Kochetov
- Исправлена гонка при конкурентных `ALTER COLUMN` запросах, которая могла приводить к падению (сервера (исправляет issue #3421). #4592 (Alex Zatelepin
- Исправлен некорректный результат в `FULL/RIGHT JOIN` запросах с константной колонкой. #4723 (Artem (Zuikov
- Исправлено появление дубликатов в `GLOBAL JOIN` со звездочкой. #4705 (Artem Zuikov
- Исправлено определение параметров кодеков в запросах `ALTER MODIFY`, если тип колонки не был (указан. #4883 (alesapin
- Функции `cutQueryStringAndFragment()` и `queryStringAndFragment()` теперь работают корректно, когда `URL` (содержит фрагмент, но не содержит запроса. #4894 (Vitaly Baranov
- Исправлена редкая ошибка, возникавшая при установке настройки `min_bytes_to_use_direct_io` больше нуля. Она возникла при необходимости сдвинуться в файле, который уже прочитан до конца. #4897 ((alesapin
- Исправлено неправильное определение типов аргументов для агрегатных функций с `LowCardinality` (аргументами (исправляет #4919). #4922 (Nikolai Kochetov
- Исправлена неверная квалификация имён в `GLOBAL JOIN`. #4969 (Artem Zuikov
- Исправлен результат функции `toISOWeek` для 1970 года. #4988 (alexey-milovidov
- Исправлено дублирование `DROP`, `TRUNCATE` и `OPTIMIZE` запросов, когда они выполнялись `ON CLUSTER` для (семейства таблиц `ReplicatedMergeTree*`. #4991 (alesapin

## Обратно несовместимые изменения

- Настройка `insert_sample_with_metadata` переименована в `input_format_defaults_for_omitted_fields`. #4771 (Artem (Zuikov
- Добавлена настройка `max_partitions_per_insert_block` (со значением по умолчанию 100). Если вставляемый блок содержит большое количество партиций, то бросается исключение. Лимит можно убрать (выставив настройку в 0 (не рекомендуется). #4845 (alexey-milovidov
- Функции мультипоиска были переименованы (`multiPosition` в `multiSearchAllPositions`, `multiSearch` в (`multiSearchAny`, `firstMatch` в `multiSearchFirstIndex`). #4780 (Danila Kutenin

## Улучшение производительности

- Оптимизирован поиска с помощью алгоритма Volnitsky с помощью инлайнинга. Это дает около 5-10% улучшения производительности поиска для запросов ищущих множество слов или много одинаковых (биграмм. #4862 (Danila Kutenin

Исправлено снижение производительности при выставлении настройки `use_uncompressed_cache` больше (нуля для запросов, данные которых целиком лежат в кеше. [#4913](#) (alesapin)

## Улучшения сборки/тестирования/пакетирования

Более строгие настройки для debug-сборок: более гранулярные маппинги памяти и использование ASLR; добавлена защита памяти для кеша засечек и индекса. Это позволяет найти больше ошибок порчи памяти, которые не обнаруживают address-санитайзер и thread-санитайзер. [#4632](#) (alexey-milovidov)

Добавлены настройки `ENABLE_PROTOBUF`, `ENABLE_PARQUET` и `ENABLE_BROTLI` которые позволяют отключить (соответствующие компоненты. [#4669](#) (Silviu Caragea)

Теперь при зависании запросов во время работы тестов будет показан список запросов и стек-трейсы (всех потоков. [#4675](#) (alesapin)

(Добавлены ретраи при ошибке `Connection loss` в `clickhouse-test`. [#4682](#) (alesapin)

(Добавлена возможность сборки под FreeBSD в `packager`-скрипт. [#4712](#) [#4748](#) (alesapin)

(Теперь при установке предлагается установить пароль для пользователя 'default'. [#4725](#) (proller)

(Убраны предупреждения из библиотеки `rdkafka` при сборке. [#4740](#) (alexey-milovidov)

(Добавлена возможность сборки без поддержки ssl. [#4750](#) (proller)

Добавлена возможность запускать докер-образ с `clickhouse-server` из под любого пользователя. [#4753](#) ((Mikhail f. Shiryayev)

(Boost обновлен до 1.69. [#4793](#) (proller)

Отключено использование `mremap` при сборке с thread-санитайзером, что приводило к ложным (срабатываниям. Исправлены ошибки thread-санитайзера в `stateful`-тестах. [#4859](#) (alexey-milovidov)

Добавлен тест проверяющий использование схемы форматов для HTTP-интерфейса. [#4864](#) (Vitaly Baranov)

## ClickHouse release 19.4.4.33, 2019-04-17

### Исправление ошибок

В случае невозможности выделить память вместо вызова `std::terminate` бросается исключение (`std::bad_alloc`. [#4665](#) (alexey-milovidov)

Исправлены ошибки чтения `carpproto` из буфера. Иногда файлы не загружались по HTTP. [#4674](#) ((Vladislav)

(Исправлена ошибка `Unknown log entry type: 0` после запроса `OPTIMIZE TABLE FINAL`. [#4683](#) (Amos Bird)

При передаче неправильных аргументов в `hasAny` и `hasAll` могла происходить ошибка сегментирования. [#4698](#) (alexey-milovidov)

Исправлен дедлок, который мог происходить при запросе `DROP DATABASE dictionary`. [#4701](#) (alexey-milovidov)

(Исправлено неопределенное поведение в функциях `median` и `quantile`. [#4702](#) (hcz)

Исправлено определение уровня сжатия при указании настройки `network_compression_method` в нижнем регистре. Было сломано в v19.1. [#4706](#) (proller)

(Настройка `<timezone>UTC</timezone>` больше не игнорируется (исправляет issue [#4658](#)). [#4718](#) (proller)

(Исправлено поведение функции `histogram` с `Distributed` таблицами. [#4741](#) (olegk)

Исправлено срабатывание thread-санитайзера с ошибкой `destroy of a locked mutex`. [#4742](#) (alexey-milovidov)

Исправлено срабатывание thread-санитайзера при завершении сервера, вызванное гонкой при использовании системных логов. Также исправлена потенциальная ошибка `use-after-free` при (завершении сервера в котором был включен `part_log`. [#4758](#) (alexey-milovidov)

Исправлена перепроверка кусков в `ReplicatedMergeTreeAlterThread` при появлении ошибок. [#4772](#) (Nikolai Kochetov)

Исправлена работа арифметических операций с промежуточными состояниями агрегатных функций (для константных аргументов (таких как результаты подзапросов). [#4776](#) (alexey-milovidov)

Теперь имена колонок всегда экранируются в файлах с метаданной. В противном случае было (невозможно создать таблицу с колонкой с именем `index`. [#4782](#) (alexey-milovidov)

(Исправлено падение в запросе `ALTER ... MODIFY ORDER BY` к `Distributed` таблице. [#4790](#) (TCeason)

Исправлена ошибка сегментирования при запросах с `JOIN ON` и включенной настройкой (`enable_optimize_predicate_expression`. [#4794](#) (Winter Zhang)



- Исправлено добавление лишней строки после чтения protobuf-сообщения из таблицы с движком `Kafka`. [#4808](#) (Vitaly Baranov)
- (Исправлена ошибка сегментирования в `clickhouse-copier`. [#4835](#) (proller)
- Исправлена гонка при `SELECT` запросе из `system.tables` если таблица была конкурентно переименована (или к ней был применен `ALTER` запрос. [#4836](#) (alexey-milovidov)
- (Исправлена гонка при скачивании куска, который уже является устаревшим. [#4839](#) (alexey-milovidov)
- Исправлена редкая гонка при `RENAME` запросах к таблицам семейства MergeTree. [#4844](#) (alexey-milovidov)
- Исправлена ошибка сегментирования в функции `arrayIntersect`. Ошибка возникала при вызове функции (с константными и не константными аргументами. [#4847](#) (Lixiang Qian)
- Исправлена редкая ошибка при чтении из колонки типа `Array(LowCardinality)`, которая возникала, если в (колонке содержалось большее количество подряд идущих пустых массивов. [#4850](#) (Nikolai Kochetov)
- Исправлена ошибка `No message received`, возникавшая при скачивании кусков между репликами. [#4856](#) ((alesapin)
- Исправлена ошибка в функции `arrayIntersect` приводившая к неправильным результатам в случае (нескольких повторяющихся значений в массиве. [#4871](#) (Nikolai Kochetov)
- Исправлена гонка при конкурентных `ALTER COLUMN` запросах, которая могла приводить к падению (сервера (исправляет issue [#3421](#)). [#4592](#) (Alex Zatelepin)
- Исправлено определение параметров кодеков в запросах `ALTER MODIFY`, если тип колонки не был (указан. [#4883](#) (alesapin)
- Функции `cutQueryStringAndFragment()` и `queryStringAndFragment()` теперь работают корректно, когда URL (содержит фрагмент, но не содержит запроса. [#4894](#) (Vitaly Baranov)
- Исправлена редкая ошибка, возникавшая при установке настройки `min_bytes_to_use_direct_io` больше нуля. Она возникла при необходимости сдвинуться в файле, который уже прочитан до конца. [#4897](#) ((alesapin)
- Исправлено неправильное определение типов аргументов для агрегатных функций с `LowCardinality` (аргументами (исправляет [#4919](#)). [#4922](#) (Nikolai Kochetov)
- (Исправлен результат функции `toISOWeek` для 1970 года. [#4988](#) (alexey-milovidov)
- Исправлено дублирование `DROP`, `TRUNCATE` и `OPTIMIZE` запросов, когда они выполнялись `ON CLUSTER` для (семейства таблиц `ReplicatedMergeTree*`. [#4991](#) (alesapin)

## Улучшения

- Теперь обычные колонки, а также колонки `DEFAULT`, `MATERIALIZED` и `ALIAS` хранятся в одном списке ((исправляет issue [#2867](#)). [#4707](#) (Alex Zatelepin)

## ClickHouse release 19.4.3.11, 2019-04-02

### Исправление ошибок

- Исправлено падение в запроса с `FULL/RIGHT JOIN` когда объединение происходило по nullable и не (nullable колонке. [#4855](#) (Artem Zuikov)
- (Исправлена ошибка сегментирования в `clickhouse-copier`. [#4835](#) (proller)

### Улучшения сборки/тестирования/пакетирования

- Добавлена возможность запускать докер-образ с `clickhouse-server` из под любого пользователя. [#4753](#) ((Mikhail f. Shiryayev)

## ClickHouse release 19.4.2.7, 2019-03-30

### Исправление ошибок

- Исправлена редкая ошибка при чтении из колонки типа `Array(LowCardinality)`, которая возникала, если в (колонке содержалось большее количество подряд идущих пустых массивов. [#4850](#) (Nikolai Kochetov)

## ClickHouse release 19.4.1.3, 2019-03-19

### Исправление ошибок

Исправлено поведение удаленных запросов, которые одновременно содержали `LIMIT BY` и `LIMIT`. Раньше для таких запросов `LIMIT` мог быть выполнен до `LIMIT BY`, что приводило к перефильтрации. [#4708](#) (([Constantin S. Pan](#)

## ClickHouse release 19.4.0.49, 2019-03-09

### Новые возможности

Добавлена полная поддержка формата `Protobuf` (чтение и запись, вложенные структуры данных). [#4174](#) [#4493](#) ([Vitaly Baranov](#))

Добавлены функции для работы с битовыми масками с использованием библиотеки `Roaring Bitmaps`. [#4207](#) ([Andy Yang](#)) [#4568](#) ([Vitaly Baranov](#))

(Поддержка формата `Parquet` [#4448](#) ([proller](#))

Вычисление расстояния между строками с помощью подсчёта N-грам - для приближённого сравнения (строк. Алгоритм похож на `q-gram metrics` в языке R. [#4466](#) ([Danila Kutenin](#))

Движок таблиц `GraphiteMergeTree` поддерживает отдельные шаблоны для правил агрегации и для (правил времени хранения. [#4426](#) ([Mikhail f. Shiryayev](#))

Добавлены настройки `max_execution_speed` и `max_execution_speed_bytes` для того, чтобы ограничить потребление ресурсов запросами. Добавлена настройка `min_execution_speed_bytes` в дополнение к (`min_execution_speed`. [#4430](#) ([Winter Zhang](#))

Добавлена функция `flatten` - конвертация многомерных массивов в плоский массив. [#4555](#) [#4409](#) (([alexey-milovidov](#), [kzon](#))

Добавлены функции `arrayEnumerateDenseRanked` и `arrayEnumerateUniqRanked` (похожа на `arrayEnumerateUniq` но позволяет указать глубину, на которую следует смотреть в многомерные массивы). [#4475](#) ([proller](#)) [#4601](#) ([alexey-milovidov](#))

Добавлена поддержка множества JOIN в одном запросе без подзапросов, с некоторыми ограничениями: без звёздочки и без алиасов сложных выражений в ON/WHERE/GROUP BY/... [#4462](#) (([Artem Zuikov](#)

### Исправления ошибок

.Этот релиз также содержит все исправления из 19.3 и 19.1

Исправлена ошибка во вторичных индексах (экспериментальная возможность): порядок гранул при (INSERT был неверным. [#4407](#) ([Nikita Vasilev](#))

Исправлена работа вторичного индекса (экспериментальная возможность) типа `set` для столбцов типа `Nullable` и `LowCardinality`. Ранее их использование вызывало ошибку `Data type must be deserialized with multiple streams` при запросе SELECT. [#4594](#) ([Nikolai Kochetov](#))

Правильное запоминание времени последнего обновления при полной перезагрузке словарей типа (`executable`. [#4551](#) ([Tema Novikov](#))

(Исправлена неработоспособность прогресс-бара, возникшая в версии 19.3 [#4627](#) ([filimonov](#))

Исправлены неправильные значения `MemoryTracker`, если кусок памяти был уменьшен в размере, в (очень редких случаях. [#4619](#) ([alexey-milovidov](#))

(Исправлено `undefined behaviour` в `ThreadPool` [#4612](#) ([alexey-milovidov](#))

Исправлено очень редкое падение с сообщением `mutex lock failed: Invalid argument`, которое могло (произойти, если таблица типа `MergeTree` удалялась одновременно с SELECT. [#4608](#) ([Alex Zatelepin](#))

(Совместимость ODBC драйвера с типом данных `LowCardinality` [#4381](#) ([proller](#))

(Исправление ошибки `AIOcontextPool: Found io_event with unknown id 0` под ОС FreeBSD [#4438](#) ([urgordeadbeef](#))

Таблица `system.part_log` создавалась независимо от того, была ли она объявлена в конфигурации. [#4483](#) ([alexey-milovidov](#))

(Исправлено `undefined behaviour` в функции `dictIsIn` для словарей типа `cache`. [#4515](#) ([alesapin](#))

Исправлен deadlock в случае, если запрос SELECT блокирует одну и ту же таблицу несколько раз (например - из разных потоков, либо при выполнении разных подзапросов) и одновременно с этим (производится DDL запрос. [#4535](#) ([Alex Zatelepin](#))

Настройка `compile_expressions` выключена по-умолчанию до тех пор, пока мы не зафиксируем исходники используемой библиотеки `LLVM` и не будем проверять её под `ASan` (сейчас библиотека `LLVM` берётся из (системы). [#4579](#) ([alesapin](#))

- Исправлено падение по `std::terminate`, если `invalidate_query` для внешних словарей с источником `clickhouse` вернул неправильный результат (пустой; более чем одну строку; более чем один столбец).
- Исправлена ошибка, из-за которой запрос `invalidate_query` производился каждые пять секунд, (независимо от указанного `lifetime`. #4583 (alexey-milovidov
- Исправлен deadlock в случае, если запрос `invalidate_query` для внешнего словаря с источником `clickhouse` использовал таблицу `system.dictionaries` или базу данных типа `Dictionary` (редкий случай). #4599 (alexey-milovidov
- Исправлена работа CROSS JOIN с пустым WHERE #4598 (Artem Zuikov
- Исправлен segfault в функции `replicate` с константным аргументом. #4603 (alexey-milovidov
- Исправлена работа predicate pushdown (настройка `enable_optimize_predicate_expression`) с лямбда-функциями. #4408 (Winter Zhang
- (Множественные исправления для множества JOIN в одном запросе. #4595 (Artem Zuikov

## Улучшения

- (Поддержка алиасов в секции JOIN ON для правой таблицы #4412 (Artem Zuikov
- (Используются правильные алиасы в случае множественных JOIN с подзапросами. #4474 (Artem Zuikov
- Исправлена логика работы predicate pushdown (настройка `enable_optimize_predicate_expression`) для JOIN. #4387 (Ivan

## Улучшения производительности

- (Улучшена эвристика оптимизации "перенос в PREWHERE". #4405 (alexey-milovidov
- Используются настоящие lookup таблицы вместо хэш-таблиц в случае 8 и 16 битных ключей.
- (Интерфейс хэш-таблиц обобщён, чтобы поддерживать этот случай. #4536 (Amos Bird
- (Улучшена производительность сравнения строк. #4564 (alexey-milovidov
- Очередь DDL операций (для запросов ON CLUSTER) очищается в отдельном потоке, чтобы не замедлять основную работу. #4502 (Alex Zatelepin
- Даже если настройка `min_bytes_to_use_direct_io` выставлена в 1, не каждый файл открывался в режиме O\_DIRECT, потому что размер файлов иногда недооценивался на размер одного сжатого блока. #4526 (alexey-milovidov

## Улучшения сборки/тестирования/пакетирования

- (Добавлена поддержка компилятора clang-9 #4604 (alexey-milovidov
- (Исправлены неправильные `__asm__` инструкции #4621 (Konstantin Podshumok
- Добавлена поддержка задания настроек выполнения запросов для `clickhouse-performance-test` из командной строки. #4437 (alesapin
- (Тесты словарей перенесены в интеграционные тесты. #4477 (alesapin
- В набор автоматизированных тестов производительности добавлены запросы, находящиеся в разделе ("benchmark" на официальном сайте. #4496 (alexey-milovidov
- (Исправления сборки в случае использования внешних библиотек lz4 и xxhash. #4495 (Orivej Desh
- Исправлен undefined behaviour, если функция `quantileTiming` была вызвана с отрицательным или нецелым аргументом (обнаружено с помощью fuzz test под undefined behaviour sanitizer). #4506 (alexey-milovidov
- (Исправлены опечатки в коде. #4531 (sdk2
- (Исправлена сборка под Mac. #4371 (Vitaly Baranov
- (Исправлена сборка под FreeBSD и для некоторых необычных конфигурациях сборки. #4444 (proller

## ClickHouse release 19.3.7, 2019-03-12

### Исправления ошибок

- Исправлена ошибка в #3920. Ошибка проявлялась в виде случайных повреждений кэша (сообщения `Unknown codec family code`, `Cannot seek through file`) и segfault. Ошибка впервые возникла в 19.1 и присутствует во всех версиях до 19.1.10 и 19.3.6. #4623 (alexey-milovidov

## ClickHouse release 19.3.6, 2019-03-02

### Исправления ошибок

- Если в пуле потоков было более 1000 потоков, то при выходе из потока, вызывается `std::terminate`. [Azat \(Khuzhin #4485 #4505\)](#) ([alexey-milovidov](#))
- Теперь возможно создавать таблицы `ReplicatedMergeTree*` с комментариями столбцов без указания `DEFAULT`, а также с `CODEC` но без `COMMENT` и `DEFAULT`. Исправлено сравнение `CODEC` друг с другом. [#4523](#) ([alesapin](#))
- Исправлено падение при `JOIN` по массивам и кортежам. [#4552](#) ([Artem Zuikov](#))
- Исправлено падение `clickhouse-copier` с сообщением `ThreadStatus not created`. [#4540](#) ([Artem Zuikov](#))
- Исправлено зависание сервера при завершении работы в случае использования распределённых DDL. [#4472](#) ([Alex Zatelepin](#))
- В сообщениях об ошибке при парсинге текстовых форматов, выдавались неправильные номера (столбцов, в случае, если номер больше 10. [#4484](#) ([alexey-milovidov](#))

## Улучшения сборки/тестирования/пакетирования

- Исправлена сборка с включенным `AVX`. [#4527](#) ([alexey-milovidov](#))
- Исправлена поддержка расширенных метрик выполнения запроса в случае, если ClickHouse был собран на системе с новым ядром Linux, а запускается на системе с существенно более старым ядром. [#4541](#) ([nvartolomei](#))
- Продолжение работы в случае невозможности применить настройку `core_dump.size_limit` с выводом (предупреждения. [#4473](#) ([proller](#))
- (Удалено `inline` для `void readBinary(...)` в `Field.cpp`. [#4530](#) ([hcz](#))

## ClickHouse release 19.3.5, 2019-02-21

### :Исправления ошибок

- Исправлена ошибка обработки длинных `http`-запросов на вставку на стороне сервера. [#4454](#) ([alesapin](#))
- Исправлена обратная несовместимость со старыми версиями, появившаяся из-за некорректной (реализации) настройки `send_logs_level`. [#4445](#) ([alexey-milovidov](#))
- Исправлена обратная несовместимость табличной функции `remote`, появившаяся из-за добавления (комментариев колонок. [#4446](#) ([alexey-milovidov](#))

## ClickHouse release 19.3.4, 2019-02-16

### :Улучшения

- При выполнении запроса `ATTACH TABLE` при проверке ограничений на используемую память теперь не учитывается память, занимаемая индексом таблицы. Это позволяет избежать ситуации, когда (невозможно сделать `ATTACH TABLE` после соответствующего `DETACH TABLE`. [#4396](#) ([alexey-milovidov](#))
- Немного увеличены ограничения на максимальный размер строки и массива, полученные от ZooKeeper. Это позволяет продолжать работу после увеличения настройки ZooKeeper `CLIENT_JVMFLAGS=-Djute.maxbuffer=...`. [#4398](#) ([alexey-milovidov](#))
- Теперь реплику, отключенную на длительный период, можно восстановить, даже если в её очереди (скопилось огромное число записей. [#4399](#) ([alexey-milovidov](#))
- Для вторичных индексов типа `set` добавлен обязательный параметр (максимальное число хранимых (значений). [#4386](#) ([Nikita Vasilev](#))

### :Исправления ошибок

- Исправлен неверный результат запроса с модификатором `WITH ROLLUP` при группировке по (единственному столбцу типа `LowCardinality`. [#4384](#) ([Nikolai Kochetov](#))
- Исправлена ошибка во вторичном индексе типа `set` (гранулы, в которых было больше, чем `max_rows` (строк, игнорировались). [#4386](#) ([Nikita Vasilev](#))
- Исправлена подстановка `alias`-ов в запросах с подзапросом, содержащим этот же `alias` ([#4110](#)). [#4351](#) ([Artem Zuikov](#))

### :Улучшения сборки/тестирования/пакетирования

- (Множество исправлений для сборки под FreeBSD. [#4397](#) ([proller](#))
- (Возможность запускать `clickhouse-server` для stateless тестов из `docker`-образа. [#4347](#) ([Vasily Nemkov](#))

# ClickHouse release 19.3.3, 2019-02-13

## :Новые возможности

- Добавлен запрос `KILL MUTATION`, который позволяет удалять мутации, которые по какой-то причине не могут выполняться. В таблицу `system.mutations` для облегчения диагностики добавлены столбцы (`latest_failed_part`, `latest_fail_time`, `latest_fail_reason`. #4287 (Alex Zatelepin
- (Добавлена агрегатная функция `entropy`, которая вычисляет энтропию Шеннона. #4238 (Quid37
- (Добавлена обобщённая реализация функции `arrayWithConstant`. #4322 (alexey-milovidov
- (Добавлен оператор сравнения `NOT BETWEEN`. #4228 (Dmitry Naumov
- Добавлена функция `sumMapFiltered` - вариант `sumMap`, позволяющий указать набор ключей, по которым (будет производиться суммирование. #4129 (Léo Ercolanelli
- (Добавлена функция `sumMapWithOverflow`. #4151 (Léo Ercolanelli
- (Добавлена поддержка `Nullable` типов в табличной функции `mysql`. #4198 (Emmanuel Donin de Rosière
- (Добавлена поддержка произвольных константных выражений в секции `LIMIT`. #4246 (k3box
- Добавлена агрегатная функция `topKWeighted` - вариант `topK`, позволяющий задавать (целый (неотрицательный) вес добавляемого значения. #4245 (Andrew Golman
- Движок `Join` теперь поддерживает настройку `join_any_take_last_row`, которая позволяет перезаписывать (значения для существующих ключей. #3973 (Amos Bird
- (Добавлена функция `toStartOfInterval`. #4304 (Vitaly Baranov
- (Добавлена функция `toStartOfTenMinutes`. #4298 (Vitaly Baranov
- (Добавлен формат `RowBinaryWithNamesAndTypes`. #4200 (Oleg V. Kozlyuk
- (Добавлены типы `IPv4` и `IPv6`. Более эффективная реализация функций `IPv*`. #3669 (Vasily Nemkov
- (Добавлен выходной формат `Protobuf`. #4005 #4158 (Vitaly Baranov
- В HTTP-интерфейсе добавлена поддержка алгоритма сжатия `brrotli` для вставляемых данных. #4235 ((Mikhail
- Клиент командной строки теперь подсказывает правильное имя, если пользователь опечатался в (названии функции. #4239 (Danila Kutenin
- (В HTTP-ответ сервера добавлен заголовок `Query-Id`. #4231 (Mikhail

## :Экспериментальные возможности

- Добавлена поддержка вторичных индексов типа `minmax` и `set` для таблиц семейства `MergeTree` ((позволяют быстро пропускать целые блоки данных). #4143 (Nikita Vasilev
- Добавлена поддержка преобразования `CROSS JOIN` в `INNER JOIN`, если это возможно. #4221 #4266 (Artem (Zuikov

## :Исправления ошибок

- Исправлена ошибка `Not found column` для случая дублирующихся столбцов в секции `JOIN ON`. #4279 ((Artem Zuikov
- Команда `START REPLICATED SENDS` теперь действительно включает посылку кусков данных при (репликации. #4229 (nvartolomei
- (Исправлена агрегация столбцов типа `Array(LowCardinality)`. #4055 (KochetovNicolai
- Исправлена ошибка, приводившая к тому, что при исполнении запроса `INSERT ... SELECT ... FROM file(...)` терялась первая строка файла, если он был в формате `CSVWithNames` или `TSVWithNames`. #4297 (alexey-milovidov
- Исправлено падение при перезагрузке внешнего словаря, если словарь недоступен. Ошибка возникла (в 19.1.6. #4188 (proller
- Исправлен неверный результат `ALL JOIN`, если в правой таблице присутствуют дубликаты ключа `join`. (#4184 (Artem Zuikov
- Исправлено падение сервера при включённой опции `use_uncompressed_cache`, а также исключение о (неправильном размере разжатых данных. #4186 (alesapin
- Исправлена ошибка, приводящая к неправильному результату сравнения больших (не помещающихся (в `Int16`) дат при включённой настройке `compile_expressions`. #4341 (alesapin
- (Исправлен бесконечный цикл при запросе из табличной функции `numbers(0)`. #4280 (alexey-milovidov
- Временно отключён `pushdown` предикатов в подзапрос, если он содержит `ORDER BY`. #3890 (Winter (Zhang



- Исправлена ошибка `Illegal instruction` при использовании функций для работы с base64 на старых CPU. (Ошибка проявлялась только, если ClickHouse был скомпилирован с gcc-8. [#4275 \(alexey-milovidov\)](#)
- Исправлена ошибка `No message received` при запросах к PostgreSQL через ODBC-драйвер и TLS-соединение, исправлен segfault при использовании MySQL через ODBC-драйвер. [#4170 \(alexey-milovidov\)](#)
- Исправлен неверный результат при использовании значений типа `Date` или `DateTime` в ветвях условного оператора (функции `if`). Функция `if` теперь работает для произвольного типа значений в ветвях. [#4243 \(alexey-milovidov\)](#)
- Словари с источником из локального ClickHouse теперь исполняются локально, а не используя TCP-соединение. [#4166 \(alexey-milovidov\)](#)
- Исправлено зависание запросов к таблице с движком `File` после того, как `SELECT` из этой таблицы (завершился с ошибкой `No such file or directory`. [#4161 \(alexey-milovidov\)](#)
- Исправлена ошибка, из-за которой при запросе к таблице `system.tables` могло возникать исключение `(table doesn't exist)`. [#4313 \(alexey-milovidov\)](#)
- Исправлена ошибка, приводившая к падению `clickhouse-client` в интерактивном режиме, если успеть (выйти из него во время загрузки подсказок командной строки. [#4317 \(alexey-milovidov\)](#)
- Исправлена ошибка, приводившая к неверным результатам исполнения мутаций, содержащих (оператор `IN`. [#4099 \(Alex Zatelepin\)](#)
- Исправлена ошибка, из-за которой, если была создана база данных с движком `Dictionary`, все словари загружались при старте сервера, а словари с источником из локального ClickHouse не могли (загрузиться. [#4255 \(alexey-milovidov\)](#)
- Исправлено повторное создание таблиц с системными логами (`system.query_log`, `system.part_log`) при (остановке сервера. [#4254 \(alexey-milovidov\)](#)
- Исправлен вывод типа возвращаемого значения, а также использование блокировок в функции `joinGet`. [#4153 \(Amos Bird\)](#)
- Исправлено падение сервера при использовании настройки `allow_experimental_multiple_joins_emulation`. [52de2c \(Artem Zuikov\)](#)
- (Исправлено некорректное сравнение значений типа `Date` и `DateTime`. [#4237 \(alexey-milovidov\)](#)
- Исправлена ошибка, проявлявшаяся при fuzz-тестировании с undefined behaviour-санитайзером: добавлена проверка типов параметров для семейства функций `quantile*Weighted`. [#4145 \(alexey-milovidov\)](#)
- Исправлена редкая ошибка, из-за которой при удалении старых кусков данных может возникать (ошибка `File not found`. [#4378 \(alexey-milovidov\)](#)
- Исправлена установка пакета при отсутствующем файле `/etc/clickhouse-server/config.xml`. [#4343 \(\(proller\)](#)

## :Улучшения сборки/тестирования/пакетирования

- При установке Debian-пакета символическая ссылка `/etc/clickhouse-server/preprocessed` теперь (создаётся, учитывая пути, прописанные в конфигурационном файле. [#4205 \(proller\)](#)
- (Исправления сборки под FreeBSD. [#4225 \(proller\)](#)
- Добавлена возможность создавать, заполнять и удалять таблицы в тестах производительности. [#4220 \(alesapin\)](#)
- Добавлен скрипт для поиска дублирующихся `include`-директив в исходных файлах. [#4326 \(alexey-milovidov\)](#)
- (В тестах производительности добавлена возможность запускать запросы по номеру. [#4264 \(alesapin\)](#)
- Пакет с `debug`-символами добавлен в список рекомендованных для основного пакета. [#4274 \(alexey-milovidov\)](#)
- (Рефакторинг утилиты `performance-test`. Улучшено логирование и обработка сигналов. [#4171 \(alesapin\)](#)
- (ЗадOCUMENTИРОВАН анонимизированный датасет Яндекс.Метрики. [#4164 \(alesapin\)](#)
- Добавлен инструмент для конвертирования кусков данных таблиц, созданных с использованием (старого синтаксиса с помесечным партиционированием, в новый формат. [#4195 \(Alex Zatelepin\)](#)
- (Добавлена документация для двух датасетов, загруженных в S3. [#4144 \(alesapin\)](#)
- Добавлен инструмент, собирающий changelog из описаний pull request-ов. [#4169 #4173 \(\(KochetovNicolai\) \(KochetovNicolai\)](#)
- (Добавлен puppet-модуль для Clickhouse. [#4182 \(Maxim Fedotov\)](#)
- (Добавлена документация для нескольких недокументированных функций. [#4168 \(Winter Zhang\)](#)

(Исправления сборки под ARM. [#4210#4306 #4291 \(proller\)](#) (proller) •  
(Добавлена возможность запускать тесты словарей из `ctest`. [#4189 \(proller\)](#) •  
(Теперь директорией с SSL-сертификатами по умолчанию является `/etc/ssl`. [#4167 \(alexey-milovidov\)](#) •  
(Добавлена проверка доступности SSE и AVX-инструкций на старте. [#4234 \(lgr\)](#) •  
(Init-скрипт теперь дожидается, пока сервер запустится. [#4281 \(proller\)](#) •

## :Обратно несовместимые изменения

Удалена настройка `allow_experimental_low_cardinality_type`. Семейство типов данных `LowCardinality` готово •  
(для использования в production. [#4323 \(alexey-milovidov\)](#) •  
Размер кэша засечек и кэша разжатых блоков теперь уменьшается в зависимости от доступного •  
(объёма памяти. [#4240 \(Lopatin Konstantin\)](#) •  
Для запроса `CREATE TABLE` добавлено ключевое слово `INDEX`. Имя столбца `index` теперь надо оборачивать •  
(в двойные или обратные кавычки: ``index``. [#4143 \(Nikita Vasilev\)](#) •  
Функция `sumMap` теперь возвращает тип с большей областью значений вместо переполнения. Если •  
необходимо старое поведение, следует использовать добавленную функцию `sumMapWithOverflow`. •  
([#4151 \(Léo Ercolanelli\)](#)

## :Улучшения производительности

(Для запросов без секции `LIMIT` вместо `std::sort` теперь используется `pdqsort`. [#4236 \(Evgenii Pravda\)](#) •  
Теперь сервер переиспользует потоки для выполнения запросов из глобального пула потоков. В •  
(краевых случаях это влияет на производительность. [#4150 \(alexey-milovidov\)](#)

## :Улучшения

Теперь, если в нативном протоколе послать запрос `INSERT INTO tbl VALUES (...)` (с данными в запросе), •  
(отдельно посылать разобранные данные для вставки не нужно. [#4301 \(alesapin\)](#) •  
(Добавлена поддержка AIO для FreeBSD. [#4305 \(urgordeadbeef\)](#) •  
Запрос `SELECT * FROM a JOIN b USING a, b` теперь возвращает столбцы `a` и `b` только из левой таблицы. •  
([#4141 \(Artem Zuikov\)](#) •  
Добавлена опция командной строки `-C` для клиента, которая работает так же, как и опция `-c`. [#4232](#) •  
([\(syominsergey\)](#) •  
Если для опции `--password` клиента командной строки не указано значение, пароль запрашивается из •  
(стандартного входа. [#4230 \(BSD\\_Conqueror\)](#) •  
Добавлена подсветка метасимволов в строковых литералах, содержащих выражения для оператора •  
(`LIKE` и регулярные выражения. [#4327 \(alexey-milovidov\)](#) •  
(Добавлена отмена HTTP-запроса, если сокет клиента отваливается. [#4213 \(nvartolomei\)](#) •  
Теперь сервер время от времени посылает пакеты `Progress` для поддержания соединения. [#4215](#) •  
([\(Ivan\)](#) •  
Немного улучшено сообщение о причине, почему запрос `OPTIMIZE` не может быть исполнен (если •  
(включена настройка `optimize_throw_if_noop`). [#4294 \(alexey-milovidov\)](#) •  
(Добавлена поддержка опции `--version` для `clickhouse-server`. [#4251 \(Lopatin Konstantin\)](#) •  
(Добавлена поддержка опции `--help/-h` для `clickhouse-server`. [#4233 \(Yuriy Baranov\)](#) •  
Добавлена поддержка скалярных подзапросов, возвращающих состояние агрегатной функции. [#4348](#) •  
([\(Nikolai Kochetov\)](#) •  
Уменьшено время ожидания завершения сервера и завершения запросов `ALTER`. [#4372 \(alexey-](#) •  
([milovidov\)](#) •  
Добавлена информация о значении настройки `replicated_can_become_leader` в таблицу `system.replicas`. •  
(Добавлено логирование в случае, если реплика не собирается стать лидером. [#4379 \(Alex Zatelepin\)](#)

## ClickHouse release 19.1.14, 2019-03-14

Исправлена ошибка `Column ... queried more than once`, которая могла произойти в случае включенной •  
настройки `asterisk_left_columns_only` в случае использования `GLOBAL JOIN` а также `SELECT *` (редкий случай). •  
(Эта ошибка изначально отсутствует в версиях 19.3 и более новых. [6bac7d8d \(Artem Zuikov\)](#)

## ClickHouse release 19.1.13, 2019-03-12

.Этот релиз содержит такие же исправления ошибок, как и 19.3.7

## ClickHouse release 19.1.10, 2019-03-03

.Этот релиз содержит такие же исправления ошибок, как и 19.3.6

## ClickHouse release 19.1.9, 2019-02-21

### :Исправления ошибок

Исправлена обратная несовместимость со старыми версиями, появившаяся из-за некорректной

(реализации настройки `send_logs_level`. #4445 (alexey-milovidov

Исправлена обратная несовместимость табличной функции `remote`, появившаяся из-за добавления  
(комментариев колонок. #4446 (alexey-milovidov

## ClickHouse release 19.1.8, 2019-02-16

### :Исправления ошибок

Исправлена установка пакета при отсутствующем файле `/etc/clickhouse-server/config.xml`. #4343  
(proller

## ClickHouse release 19.1.7, 2019-02-15

### :Исправления ошибок

Исправлен вывод типа возвращаемого значения, а также использование блокировок в функции `joinGet`.  
(#4153 (Amos Bird

Исправлено повторное создание таблиц с системными логами (`system.query_log`, `system.part_log`) при  
(остановке сервера. #4254 (alexey-milovidov

Исправлена ошибка, из-за которой, если была создана база данных с движком `Dictionary`, все словари  
загружались при старте сервера, а словари с источником из локального ClickHouse не могли  
(загрузиться. #4255 (alexey-milovidov

Исправлена ошибка, приводившая к неверным результатам исполнения мутаций, содержащих  
(оператор `IN`. #4099 (Alex Zatelepin

Исправлена ошибка, приводившая к падению `clickhouse-client` в интерактивном режиме, если успеть  
(выйти из него во время загрузки подсказок командной строки. #4317 (alexey-milovidov

Исправлена ошибка, из-за которой при запросе к таблице `system.tables` могло возникать исключение  
(`table doesn't exist`. #4313 (alexey-milovidov

Исправлено зависание запросов к таблице с движком `File` после того, как `SELECT` из этой таблицы  
(завершился с ошибкой `No such file or directory`. #4161 (alexey-milovidov

Словари с источником из локального ClickHouse теперь исполняются локально, а не используя TCP-  
(соединение. #4166 (alexey-milovidov

Исправлена ошибка `No message received` при запросах к PostgreSQL через ODBC-драйвер и TLS-  
соединение, исправлен `segfault` при использовании MySQL через ODBC-драйвер. #4170 (alexey-  
(milovidov

Временно отключён `pushdown` предикатов в подзапрос, если он содержит `ORDER BY`. #3890 (Winter  
(Zhang

(Исправлен бесконечный цикл при запросе из табличной функции `numbers(0)`. #4280 (alexey-milovidov

Исправлена ошибка, приводящая к неправильному результату сравнения больших (не помещающихся  
(в `Int16`) дат при включённой настройке `compile_expressions`. #4341 (alesapin

Исправлено падение сервера при включённой опции `uncompressed_cache`, а также исключение о  
(неправильном размере разжатых данных. #4186 (alesapin

Исправлен неверный результат `ALL JOIN`, если в правой таблице присутствуют дубликаты ключа `join`.  
(#4184 (Artem Zuikov

Исправлена ошибка, приводившая к тому, что при исполнении запроса `INSERT ... SELECT ... FROM file(...)`  
терялась первая строка файла, если он был в формате `CSVWithNames` или `TSVWithNames`. #4297 (alexey-  
(milovidov

(Исправлена агрегация столбцов типа `Array(LowCardinality)`. #4055 (KochetovNicolai



- При установке Debian-пакета символическая ссылка /etc/clickhouse-server/preprocessed теперь (создаётся, учитывая пути, прописанные в конфигурационном файле. [#4205](#) ([proller](#)
- Исправлена ошибка, проявлявшаяся при fuzz-тестировании с undefined behaviour-санитайзером: добавлена проверка типов параметров для семейства функций [quantile\\*Weighted](#). [#4145](#) ([alexey-milovidov](#)
- Команда [START REPLICATED SENDS](#) теперь действительно включает посылку кусков данных при (репликации. [#4229](#) ([nvartolomei](#)
- Исправлена ошибка [Not found column](#) для случая дублирующихся столбцов в секции [JOIN ON](#). [#4279](#) ([Artem Zuikov](#)
- (Теперь директорией с SSL-сертификатами по умолчанию является [/etc/ssl](#). [#4167](#) ([alexey-milovidov](#)
- Исправлено падение при перезагрузке внешнего словаря, если словарь недоступен. Ошибка возникла (в 19.1.6. [#4188](#) ([proller](#)
- (Исправлено некорректное сравнение значений типа [Date](#) и [DateTime](#). [#4237](#) ([valexey](#)
- Исправлен неверный результат при использовании значений типа [Date](#) или [DateTime](#) в ветвях условного оператора (функции [if](#)). Функция [if](#) теперь работает для произвольного типа значений в ветвях. [#4243](#) ([alexey-milovidov](#)

## ClickHouse release 19.1.6, 2019-01-24

### :Новые возможности

- (Задание формата сжатия для отдельных столбцов. [#3899](#) [#4111](#) ([alesapin](#), [Winter Zhang](#), [Anatoly](#)
- (Формат сжатия [Delta](#). [#4052](#) ([alesapin](#)
- (Изменение формата сжатия запросом [ALTER](#). [#4054](#) ([alesapin](#)
- Добавлены функции [left](#), [right](#), [trim](#), [ltrim](#), [rtrim](#), [timestampadd](#), [timestampsub](#) для совместимости со (стандартом SQL. [#3826](#) ([Ivan Blinkov](#)
- (Поддержка записи в движок [HDFS](#) и табличную функцию [hdfs](#). [#4084](#) ([alesapin](#)
- Добавлены функции поиска набора константных строк в тексте: [multiPosition](#), [multiSearch](#), [firstMatch](#) также (с суффиксами [-UTF8](#), [-CaseInsensitive](#), и [-CaseInsensitiveUTF8](#). [#4053](#) ([Danila Kutenin](#)
- Пропуск неиспользуемых шардов в случае, если запрос [SELECT](#) содержит фильтрацию по ключу (шардирования (настройка [optimize\\_skip\\_unused\\_shards](#)). [#3851](#) ([Gleb Kanterov](#), [Ivan](#)
- Пропуск строк в случае ошибки парсинга для движка [Kafka](#) (настройка [kafka\\_skip\\_broken\\_messages](#)). [#4094](#) ([Ivan](#)
- Поддержка применения мультиклассовых моделей [CatBoost](#). Функция [modelEvaluate](#) возвращает кортеж в случае использования мультиклассовой модели. [libcatboostmodel.so](#) should be built with [#607](#). [#3959](#) ([KochetovNicolai](#)
- (Добавлены функции [filesystemAvailable](#), [filesystemFree](#), [filesystemCapacity](#). [#4097](#) ([Boris Granveaud](#)
- (Добавлены функции хеширования [xxHash64](#) и [xxHash32](#). [#3905](#) ([filimonov](#)
- Добавлена функция хеширования [gccMurmurHash](#) (GCC flavoured Murmur hash), использующая те же (hash seed, что и [gcc](#) [#4000](#) ([sundyli](#)
- (Добавлены функции хеширования [javaHash](#), [hiveHash](#). [#3811](#) ([shangshujie365](#)
- Добавлена функция [remoteSecure](#). Функция работает аналогично [remote](#), но использует безопасное (соединение. [#4088](#) ([proller](#)

### :Экспериментальные возможности

- Эмуляция запросов с несколькими секциями [JOIN](#) (настройка [allow\\_experimental\\_multiple\\_joins\\_emulation](#)). ([#3946](#) ([Artem Zuikov](#)

### :Исправления ошибок

- Ограничен размер кеша скомпилированных выражений в случае, если не указана настройка ([compiled\\_expression\\_cache\\_size](#) для экономии потребляемой памяти. [#4041](#) ([alesapin](#)
- Исправлена проблема зависания потоков, выполняющих запрос [ALTER](#) для таблиц семейства [Replicated](#), (а также потоков, обновляющих конфигурацию из ZooKeeper. [#2947](#) [#3891](#) [#3934](#) ([Alex Zatelepin](#)
- Исправлен race condition в случае выполнения распределенной задачи запроса [ALTER](#). Race condition приводил к состоянию, когда более чем одна реплика пыталась выполнить задачу, в результате чего (все такие реплики, кроме одной, падали с ошибкой обращения к ZooKeeper. [#3904](#) ([Alex Zatelepin](#)

- Исправлена проблема обновления настройки `from_zk`. Настройка, указанная в файле конфигурации, не (обновлялась в случае, если запрос к ZooKeeper падал по timeout. [#2947](#) [#3947](#) (Alex Zatelepin
- Исправлена ошибка в вычислении сетевого префикса при указании IPv4 маски подсети. [#3945](#) ((alesapin
- Исправлено падение (`std::terminate`) в редком сценарии, когда новый поток не мог быть создан из-за (нехватки ресурсов. [#3956](#) (alexey-milovidov
- Исправлено падение табличной функции `remote` в случае, когда не удавалось получить структуру (таблицы из-за ограничений пользователя. [#4009](#) (alesapin
- Исправлена утечка сетевых сокетов. Сокеты создавались в пуле и никогда не закрывались. При создании потока, создавались новые сокеты в случае, если все доступные использовались. [#4017](#) ((Alex Zatelepin
- Исправлена проблема закрывания `/proc/self/fd` раньше, чем все файловые дескрипторы были прочитаны (из `/proc` после создания процесса `odbc-bridge`. [#4120](#) (alesapin
- Исправлен баг в монотонном преобразовании String в UInt в случае использования String в первичном (ключе. [#3870](#) (Winter Zhang
- Исправлен баг в вычислении монотонности функции преобразования типа целых значений. [#3921](#) ((alexey-milovidov
- Исправлено падение в функциях `arrayEnumerateUniq`, `arrayEnumerateDense` при передаче невалидных (аргументов. [#3909](#) (alexey-milovidov
- (Исправлен undefined behavior в StorageMerge. [#3910](#) (Amos Bird
- (Исправлено падение в функциях `addDays`, `subtractDays`. [#3913](#) (alexey-milovidov
- Исправлена проблема, в результате которой функции `round`, `floor`, `trunc`, `ceil` могли возвращать неверный (результат для отрицательных целочисленных аргументов с большим значением. [#3914](#) (alexey-milovidov
- Исправлена проблема, в результате которой 'kill query sync' приводил к падению сервера. [#3916](#) ((muVulDeePecker
- Исправлен баг, приводящий к большой задержке в случае пустой очереди репликации. [#3928](#) [#3932](#) ((alesapin
- Исправлено избыточное использование памяти в случае вставки в таблицу с `LowCardinality` в первичном (ключе. [#3955](#) (KochetovNicolai
- Исправлена сериализация пустых массивов типа `LowCardinality` для формата `Native`. [#3907](#) [#4011](#) ((KochetovNicolai
- Исправлен неверный результат в случае использования `distinct` для числового столбца `LowCardinality`. ([#3895](#) [#4012](#) (KochetovNicolai
- Исправлена компиляция вычисления агрегатных функций для ключа `LowCardinality` (для случая, когда (включена настройка `compile`). [#3886](#) (KochetovNicolai
- (Исправлена передача пользователя и пароля для запросов с реплик. [#3957](#) (alesapin) (小路
- Исправлен очень редкий race condition возникающий при перечислении таблиц из базы данных типа (`Dictionary` во время перезагрузки словарей. [#3970](#) (alexey-milovidov
- Исправлен неверный результат в случае использования HAVING с ROLLUP или CUBE. [#3756](#) [#3837](#) ((Sam Chou
- Исправлена проблема с алиасами столбцов для запросов с `JOIN ON` над распределенными таблицами. ([#3980](#) (Winter Zhang
- Исправлена ошибка в реализации функции `quantileTDigest` (нашел Artem Vakhrushev). Эта ошибка (никогда не происходит в ClickHouse и актуальна только для тех, кто использует кодовую базу (ClickHouse напрямую в качестве библиотеки. [#3935](#) (alexey-milovidov

## :Улучшения

- Добавлена поддержка `IF NOT EXISTS` в выражении `ALTER TABLE ADD COLUMN`, `IF EXISTS` в выражении (`DROP/MODIFY/CLEAR/COMMENT COLUMN`. [#3900](#) (Boris Granveaud
- Функция `parseDateTimeBestEffort` теперь поддерживает форматы `DD.MM.YYYY`, `DD.MM.YY`, `DD-MM-YYYY`, `DD-Mon-(YYYY`, `DD/Month/YYYY` и аналогичные. [#3922](#) (alexey-milovidov
- (`CapnProtoInputStream` теперь поддерживает jagged структуры. [#4063](#) (Odin Hultgren Van Der Horst
- Улучшение usability: добавлена проверка, что сервер запущен от пользователя, совпадающего с (владельцем директории данных. Запрещен запуск от пользователя root в случае, если root не владеет (директорией с данными. [#3785](#) (sergey-v-galtsev

- Улучшена логика проверки столбцов, необходимых для JOIN, на стадии анализа запроса. [#3930 \(Artem Zuikov\)](#)
- Уменьшено число поддерживаемых соединений в случае большого числа распределенных таблиц. [#3726 \(Winter Zhang\)](#)
- Добавлена поддержка строки с totals для запроса с **WITH TOTALS** через ODBC драйвер. [#3836 \(Maksim Koritckiy\)](#)
- (Поддержано использование **Enum** в качестве чисел в функции **if**. [#3875 \(Ivan\)](#)
- Добавлена настройка **low\_cardinality\_allow\_in\_native\_format**. Если она выключена, то тип **LowCardinality** не (используется в формате **Native**. [#3879 \(KochetovNicolai\)](#)
- Удалены некоторые избыточные объекты из кеша скомпилированных выражений для уменьшения (потребления памяти. [#4042 \(alesapin\)](#)
- Добавлена проверка того, что в запрос **SET send\_logs\_level = 'value'** передается верное значение. [#3873 \(\(Sabyanin Maxim\)](#)
- (Добавлена проверка типов для функций преобразования типов. [#3896 \(Winter Zhang\)](#)

## :Улучшения производительности

- Добавлена настройка **use\_minimalistic\_part\_header\_in\_zookeeper** для движка MergeTree. Если настройка включена, Replicated таблицы будут хранить метаданные куски в компактном виде (в соответствующем znode для этого куски). Это может значительно уменьшить размер для ZooKeeper snapshot (особенно для таблиц с большим числом столбцов). После включения данной настройки будет невозможно сделать откат к версии, которая эту настройку не поддерживает. [#3960 \(Alex Zatelepin\)](#)
- Добавлена реализация функций **sequenceMatch** и **sequenceCount** на основе конечного автомата в случае, (если последовательность событий не содержит условия на время. [#4004 \(Léo Ercolanelli\)](#)
- (Улучшена производительность сериализации целых чисел. [#3968 \(Amos Bird\)](#)
- Добавлен zero left padding для PODArray. Теперь элемент с индексом -1 является валидным нулевым значением. Эта особенность используется для удаления условного выражения при вычислении (оффсетов массивов. [#3920 \(Amos Bird\)](#)
- (Откат версии **jemalloc**, приводящей к деградации производительности. [#4018 \(alexey-milovidov\)](#)

## :Обратно несовместимые изменения

- Удалена недокументированная возможность **ALTER MODIFY PRIMARY KEY**, замененная выражением **ALTER (MODIFY ORDER BY**. [#3887 \(Alex Zatelepin\)](#)
- (Удалена функция **shardByHash**. [#3833 \(alexey-milovidov\)](#)
- Запрещено использование скалярных подзапросов с результатом, имеющим тип **AggregateFunction**. [#3865 \(Ivan\)](#)

## :Улучшения сборки/тестирования/пакетирования

- (Добавлена поддержка сборки под PowerPC (**ppc64le**). [#4132 \(Danila Kutenin\)](#)
- (Функциональные stateful тесты запускаются на публично доступных данных. [#3969 \(alexey-milovidov\)](#)
- Исправлена ошибка, при которой сервер не мог запуститься с сообщением **bash: /usr/bin/clickhouse-extract-from-config: Operation not permitted** при использовании Docker или systemd-nspawn. [#4136 \(alexey-milovidov\)](#)
- Обновлена библиотека **rdkafka** до версии v1.0.0-RC5. Использована **srpkafka** на замену интерфейса (языка C. [#4025 \(Ivan\)](#)
- Обновлена библиотека **mariadb-client**. Исправлена проблема, обнаруженная с использованием UBSan. [#3924 \(alexey-milovidov\)](#)
- (Исправления для сборок с UBSan. [#3926](#) [#3021](#) [#3948 \(alexey-milovidov\)](#)
- .Добавлены покоммитные запуски тестов с UBSan сборкой
- .Добавлены покоммитные запуски тестов со статическим анализатором PVS-Studio
- (Исправлены проблемы, найденные с использованием PVS-Studio. [#4013 \(alexey-milovidov\)](#)
- (Исправлены проблемы совместимости glibc. [#4100 \(alexey-milovidov\)](#)
- (Docker образы перемещены на Ubuntu 18.10, добавлена совместимость с glibc >= 2.28 [#3965 \(alesapin\)](#)
- Добавлена переменная окружения **CLICKHOUSE\_DO\_NOT\_CHOWN**, позволяющая не делать shown (директории для Docker образа сервера. [#3967 \(alesapin\)](#)

- Включены большинство предупреждений из `-Weverything` для clang. Включено `-Wpedantic`. [#3986](#) (alexey-milovidov)
- (Добавлены некоторые предупреждения, специфичные только для clang 8. [#3993](#) (alexey-milovidov)
- При использовании динамической линковки используется `libLLVM` вместо библиотеки `LLVM`. [#3989](#) (Orivej Desh)
- Добавлены переменные окружения для параметров `TSan`, `UBSan`, `ASan` в тестовом Docker образе. [#4072](#) (alesapin)
- Debian пакет `clickhouse-server` будет рекомендовать пакет `libcap2-bin` для того, чтобы использовать утилиту `setcap` для настроек. Данный пакет опционален. [#4093](#) (alexey-milovidov)
- (Уменьшено время сборки, убраны ненужные включения заголовочных файлов. [#3898](#) (proller)
- (Добавлены тесты производительности для функций хеширования. [#3918](#) (filimonov)
- (Исправлены циклические зависимости библиотек. [#3958](#) (proller)
- (Улучшена компиляция при малом объеме памяти. [#4030](#) (proller)
- Добавлен тестовый скрипт для воспроизведения деградации производительности в `jemalloc`. [#4036](#) (alexey-milovidov)
- (Исправления опечаток в комментариях и строковых литералах. [#4122](#) (maiha)
- (Исправления опечаток в комментариях. [#4089](#) (Evgenii Pravda)

## ClickHouse release 18.16.1, 2018-12-21

### :Исправления ошибок

- Исправлена проблема, приводившая к невозможности обновить словари с источником ODBC. [#3825](#), [#3829](#)
- JIT-компиляция агрегатных функций теперь работает с LowCardinality столбцами. [#3838](#)

### :Улучшения

- Добавлена настройка `low_cardinality_allow_in_native_format` (по умолчанию включена). Если её выключить, столбцы LowCardinality в Native формате будут преобразовываться в соответствующий обычный тип при SELECT и из этого типа при INSERT. [#3879](#)

### :Улучшения сборки

- .Исправления сборки под macOS и ARM

## ClickHouse release 18.16.0, 2018-12-14

### :Новые возможности

- Вычисление `DEFAULT` выражений для отсутствующих полей при загрузке данных в полуструктурированных форматах (`JSONEachRow`, `TSKV`) (требуется включить настройку запроса `insert_sample_with_metadata`). [#3555](#)
- Для запроса `ALTER TABLE` добавлено действие `MODIFY ORDER BY` для изменения ключа сортировки при одновременном добавлении или удалении столбца таблицы. Это полезно для таблиц семейства `MergeTree`, выполняющих дополнительную работу при слияниях, согласно этому ключу сортировки, как например, `SummingMergeTree`, `AggregatingMergeTree` и т. п. [#3581](#) [#3755](#)
- Для таблиц семейства `MergeTree` появилась возможность указать различный ключ сортировки (`ORDER BY`) и индекс (`PRIMARY KEY`). Ключ сортировки может быть длиннее, чем индекс. [#3581](#)
- Добавлена табличная функция `hdfs` и движок таблиц `HDFS` для импорта и экспорта данных в HDFS. chenxing-xc
- Добавлены функции для работы с base64: `base64Encode`, `base64Decode`, `tryBase64Decode`. Alexander Krashennnikov
- Для агрегатной функции `uniqCombined` появилась возможность настраивать точность работы с помощью параметра (выбирать количество ячеек HyperLogLog). [#3406](#)
- Добавлена таблица `system.contributors`, содержащая имена всех, кто делал коммиты в ClickHouse. [#3452](#)
- Добавлена возможность не указывать партицию для запроса `ALTER TABLE ... FREEZE` для бэкапа сразу всех партиций. [#3514](#)
- Добавлены функции `dictGet`, `dictGetOrDefault` без указания типа возвращаемого значения. Тип определяется автоматически из описания словаря. Amos Bird

Возможность указания комментария для столбца в описании таблицы и изменения его с помощью **ALTER**. [#3377](#)

Возможность чтения из таблицы типа **Join** в случае простых ключей. [Amos Bird](#)

Возможность указания настроек **join\_use\_nulls**, **max\_rows\_in\_join**, **max\_bytes\_in\_join**, **join\_overflow\_mode** при создании таблицы типа **Join**. [Amos Bird](#)

Добавлена функция **joinGet**, позволяющая использовать таблицы типа **Join** как словарь. [Amos Bird](#)

Добавлены столбцы **partition\_key**, **sorting\_key**, **primary\_key**, **sampling\_key** в таблицу **system.tables**, позволяющие получить информацию о ключах таблицы. [#3609](#)

Добавлены столбцы **is\_in\_partition\_key**, **is\_in\_sorting\_key**, **is\_in\_primary\_key**, **is\_in\_sampling\_key** в таблицу **system.columns**. [#3609](#)

Добавлены столбцы **min\_time**, **max\_time** в таблицу **system.parts**. Эти столбцы заполняются, если ключ партиционирования является выражением от столбцов типа **DateTime**. [Emmanuel Donin de Rosière](#)

## :Исправления ошибок

Исправления и улучшения производительности для типа данных **LowCardinality**. **GROUP BY** по **LowCardinality(Nullable(...))**. Получение **extremes** значений. Выполнение функций высшего порядка. **LEFT ARRAY JOIN**. Распределённый **GROUP BY**. Функции, возвращающие **Array**. Выполнение **ORDER BY**. Запись в **Distributed** таблицы ([nicelulu](#)). Обратная совместимость для запросов **INSERT** от старых клиентов, реализующих **Native** протокол. Поддержка **LowCardinality** для **JOIN**. Производительность при работе в один поток. [#3823](#) [#3803](#) [#3799](#) [#3769](#) [#3744](#) [#3681](#) [#3651](#) [#3649](#) [#3641](#) [#3632](#) [#3568](#) [#3523](#) [#3518](#)

Исправлена работа настройки **select\_sequential\_consistency**. Ранее, при включенной настройке, после начала записи в новую партицию, мог возвращаться неполный результат. [#2863](#)

Корректное указание базы данных при выполнении DDL запросов **ON CLUSTER**, а также при выполнении **ALTER UPDATE/DELETE**. [#3772](#) [#3460](#)

Корректное указание базы данных для подзапросов внутри **VIEW**. [#3521](#)

Исправлена работа **PREWHERE** с **FINAL** для **VersionedCollapsingMergeTree**. [7167bfd7](#)

Возможность с помощью запроса **KILL QUERY** отмены запросов, которые ещё не начали выполняться из-за ожидания блокировки таблицы. [#3517](#)

Исправлены расчёты с датой и временем в случае, если стрелки часов были переведены назад в полночь (это происходит в Иране, а также было в Москве с 1981 по 1983 год). Ранее это приводило к тому, что стрелки часов переводились на сутки раньше, чем нужно, а также приводило к некорректному форматированию даты-с-временем в текстовом виде. [#3819](#)

Исправлена работа некоторых случаев **VIEW** и подзапросов без указания базы данных. [Winter Zhang](#)

Исправлен race condition при одновременном чтении из **MATERIALIZED VIEW** и удалением **MATERIALIZED VIEW** из-за отсутствия блокировки внутренней таблицы **MATERIALIZED VIEW**. [#3404](#) [#3694](#)

Исправлена ошибка **Lock handler cannot be nullptr**. [#3689](#)

Исправления выполнения запросов при включенной настройке **compile\_expressions** (включена по умолчанию) - убрана свёртка недетерминированных константных выражений, как например, функции **now**. [#3457](#)

.Исправлено падение при указании неконстантного аргумента **scale** в функциях **toDecimal32/64/128**

Исправлена ошибка при попытке вставки в формате **Values** массива с **NULL** элементами в столбец типа **Array** без **Nullable** (в случае **input\_format\_values\_interpret\_expressions = 1**). [#3487](#) [#3503](#)

Исправлено непрерывное логгирование ошибок в **DDLWorker**, если ZooKeeper недоступен. [8f50c620](#)

Исправлен тип возвращаемого значения для функций **quantile\*** от аргументов типа **Date** и **DateTime**. [#3580](#)

Исправлена работа секции **WITH**, если она задаёт простой алиас без выражений. [#3570](#)

Исправлена обработка запросов с именованными подзапросами и квалифицированными именами столбцов при включенной настройке **enable\_optimize\_predicate\_expression**. [Winter Zhang](#)

Исправлена ошибка **Attempt to attach to nullptr thread group** при работе материализованных представлений. [Marek Vavruša](#)

Исправлено падение при передаче некоторых некорректных аргументов в функцию **arrayReverse**. [73e3a7b6](#)

Исправлен buffer overflow в функции **extractURLParameter**. Увеличена производительность. Добавлена корректная обработка строк, содержащих нулевые байты. [141e9799](#)

Исправлен buffer overflow в функциях **lowerUTF8**, **upperUTF8**. Удалена возможность выполнения этих функций над аргументами типа **FixedString**. [#3662](#)



- Исправлен редкий race condition при удалении таблиц типа MergeTree. #3680
- Исправлен race condition при чтении из таблиц типа Buffer и одновременном ALTER либо DROP таблиц назначения. #3719
- Исправлен segfault в случае превышения ограничения max\_temporary\_non\_const\_columns. #3788

## :Улучшения

- Обработанные конфигурационные файлы записываются сервером не в /etc/clickhouse-server/ директорию, а в директорию preprocessed\_configs внутри path. Это позволяет оставить директорию /etc/clickhouse-server/ недоступной для записи пользователем clickhouse, что повышает безопасность. #2443
- Настройка min\_merge\_bytes\_to\_use\_direct\_io выставлена по-умолчанию в 10 GiB. Слияния, образующие крупные куски таблиц семейства MergeTree, будут производиться в режиме O\_DIRECT, что исключает вымывание кэша. #3504
- Ускорен запуск сервера в случае наличия очень большого количества таблиц. #3398
- Добавлен пул соединений и HTTP Keep-Alive для соединения между репликами. #3594
- В случае ошибки синтаксиса запроса, в HTTP интерфейсе возвращается код 400 Bad Request (ранее возвращался код 500). 31bc680a
- Для настройки join\_default\_strictness выбрано значение по-умолчанию ALL для совместимости. 120e2cbe
- Убрано логгирование в stderr из библиотеки re2 в случае некорректных или сложных регулярных выражений. #3723
- Для движка таблиц Kafka: проверка наличия подписок перед началом чтения из Kafka; настройка таблицы kafka\_max\_block\_size. Marek Vavruša
- Функции cityHash64, farmHash64, metroHash64, sipHash64, halfMD5, murmurHash2\_32, murmurHash2\_64, murmurHash3\_32, murmurHash3\_64 теперь работают для произвольного количества аргументов, а также для аргументов-кортежей. #3451 #3519
- Функция arrayReverse теперь работает с любыми типами массивов. 73e3a7b6
- Добавлен опциональный параметр - размер слота для функции timeSlots. Kirill Shvakov
- Для FULL и RIGHT JOIN учитывается настройка max\_block\_size для потока неприсоединённых данных из правой таблицы. Amos Bird
- В clickhouse-benchmark и clickhouse-performance-test добавлен параметр командной строки --secure для включения TLS. #3688 #3690
- Преобразование типов в случае, если структура таблицы типа Buffer не соответствует структуре таблицы назначения. Vitaly Baranov
- Добавлена настройка tcp\_keep\_alive\_timeout для включения keep-alive пакетов после неактивности в течение указанного интервала времени. #3441
- Убрано излишнее квотирование значений ключа партиции в таблице system.parts, если он состоит из одного столбца. #3652
- Функция деления с остатком работает для типов данных Date и DateTime. #3385
- Добавлены синонимы функций POWER, LN, LCASE, UCASE, REPLACE, LOCATE, SUBSTR, MID. #3774 #3763
- Некоторые имена функций сделаны регистронезависимыми для совместимости со стандартом SQL.
- Добавлен синтаксический сахар SUBSTRING(expr FROM start FOR length) для совместимости с SQL. #3804
- Добавлена возможность фиксации (mlock) страниц памяти, соответствующих исполняемому коду clickhouse-server для предотвращения вытеснения их из памяти. Возможность выключена по-умолчанию. #3553
- Увеличена производительность чтения с O\_DIRECT (с включенной опцией min\_bytes\_to\_use\_direct\_io). #3405
- Улучшена производительность работы функции dictGet...OrDefault в случае константного аргумента-ключа и неконстантного аргумента-default. Amos Bird
- В функции firstSignificantSubdomain добавлена обработка доменов gov, mil, edu. Igor Hatarist Увеличена производительность работы. #3628
- Возможность указания произвольных переменных окружения для запуска clickhouse-server посредством .SYS-V init.d-скрипта с помощью указания CLICKHOUSE\_PROGRAM\_ENV в /etc/default/clickhouse Pavlo Bashynskyi
- Правильный код возврата init-скрипта clickhouse-server. #3516
- В таблицу system.metrics добавлена метрика VersionInteger, а в system.build\_options добавлена строка VERSION\_INTEGER, содержащая версию ClickHouse в числовом представлении, вида 18016000. #3644

- Удалена возможность сравнения типа `Date` с числом, чтобы избежать потенциальных ошибок вида `date = 2018-12-17`, где ошибочно не указаны кавычки вокруг даты. [#3687](#)
- Исправлено поведение функций с состоянием типа `rowNumberInAllBlocks` - раньше они выдавали число на единицу больше вследствие их запуска во время анализа запроса. [Amos Bird](#)
- При невозможности удалить файл `force_restore_data`, выводится сообщение об ошибке. [Amos Bird](#)

## :Улучшение сборки

- Обновлена библиотека `jemalloc`, что исправляет потенциальную утечку памяти. [Amos Bird](#)
- Для debug сборок включено по-умолчанию профилирование `jemalloc`. [2cc82f5c](#)
- Добавлена возможность запуска интеграционных тестов, при наличии установленным в системе лишь `Docker`. [#3650](#)
- Добавлен fuzz тест выражений в SELECT запросах. [#3442](#)
- Добавлен покоммитный стресс-тест, выполняющий функциональные тесты параллельно и в произвольном порядке, позволяющий обнаружить больше race conditions. [#3438](#)
- Улучшение способа запуска clickhouse-server в Docker образе. [Elghazal Ahmed](#)
- Для Docker образа добавлена поддержка инициализации базы данных с помощью файлов в директории `/docker-entrypoint-initdb.d`. [Konstantin Lebedev](#)
- Исправления для сборки под ARM. [#3709](#)

## :Обратно несовместимые изменения

- Удалена возможность сравнения типа `Date` с числом, необходимо вместо `toDate('2018-12-18') = 17883`, использовать явное приведение типов `= toDate(17883)` [#3687](#)

## ClickHouse release 18.14.19, 2018-12-19

### :Исправления ошибок

- Исправлена проблема, приводившая к невозможности обновить словари с источником ODBC. [#3825](#), [#3829](#)
- Исправлен segfault в случае превышения ограничения `max_temporary_non_const_columns`. [#3788](#)
- Корректное указание базы данных при выполнении DDL запросов `ON CLUSTER`. [#3460](#)

### :Улучшения сборки

- .Исправления сборки под ARM

## ClickHouse release 18.14.18, 2018-12-04

### :Исправления ошибок

- Исправлена ошибка в функции `dictGet...` для словарей типа `range`, если один из аргументов константный, а другой - нет. [#3751](#)
- Исправлена ошибка, приводящая к выводу сообщений `netlink: '...': attribute type 1 has an invalid length` в логе ядра Linux, проявляющаяся на достаточно новых ядрах Linux. [#3749](#)
- Исправлен segfault при выполнении функции `empty` от аргумента типа `FixedString`. [Daniel, Dao Quang Minh](#)
- Исправлена избыточная аллокация памяти при большом значении настройки `max_query_size` (кусочек памяти размера `max_query_size` выделялся сразу). [#3720](#)

### :Улучшения процесса сборки ClickHouse

- Исправлена сборка с использованием библиотек LLVM/Clang версии 7 из пакетов ОС (эти библиотеки используются для динамической компиляции запросов). [#3582](#)

## ClickHouse release 18.14.17, 2018-11-30

### :Исправления ошибок

- Исправлена ситуация, при которой ODBC Bridge продолжал работу после завершения работы сервера ClickHouse. Теперь ODBC Bridge всегда завершает работу вместе с сервером. [#3642](#)

- Исправлена синхронная вставка в **Distributed** таблицу в случае явного указания неполного списка столбцов или списка столбцов в измененном порядке. **#3673**
- Исправлен редкий race condition, который мог привести к падению сервера при удалении MergeTree-таблиц. **#3680**
- Исправлен deadlock при выполнении запроса, возникающий если создание новых потоков выполнения невозможно из-за ошибки **Resource temporarily unavailable**. **#3643**
- Исправлена ошибка парсинга **ENGINE** при создании таблицы с синтаксисом **AS table** в случае, когда **AS table** указывался после **ENGINE**, что приводило к игнорированию указанного движка. **#3692**

## ClickHouse release 18.14.15, 2018-11-21

### :Исправления ошибок

- При чтении столбцов типа **Array(String)**, размер требуемого куска памяти оценивался слишком большим, что приводило к исключению "Memory limit exceeded" при выполнении запроса. Ошибка появилась в версии 18.12.13. **#3589**

## ClickHouse release 18.14.14, 2018-11-20

### :Исправления ошибок

- Исправлена работа запросов **ON CLUSTER** в случае, когда в конфигурации кластера включено шифрование (флаг **<secure>**). **#3599**

### :Улучшения процесса сборки ClickHouse

- Исправлены проблемы сборки (Illum-7 из системы, macOS) **#3582**

## ClickHouse release 18.14.13, 2018-11-08

### :Исправления ошибок

- Исправлена ошибка **Block structure mismatch in MergingSorted stream**. **#3162**
- Исправлена работа запросов **ON CLUSTER** в случае, когда в конфигурации кластера включено шифрование (флаг **<secure>**). **#3465**
- Исправлена ошибка при использовании **SAMPLE**, **PREWHERE** и столбцов-алиасов. **#3543**
- Исправлена редкая ошибка **unknown compression method** при использовании настройки **min\_bytes\_to\_use\_direct\_io**. **3544**

### :Улучшения производительности

- Исправлена деградация производительности запросов с **GROUP BY** столбцов типа Int16, Date на процессорах AMD EPYC. **Игорь Лапко**
- Исправлена деградация производительности при обработке длинных строк. **#3530**

### :Улучшения процесса сборки ClickHouse

- Доработки для упрощения сборки в Arcadia. **#3475**, **#3535**

## ClickHouse release 18.14.12, 2018-11-02

### :Исправления ошибок

- Исправлена ошибка при join-запросе двух неименованных подзапросов. **#3505**
- Исправлена генерация пустой **WHERE**-части при запросах к внешним базам. **hotid**
- Исправлена ошибка использования неправильной настройки таймаута в ODBC-словарях. **Marek Vavruša**

## ClickHouse release 18.14.11, 2018-10-29

### :Исправления ошибок

- Исправлена ошибка **Block structure mismatch in UNION stream: different number of columns** в запросах с **LIMIT**. **#2156**



Исправлены ошибки при слиянии данных в таблицах, содержащих массивы внутри Nested структур. [#3397](#)

Исправлен неправильный результат запросов при выключенной настройке

`merge_tree_uniform_read_distribution` (включена по умолчанию). [#3429](#)

Исправлена ошибка при вставке в Distributed таблицу в формате Native. [#3411](#)

## ClickHouse release 18.14.10, 2018-10-23

Настройка `compile_expressions` (JIT компиляция выражений) выключена по умолчанию. [#3410](#)

.Настройка `enable_optimize_predicate_expression` выключена по умолчанию

## ClickHouse release 18.14.9, 2018-10-16

### :Новые возможности

Модификатор `WITH CUBE` для `GROUP BY` (также доступен синтаксис: `GROUP BY CUBE(...)`). [#3172](#)

Добавлена функция `formatDateTime`. [Alexandr Krasheninnikov](#)

Добавлен движок таблиц `JDBC` и табличная функция `jdbc` (для работы требуется установка `clickhouse-jdbc-bridge`). [Alexandr Krasheninnikov](#)

Добавлены функции для работы с ISO номером недели: `toISOWeek`, `toISOYear`, `toStartOfISOYear`, а также `toDayOfYear`. [#3146](#)

Добавлена возможность использования столбцов типа `Nullable` для таблиц типа `MySQL`, `ODBC`. [#3362](#)

Возможность чтения вложенных структур данных как вложенных объектов в формате `JSONEachRow`.

Добавлена настройка `input_format_import_nested_json`. [Veloman Yunkan](#)

Возможность параллельной обработки многих `MATERIALIZED VIEW` при вставке данных. Настройка `parallel_view_processing`. [Marek Vavruša](#)

Добавлен запрос `SYSTEM FLUSH LOGS` (форсированный сброс логов в системные таблицы, такие как например, `query_log`) [#3321](#)

Возможность использования предопределённых макросов `database` и `table` в объявлении `Replicated` таблиц. [#3251](#)

Добавлена возможность чтения значения типа `Decimal` в инженерной нотации (с указанием десятичной экспоненты). [#3153](#)

### :Экспериментальные возможности

Оптимизация `GROUP BY` для типов данных `LowCardinality` [#3138](#)

Оптимизации вычисления выражений для типов данных `LowCardinality` [#3200](#)

### :Улучшения

Существенно уменьшено потребление памяти для запросов с `ORDER BY` и `LIMIT`. Настройка `max_bytes_before_remerge_sort`. [#3205](#)

При отсутствии указания типа `JOIN` (`LEFT`, `INNER`, ...), подразумевается `INNER JOIN`. [#3147](#)

Корректная работа квалифицированных звёздочек в запросах с `JOIN`. [Winter Zhang](#)

Движок таблиц `ODBC` корректно выбирает способ кватирования идентификаторов в SQL диалекте удалённой СУБД. [Alexandr Krasheninnikov](#)

.Настройка `compile_expressions` (JIT компиляция выражений) включена по-умолчанию

Исправлено поведение при одновременном `DROP DATABASE/TABLE IF EXISTS` и `CREATE DATABASE/TABLE IF NOT EXISTS`. Ранее запрос `CREATE DATABASE ... IF NOT EXISTS` мог выдавать сообщение об ошибке вида "File ... already exists", а запросы `CREATE TABLE ... IF NOT EXISTS` и `DROP TABLE IF EXISTS` могли выдавать сообщение `Table ... is creating or attaching right now` [#3101](#)

Выражения `LIKE` и `IN` с константной правой частью пробрасываются на удалённый сервер при запросах из таблиц типа `MySQL` и `ODBC`. [#3182](#)

Сравнения с константными выражениями в секции `WHERE` пробрасываются на удалённый сервер при запросах из таблиц типа `MySQL` и `ODBC`. Ранее пробрасывались только сравнения с константами.

[#3182](#)

Корректное вычисление ширины строк в терминале для `Pretty` форматов, в том числе для строк с .иероглифами. [Amos Bird](#)

.Возможность указания `ON CLUSTER` для запросов `ALTER UPDATE`

- Увеличена производительность чтения данных в формате `JSONEachRow`. [#3332](#)
- Добавлены синонимы функций `LENGTH`, `CHARACTER_LENGTH` для совместимости. Функция `CONCAT` стала регистронезависимой. [#3306](#)
- Добавлен синоним `TIMESTAMP` для типа `DateTime`. [#3390](#)
- В логах сервера всегда присутствует место для `query_id`, даже если строка лога не относится к запросу. Это сделано для более простого парсинга текстовых логов сервера сторонними инструментами
- Логгирование потребления памяти запросом при превышении очередной отметки целого числа гигабайт. [#3205](#)
- Добавлен режим совместимости для случая, когда клиентская библиотека, работающая по Native протоколу, по ошибке отправляет меньшее количество столбцов, чем сервер ожидает для запроса `INSERT`. Такой сценарий был возможен при использовании библиотеки `clickhouse-cpp`. Ранее этот сценарий приводил к падению сервера. [#3171](#)
- В `clickhouse-copier`, в задаваемом пользователем выражении `WHERE`, появилась возможность использовать алиас `partition_key` (для дополнительной фильтрации по партициям исходной таблицы). Это полезно, если схема партиционирования изменяется при копировании, но изменяется незначительно. [#3166](#)
- Рабочий поток движка `Kafka` перенесён в фоновый пул потоков для того, чтобы автоматически уменьшать скорость чтения данных при большой нагрузке. [Marek Vavruša](#)
- Поддержка чтения значений типа `Tuple` и `Nested` структур как `struct` в формате `Cap'n'Proto` [Marek Vavruša](#)
- В список доменов верхнего уровня для функции `firstSignificantSubdomain` добавлен домен `biz` [decaseal](#)
- В конфигурации внешних словарей, пустое значение `null_value` интерпретируется, как значение типа данных по-умолчанию. [#3330](#)
- Поддержка функций `intDiv`, `intDivOrZero` для `Decimal`. [b48402e8](#)
- Поддержка типов `Date`, `DateTime`, `UUID`, `Decimal` в качестве ключа для агрегатной функции `sumMap`. [#3281](#)
- Поддержка типа данных `Decimal` во внешних словарях. [#3324](#)
- Поддержка типа данных `Decimal` в таблицах типа `SummingMergeTree`. [#3348](#)
- Добавлена специализация для `UUID` в функции `if`. [#3366](#)
- Уменьшено количество системных вызовов `open`, `close` при чтении из таблиц семейства `MergeTree` [#3283](#)
- Возможность выполнения запроса `TRUNCATE TABLE` на любой реплике (запрос пробрасывается на реплику-лидера). [Kirill Shvakov](#)

## :Исправление ошибок

- Исправлена ошибка в работе таблиц типа `Dictionary` для словарей типа `range_hashed`. Ошибка возникла в версии 18.12.17. [#1702](#)
- Исправлена ошибка при загрузке словарей типа `range_hashed` (сообщение `Unsupported type Nullable(...)`). Ошибка возникла в версии 18.12.17. [#3362](#)
- Исправлена некорректная работа функции `pointInPolygon` из-за накопления погрешности при вычислениях для полигонов с большим количеством близко расположенных вершин. [#3331](#) [#3341](#)
- Если после слияния кусков данных, у результирующего куска чексумма отличается от результата того же слияния на другой реплике, то результат слияния удаляется, и вместо этого кусок скачивается с другой реплики (это правильное поведение). Но после скачивания куска, он не мог добавиться в рабочий набор из-за ошибки, что кусок уже существует (так как кусок после слияния удалялся не сразу, а с задержкой). Это приводило к циклическим попыткам скачивания одних и тех же данных. [#3194](#)
- Исправлен некорректный учёт общего потребления оперативной памяти запросами (что приводило к неправильной работе настройки `max_memory_usage_for_all_queries` и неправильному значению метрики `MemoryTracking`). Ошибка возникла в версии 18.12.13. [Marek Vavruša](#)
- Исправлена работоспособность запросов `CREATE TABLE ... ON CLUSTER ... AS SELECT ...` Ошибка возникла в версии 18.12.13. [#3247](#)
- Исправлена лишняя подготовка структуры данных для `JOIN` на сервере-инициаторе запроса, если `JOIN` выполняется только на удалённых серверах. [#3340](#)
- Исправлены ошибки в движке `Kafka`: неработоспособность после исключения при начале чтения данных; блокировка при завершении [Marek Vavruša](#)
- Для таблиц `Kafka` не передавался опциональный параметр `schema` (схема формата `Cap'n'Proto`). [Vojtech Splichal](#)

Если ансамбль серверов ZooKeeper содержит серверы, которые принимают соединение, но сразу же разрывают его вместо ответа на рукопожатие, то ClickHouse выбирает для соединения другой сервер. Ранее в этом случае возникала ошибка `Cannot read all data. Bytes read: 0. Bytes expected: 4.` и сервер не мог стартовать. [8218cf3a](#)

Если ансамбль серверов ZooKeeper содержит серверы, для которых DNS запрос возвращает ошибку, то такие серверы пропускаются. [17b8e209](#)

Исправлено преобразование типов между `Date` и `DateTime` при вставке данных в формате `VALUES` (в случае, когда `input_format_values_interpret_expressions = 1`). Ранее преобразование производилось между числовым значением количества дней с начала unix эпохи и unix timestamp, что приводило к неожиданным результатам. [#3229](#)

Исправление преобразования типов между `Decimal` и целыми числами. [#3211](#)

Исправлены ошибки в работе настройки `enable_optimize_predicate_expression`. [Winter Zhang](#)

Исправлена ошибка парсинга формата CSV с числами с плавающей запятой, если используется разделитель CSV не по-умолчанию, такой как например, `;` [#3155](#)

Исправлена функция `arrayCumSumNonNegative` (она не накапливает отрицательные значения, если аккумулятор становится меньше нуля). [Aleksey Studnev](#)

Исправлена работа `Merge` таблицы поверх `Distributed` таблиц при использовании `PREWHERE`. [#3165](#)

Исправления ошибок в запросе `ALTER UPDATE`

Исправления ошибок в табличной функции `odbc`, которые возникли в версии 18.12. [#3197](#)

Исправлена работа агрегатных функций с комбинаторами `StateArray`. [#3188](#)

Исправлено падение при делении значения типа `Decimal` на ноль. [69dd6609](#)

Исправлен вывод типов для операций с использованием аргументов типа `Decimal` и целых чисел. [#3224](#)

Исправлен `segfault` при `GROUP BY` по `Decimal128`. [3359ba06](#)

Настройка `log_query_threads` (логгирование информации о каждом потоке исполнения запроса) теперь имеет эффект только если настройка `log_queries` (логгирование информации о запросах) выставлена в 1. Так как настройка `log_query_threads` включена по-умолчанию, ранее информация о потоках логгировалась даже если логгирование запросов выключено. [#3241](#)

Исправлена ошибка в распределённой работе агрегатной функции `quantiles` (сообщение об ошибке вида `Not found column quantile...`). [292a8855](#)

Исправлена проблема совместимости при одновременной работе на кластере серверов версии 18.12.17 и более старых, приводящая к тому, что при распределённых запросах с `GROUP BY` по ключам одновременно фиксированной и не фиксированной длины, при условии, что количество данных в процессе агрегации большое, могли возвращаться не до конца агрегированные данные (одни и те же ключи агрегации в двух разных строках). [#3254](#)

Исправлена обработка подстановок в `clickhouse-performance-test`, если запрос содержит только часть из объявленных в тесте подстановок. [#3263](#)

Исправлена ошибка при использовании `FINAL` совместно с `PREWHERE`. [#3298](#)

Исправлена ошибка при использовании `PREWHERE` над столбцами, добавленными при `ALTER`. [#3298](#)

Добавлена проверка отсутствия `arrayJoin` для `DEFAULT`, `MATERIALIZED` выражений. Ранее наличие `arrayJoin` приводило к ошибке при вставке данных. [#3337](#)

Добавлена проверка отсутствия `arrayJoin` в секции `PREWHERE`. Ранее это приводило к сообщениям вида `Size ... doesn't match` или `Unknown compression method` при выполнении запросов. [#3357](#)

Исправлен `segfault`, который мог возникать в редких случаях после оптимизации - замены цепочек `.AND` из равенства выражения константам на соответствующее выражение `IN`. [liuyimin-bytedance](#)

Мелкие исправления `clickhouse-benchmark`: ранее информация о клиенте не передавалась на сервер; более корректный подсчёт числа выполненных запросов при завершении работы и для ограничения числа итераций. [#3351](#) [#3352](#)

## :Обратно несовместимые изменения

Удалена настройка `allow_experimental_decimal_type`. Тип данных `Decimal` доступен для использования по-умолчанию. [#3329](#)

## ClickHouse release 18.12.17, 2018-09-16

### :Новые возможности

`invalidate_query` (возможность задать запрос для проверки необходимости обновления внешнего словаря) реализована для источника `clickhouse`. [#3126](#)

Добавлена возможность использования типов данных `UInt*`, `Int*`, `DateTime` (наравне с типом `Date`) в качестве ключа внешнего словаря типа `range_hashed`, определяющего границы диапазонов.

Возможность использования `NULL` в качестве обозначения открытого диапазона. [Vasily Nemkov](#)

Для типа `Decimal` добавлена поддержка агрегатных функций `var*`, `stddev*`. [#3129](#)

Для типа `Decimal` добавлена поддержка математических функций (`exp`, `sin` и т. п.) [#3129](#)

В таблицу `system.part_log` добавлен столбец `partition_id`. [#3089](#)

## :Исправление ошибок

Исправлена работа `Merge` таблицы поверх `Distributed` таблиц. [Winter Zhang](#)

Исправлена несовместимость (лишняя зависимость от версии `glibc`), приводящая к невозможности запуска ClickHouse на `Ubuntu Precise` и более старых. Несовместимость возникла в версии 18.12.13.

[#3130](#)

Исправлены ошибки в работе настройки `enable_optimize_predicate_expression`. [Winter Zhang](#)

Исправлено незначительное нарушение обратной совместимости, проявляющееся при одновременной работе на кластере реплик версий до 18.12.13 и создании новой реплики таблицы на сервере более новой версии (выдаётся сообщение `Can not clone replica, because the ... updated to new ClickHouse version`, что полностью логично, но не должно было происходить). [#3122](#)

## :Обратно несовместимые изменения

Настройка `enable_optimize_predicate_expression` включена по-умолчанию, что конечно очень оптимистично.

При возникновении ошибок анализа запроса, связанных с поиском имён столбцов, следует выставить `enable_optimize_predicate_expression` в 0. [Winter Zhang](#)

## ClickHouse release 18.12.14, 2018-09-13

### :Новые возможности

Добавлена поддержка запросов `ALTER UPDATE`. [#3035](#)

Добавлена настройка `allow_ddl`, управляющая доступом пользователя к DDL-запросам. [#3104](#)

Добавлена настройка `min_merge_bytes_to_use_direct_io` для движков семейства `MergeTree`, позволяющая задать порог на суммарный размер слияния после которого работа с файлами кусков будет происходить с `O_DIRECT`. [#3117](#)

В системную таблицу `system.merges` добавлен столбец `partition_id`. [#3099](#)

### Улучшения

Если в процессе мутации кусок остался неизменённым, он не будет скачан репликами. [#3103](#)

При работе с `clickhouse-client` добавлено автодополнение для имён настроек. [#3106](#)

## Исправление ошибок

Добавлена проверка размеров массивов, которые являются элементами полей типа `Nested`, при вставке. [#3118](#)

Исправлена ошибка обновления внешних словарей с источником `ODBC` и форматом хранения `hashed`. Ошибка возникла в версии 18.12.13

Исправлено падение при создании временной таблицы из запроса с условием `IN`. [Winter Zhang](#)

Исправлена ошибка в работе агрегатных функций для массивов, элементами которых может быть `NULL`. [Winter Zhang](#)

## ClickHouse release 18.12.13, 2018-09-10

### :Новые возможности

Добавлен тип данных `DECIMAL(digits, scale)` (`Decimal32(scale)`, `Decimal64(scale)`, `Decimal128(scale)`). Возможность доступна под настройкой `allow_experimental_decimal_type`. [#2846](#) [#2970](#) [#3008](#) [#3047](#)

Модификатор `WITH ROLLUP` для `GROUP BY` (также доступен синтаксис: `GROUP BY ROLLUP(...)`). [#2948](#)

- В запросах с JOIN, звёздочка раскрывается в список столбцов всех таблиц, в соответствии со стандартом SQL. Вернуть старое поведение можно, выставив настройку (уровня пользователя) `asterisk_left_columns_only` в значение 1. [Winter Zhang](#)
- Добавлена поддержка JOIN с табличной функцией. [Winter Zhang](#)
- Автодополнение по нажатию Tab в clickhouse-client. [Sergey Shcherbin](#)
- Нажатие Ctrl+C в clickhouse-client очищает запрос, если он был введен. [#2877](#)
- Добавлена настройка `join_default_strictness` (значения `'any'`, `'all'`). Её использование позволяет не указывать `ANY` или `ALL` для JOIN. [#2982](#)
- В каждой строчке лога сервера, относящейся к обработке запроса, выводится идентификатор запроса. [#2482](#)
- Возможность получения логов выполнения запроса в clickhouse-client (настройка `send_logs_level`). При распределённой обработке запроса, логи отправляются каскадно со всех серверов. [#2482](#)
- В таблицах `system.query_log` и `system.processes` (`SHOW PROCESSLIST`) появилась информация о всех изменённых настройках при выполнении запроса (вложенная структура данных `Settings`). Добавлена настройка `log_query_settings`. [#2482](#)
- В таблицах `system.query_log` и `system.processes` появилась информация о номерах потоков, участвующих в исполнении запроса (столбец `thread_numbers`). [#2482](#)
- Добавлены счётчики `ProfileEvents`, измеряющие время, потраченное на чтение и запись по сети; чтение и запись на диск; количество сетевых ошибок; время потраченное на ожидании при ограничении сетевой полосы. [#2482](#)
- Добавлены счётчики `ProfileEvents`, содержащие системные метрики из `rusage` (позволяющие получить информацию об использовании CPU в `userspace` и ядре, `page faults`, `context switches`) а также метрики `taskstats` (позволяющие получить информацию о времени ожидания IO, CPU, а также количество прочитанных и записанных данных с учётом и без учёта `page cache`). [#2482](#)
- Счётчики `ProfileEvents` учитываются не только глобально, но и на каждый запрос, а также на каждый поток выполнения запроса, что позволяет детально профилировать потребление ресурсов отдельными запросами. [#2482](#)
- Добавлена таблица `system.query_thread_log`, содержащая информацию о каждом потоке выполнения запроса. Добавлена настройка `log_query_threads`. [#2482](#)
- В таблицах `system.metrics` и `system.events` появилась встроенная документация. [#3016](#)
- Добавлена функция `arrayEnumerateDense`. [Amos Bird](#)
- Добавлены функции `arrayCumSumNonNegative` и `arrayDifference`. [Aleksey Studnev](#)
- Добавлена агрегатная функция `retention`. [Sundy Li](#)
- Возможность сложения (слияния) состояний агрегатных функций с помощью оператора плюс, а также умножения состояний агрегатных функций на целую неотрицательную константу. [#3062](#) [#3034](#)
- В таблицах семейства MergeTree добавлен виртуальный столбец `_partition_id`. [#3089](#)

## :Экспериментальные возможности

- Добавлен тип данных `LowCardinality(T)`. Тип данных автоматически создаёт локальный словарь значений и позволяет обрабатывать данные без распаковки словаря. [#2830](#)
- Добавлен кэш JIT-скомпилированных функций, а также счётчик числа использований перед компиляцией. Возможность JIT-компиляции выражений включается настройкой `compile_expressions`. [#2990](#) [#3077](#)

## :Улучшения

- Исправлена проблема неограниченного накопления лога репликации в случае наличия заброшенных реплик. Добавлен режим эффективного восстановления реплик после длительного отставания
- Увеличена производительность при выполнении `GROUP BY` в случае, если есть несколько полей агрегации, одно из которых строковое, а другие - фиксированной длины
- Увеличена производительность при использовании `PREWHERE` и при неявном переносе выражений в `.PREWHERE`
- Увеличена производительность парсинга текстовых форматов (`CSV`, `TSV`). [Amos Bird](#) [#2980](#)
- Увеличена производительность чтения строк и массивов в бинарных форматах. [Amos Bird](#)
- Увеличена производительность и уменьшено потребление памяти в запросах к таблицам `system.tables` и `system.columns` в случае наличия очень большого количества таблиц на одном сервере. [#2953](#)



- Исправлена проблема низкой производительности в случае наличия большого потока запросов, для которых возвращается ошибка (в `perf top` видна функция `_dl_addr`, при этом сервер использует мало CPU). [#2938](#)
- Прокидывание условий внутрь View (при включенной настройке `enable_optimize_predicate_expression`) [Winter Zhang](#)
- Доработки недостающей функциональности для типа данных `UUID`. [#3074](#) [#2985](#)
- Тип данных `UUID` поддержан в словарях The-Alchemist. [#2822](#)
- Функция `visitParamExtractRaw` корректно работает с вложенными структурами. [Winter Zhang](#)
- При использовании настройки `input_format_skip_unknown_fields` корректно работает пропуск значений-объектов в формате `JSONEachRow`. [BlahGeek](#)
- Для выражения `CASE` с условиями, появилась возможность не указывать `ELSE`, что эквивалентно `ELSE NULL`. [#2920](#)
- Возможность конфигурирования operation timeout при работе с ZooKeeper. [urykhy](#)
- Возможность указания смещения для `LIMIT n, m` в виде `LIMIT n OFFSET m`. [#2840](#)
- Возможность использования синтаксиса `SELECT TOP n` в качестве альтернативы для `LIMIT`. [#2840](#)
- Увеличен размер очереди записи в системные таблицы, что позволяет уменьшить количество ситуаций `SystemLog queue is full`
- В агрегатной функции `windowFunnel` добавлена поддержка событий, подходящих под несколько условий. [Amos Bird](#)
- Возможность использования дублирующихся столбцов в секции `USING` для `JOIN`. [#3006](#)
- Для форматов `Pretty` введено ограничение выравнивания столбцов по ширине. Настройка `output_format_pretty_max_column_pad_width`. В случае более широкого значения, оно всё ещё будет выведено целиком, но остальные ячейки таблицы не будут излишне широкими. [#3003](#)
- В табличной функции `odbc` добавлена возможность указания имени базы данных/схемы. [Amos Bird](#)
- Добавлена возможность использования имени пользователя, заданного в конфигурационном файле `clickhouse-client`. [Vladimir Kozbin](#)
- Счётчик `ZooKeeperExceptions` разделён на три счётчика `ZooKeeperUserExceptions`, `ZooKeeperHardwareExceptions`, `ZooKeeperOtherExceptions`
- Запросы `ALTER DELETE` работают для материализованных представлений
- Добавлена рандомизация во времени периодического запуска `cleanup thread` для таблиц типа `ReplicatedMergeTree`, чтобы избежать периодических всплесков нагрузки в случае очень большого количества таблиц типа `ReplicatedMergeTree`
- Поддержка запроса `ATTACH TABLE ... ON CLUSTER`. [#3025](#)

## :Исправление ошибок

- Исправлена ошибка в работе таблиц типа `Dictionary` (кидается исключение `Size of offsets doesn't match size of column` или `Unknown compression method`). Ошибка появилась в версии 18.10.3. [#2913](#)
- Исправлена ошибка при мере данных таблиц типа `CollapsingMergeTree`, если один из кусков данных пустой (такие куски, в свою очередь, образуются при слиянии или при `ALTER DELETE` в случае удаления всех данных), и для слияния был выбран алгоритм `vertical`. [#3049](#)
- Исправлен race condition при `DROP` или `TRUNCATE` таблиц типа `Memory` при одновременном `SELECT`, который мог приводить к падениям сервера. Ошибка появилась в версии 1.1.54388. [#3038](#)
- Исправлена возможность потери данных при вставке в `Replicated` таблицы в случае получения ошибки `Session expired` (потеря данных может быть обнаружена по метрике `ReplicatedDataLoss`). Ошибка возникла в версии 1.1.54378. [#2939](#) [#2949](#) [#2964](#)
- Исправлен segfault при `JOIN ... ON`. [#3000](#)
- Исправлена ошибка поиска имён столбцов в случае, если выражение `WHERE` состоит целиком из квалифицированного имени столбца, как например `WHERE table.column`. [#2994](#)
- Исправлена ошибка вида "Not found column" при выполнении распределённых запросов в случае, если с удалённого сервера запрашивается единственный столбец, представляющий собой выражение `IN` с подзапросом. [#3087](#)
- Исправлена ошибка `Block structure mismatch in UNION stream: different number of columns`, возникающая при распределённых запросах, если один из шардов локальный, а другой - нет, и если при этом срабатывает оптимизация переноса в `PREWHERE`. [#2226](#) [#3037](#) [#3055](#) [#3065](#) [#3073](#) [#3090](#) [#3093](#)
- Исправлена работа функции `pointInPolygon` для некоторого случая невыпуклых полигонов. [#2910](#)
- Исправлен некорректный результат при сравнении `nan` с целыми числами. [#3024](#)

- Исправлена ошибка в библиотеке [zlib-ng](#), которая могла приводить к segfault в редких случаях. [#2854](#)
- Исправлена утечка памяти при вставке в таблицу со столбцами типа [AggregateFunction](#), если состояние агрегатной функции нетривиальное (выделяет память отдельно), и если в одном запросе на вставку получается несколько маленьких блоков. [#3084](#)
- Исправлен race condition при одновременном создании и удалении одной и той же таблицы типа [Buffer](#) или [MergeTree](#)
- Исправлена возможность segfault при сравнении кортежей из некоторых нетривиальных типов, таких как, например, кортежей. [#2989](#)
- Исправлена возможность segfault при выполнении некоторых запросов [ON CLUSTER](#). [Winter Zhang](#)
- Исправлена ошибка в функции [arrayDistinct](#) в случае [Nullable](#) элементов массивов. [#2845](#) [#2937](#)
- Возможность [enable\\_optimize\\_predicate\\_expression](#) корректно поддерживает случаи с [SELECT \\*](#). [Winter Zhang](#)
- Исправлена возможность segfault при переинициализации сессии с ZooKeeper. [#2917](#)
- .Исправлена возможность блокировки при взаимодействии с ZooKeeper
- .Исправлен некорректный код суммирования вложенных структур данных в [SummingMergeTree](#)
- При выделении памяти для состояний агрегатных функций, корректно учитывается выравнивание, что позволяет использовать при реализации состояний агрегатных функций операции, для которых выравнивание является необходимым. [chenxing-xc](#)

## :Исправления безопасности

- Безопасная работа с ODBC источниками данных. Взаимодействие с ODBC драйверами выполняется через отдельный процесс [clickhouse-odbc-bridge](#). Ошибки в сторонних ODBC драйверах теперь не приводят к проблемам со стабильностью сервера или уязвимостям. [#2828](#) [#2879](#) [#2886](#) [#2893](#) [#2921](#)
- Исправлена некорректная валидация пути к файлу в табличной функции [catBoostPool](#). [#2894](#)
- Содержимое системных таблиц ([tables](#), [databases](#), [parts](#), [columns](#), [parts\\_columns](#), [merges](#), [mutations](#), [replicas](#), [replication\\_queue](#)) фильтруется согласно конфигурации доступа к базам данных для пользователя ([allow\\_databases](#)) [Winter Zhang](#)

## :Обратно несовместимые изменения

- В запросах с JOIN, звёздочка раскрывается в список столбцов всех таблиц, в соответствии со стандартом SQL. Вернуть старое поведение можно, выставив настройку (уровня пользователя) [.asterisk\\_left\\_columns\\_only](#) в значение 1

## :Изменения сборки

- .Добавлен покоммитный запуск большинства интеграционных тестов
- .Добавлен покоммитный запуск проверки стиля кода
- Корректный выбор реализации [memcpy](#) при сборке на CentOS7 / Fedora. [Etienne Champetier](#)
- При сборке с помощью clang добавлены некоторые warnings из [-Weverything](#) в дополнение к обычным [-Wall](#) [-Wextra](#) [-Werror](#). [#2957](#)
- .При debug сборке используется debug вариант [jemalloc](#)
- Абстрагирован интерфейс библиотеки для взаимодействия с ZooKeeper. [#2950](#)

## ClickHouse release 18.10.3, 2018-08-13

### :Новые возможности

- Возможность использования HTTPS для репликации. [#2760](#)
- Добавлены функции [murmurHash2\\_64](#), [murmurHash3\\_32](#), [murmurHash3\\_64](#), [murmurHash3\\_128](#) в дополнение к имеющемуся [murmurHash2\\_32](#). [#2791](#)
- Поддержка Nullable типов в ODBC драйвере ClickHouse (формат вывода [ODBCDriver2](#)) [#2834](#)
- .Поддержка [UUID](#) в ключевых столбцах

### :Улучшения

- Удаление кластеров без перезагрузки сервера при их удалении из конфигурационных файлов. [#2777](#)
- Удаление внешних словарей без перезагрузки сервера при их удалении из конфигурационных файлов. [#2779](#)
- Добавлена поддержка [SETTINGS](#) для движка таблиц [Kafka](#). [Alexander Marshalov](#)

- Доработки для типа данных `UUID` (не полностью) Šimon Podlipský. [#2618](#)
- Поддержка пустых кусков после мержей в движках `SummingMergeTree`, `CollapsingMergeTree` and `VersionedCollapsingMergeTree`. [#2815](#)
- Удаление старых записей о полностью выполнившихся мутациях (`ALTER DELETE`) [#2784](#)
- Добавлена таблица `system.merge_tree_settings`. Kirill Shvakov
- В таблицу `system.tables` добавлены столбцы зависимостей: `dependencies_database` и `dependencies_table`. Winter Zhang
- Добавлена опция конфига `max_partition_size_to_drop`. [#2782](#)
- Добавлена настройка `output_format_json_escape_forward_slashes`. Alexander Bocharov
- Добавлена настройка `max_fetch_partition_retries_count`. [#2831](#)
- Добавлена настройка `prefer_localhost_replica`, позволяющая отключить предпочтение локальной реплики и хождение на локальную реплику без межпроцессного взаимодействия. [#2832](#)
- Агрегатная функция `quantileExact` возвращает `nan` в случае агрегации по пустому множеству `Float32/Float64` типов. Sundy Li

## :Исправление ошибок

- Убрано излишнее экранирование параметров connection string для ODBC, котрое приводило к невозможности соединения. Ошибка возникла в версии 18.6.0
- Исправлена логика обработки команд на `REPLACE PARTITION` в очереди репликации. Неправильная логика могла приводить к тому, что при наличии двух `REPLACE` одной и той же партиции, один из них оставался в очереди репликации и не мог выполниться. [#2814](#)
- Исправлена ошибка при мерже, если все куски были пустыми (такие куски, в свою очередь, образуются при слиянии или при `ALTER DELETE` в случае удаления всех данных). Ошибка появилась в версии 18.1.0. [#2930](#)
- Исправлена ошибка при параллельной записи в таблицы типа `Set` или `Join`. Amos Bird
- Исправлена ошибка `Block structure mismatch in UNION stream: different number of columns`, возникающая при запросах с `UNION ALL` внутри подзапроса, в случае, если один из `SELECT` запросов содержит дублирующиеся имена столбцов. Winter Zhang
- .Исправлена утечка памяти в случае исключения при соединении с MySQL сервером
- Исправлен некорректный код возврата clickhouse-client в случае ошибочного запроса
- Исправлен некорректная работа materialized views, содержащих `DISTINCT`. [#2795](#)

## Обратно несовместимые изменения

- .Убрана поддержка запросов `CHECK TABLE` для Distributed таблиц

## :Изменения сборки

- Заменен аллокатор, теперь используется `jemalloc` вместо `tcmalloc`. На некоторых сценариях ускорение достигает 20%. В то же время, существуют запросы, замедлившиеся до 20%. Потребление памяти на некоторых сценариях примерно на 10% меньше и более стабильно. При высококонкурентной нагрузке, потребление CPU в userspace и в system незначительно вырастает. [#2773](#)
- Использование `libressl` из submodule. [#1983](#) [#2807](#)
- Использование `unixodbc` из submodule. [#2789](#)
- Использование `mariadb-connector-c` из submodule. [#2785](#)
- В репозиторий добавлены файлы функциональных тестов, рассчитывающих на наличие тестовых данных (пока без самих тестовых данных)

## ClickHouse release 18.6.0, 2018-08-02

### :Новые возможности

- :Добавлена поддержка ON выражений для JOIN ON синтаксиса  
`[... (... ,JOIN ON Expr([table].column, ...) = Expr([table].column, ...) [AND Expr([table].column, ...) = Expr([table].column`  
 Выражение должно представлять из себя цепочку равенств, объединенных оператором AND. Каждая часть равенства может являться произвольным выражением над столбцами одной из таблиц.  
 Поддержана возможность использования fully qualified имен столбцов (`table.name`, `database.table.name`, `table_alias.name`, `subquery_alias.name`) для правой таблицы. [#2742](#)



Добавлена возможность включить HTTPS для репликации. [#2760](#)

## :Улучшения

Сервер передаёт на клиент также patch-компонент своей версии. Данные о patch компоненте версии добавлены в `system.processes` и `query_log`. [#2646](#)

## ClickHouse release 18.5.1, 2018-07-31

### :Новые возможности

.Добавлена функция хеширования `murmurHash2_32`. [#2756](#)

## :Улучшения

Добавлена возможность указывать значения в конфигурационных файлах из переменных окружения с помощью атрибута `from_env`. [#2741](#)

.Добавлены регистронезависимые версии функций `coalesce`, `ifNull`, `nullIf`. [#2752](#)

## :Исправление ошибок

.Исправлена возможная ошибка при старте реплики. [#2759](#)

## ClickHouse release 18.4.0, 2018-07-28

### :Новые возможности

Добавлены системные таблицы `formats`, `data_type_families`, `aggregate_function_combinators`, `table_functions`, `table_engines`, `collations` [#2721](#)

Добавлена возможность использования табличной функции вместо таблицы в качестве аргумента табличной функции `remote` и `cluster` [#2708](#)

.Поддержка `HTTP Basic` аутентификации в протоколе репликации [#2727](#)

В функции `has` добавлена возможность поиска в массиве значений типа `Enum` по числовому значению [Maxim Khrisanfov](#)

.Поддержка добавления произвольных разделителей сообщений в процессе чтения из `Kafka` [Amos Bird](#)

## :Улучшения

Запрос `ALTER TABLE t DELETE WHERE` не перезаписывает куски данных, которые не были затронуты условием `WHERE` [#2694](#)

Настройка `use_minimalistic_checksums_in_zookeeper` таблиц семейства `ReplicatedMergeTree` включена по умолчанию. Эта настройка была добавлена в версии 1.1.54378, 2018-04-16. Установка версий, более старых, чем 1.1.54378, становится невозможной

.Поддерживается запуск запросов `KILL` и `OPTIMIZE` с указанием `ON CLUSTER` [Winter Zhang](#)

## :Исправление ошибок

Исправлена ошибка `Column ... is not under aggregate function and not in GROUP BY` в случае агрегации по (выражению с оператором `IN`. Ошибка появилась в версии 18.1.0. [\(bbdd780b\)](#)

.Исправлена ошибка в агрегатной функции `windowFunnel` [Winter Zhang](#)

(Исправлена ошибка в агрегатной функции `anyHeavy` [\(a2101df2\)](#)

.)Исправлено падение сервера при использовании функции `countArray`

## :Обратно несовместимые изменения

Список параметров для таблиц `Kafka` был изменён с `Kafka(kafka_broker_list, kafka_topic_list, kafka_group_name, kafka_format[, kafka_schema, kafka_num_consumers])` на `Kafka(kafka_broker_list, kafka_topic_list, kafka_group_name, kafka_format[, kafka_row_delimiter, kafka_schema, kafka_num_consumers])`. Если вы использовали параметры `kafka_schema` или `kafka_num_consumers`, вам необходимо вручную отредактировать файлы с метаданными `path/metadata/database/table.sql`, добавив параметр `kafka_row_delimiter` со значением `"` в соответствующее место

## ClickHouse release 18.1.0, 2018-07-23

## :Новые возможности

- .(Поддержка запроса `ALTER TABLE t DELETE WHERE` для нереплицированных MergeTree-таблиц (#2634
- .(Поддержка произвольных типов для семейства агрегатных функций `uniq*` (#2010
- .(Поддержка произвольных типов в операторах сравнения (#2026
- Возможность в `users.xml` указывать маску подсети в формате `10.0.0.1/255.255.255.0`. Это необходимо для
- .(использования "дырявых" масок IPv6 сетей (#2637
- .(Добавлена функция `arrayDistinct` (#2670
- Движок SummingMergeTree теперь может работать со столбцами типа AggregateFunction (Constantin S.
- .(Pan

## :Улучшения

- Изменена схема версионирования релизов. Теперь первый компонент содержит год релиза (A.D.; по
- московскому времени; из номера вычитается 2000), второй - номер крупных изменений
- (увеличивается для большинства релизов), третий - патч-версия. Релизы по-прежнему обратно
- совместимы, если другое не указано в changelog
- .(Ускорено преобразование чисел с плавающей точкой в строку (Amos Bird
- Теперь, если при вставке из-за ошибок парсинга пропущено некоторое количество строк (такое
- возможно про включённых настройках `input_allow_errors_num`, `input_allow_errors_ratio`), это количество
- .(пишется в лог сервера (Leonardo Cecchi

## :Исправление ошибок

- .(Исправлена работа команды TRUNCATE для временных таблиц (Amos Bird
- Исправлен редкий deadlock в клиентской библиотеке ZooKeeper, который возникал при сетевой
- .(ошибке во время вычитывания ответа (с315200
- .(Исправлена ошибка при CAST в Nullable типы (#1322
- Исправлен неправильный результат функции `maxIntersection()` в случае совпадения границ отрезков
- .((Michael Furmur
- .(Исправлено неверное преобразование цепочки OR-выражений в аргументе функции (chenxing-xc
- Исправлена деградация производительности запросов, содержащих выражение `IN (подзапрос)` внутри
- .(другого подзапроса (#2571
- Исправлена несовместимость серверов разных версий при распределённых запросах, использующих
- .(функцию `CAST` не в верхнем регистре (fe8c4d6
- .(Добавлено недостающее квотирование идентификаторов при запросах к внешним СУБД (#2635

## :Обратно несовместимые изменения

- Не работает преобразование строки, содержащей число ноль, в DateTime. Пример: `SELECT`
- `toDateTime('0')`. По той же причине не работает `DateTime DEFAULT '0'` в таблицах, а также
- `<null_value>0</null_value>` в словарях. Решение: заменить 0 на `0000-00-00 00:00:00`

## ClickHouse release 1.1.54394, 2018-07-12

### :Новые возможности

- .(Добавлена агрегатная функция `histogram` (Михаил Сурин
- Возможность использования `OPTIMIZE TABLE ... FINAL` без указания партиции для `ReplicatedMergeTree`
- .((Amos Bird

### :Исправление ошибок

- Исправлена ошибка - выставление слишком маленького таймаута у сокетов (одна секунда) для
- чтения и записи при отправке и скачивании реплицируемых данных, что приводило к невозможности
- скачать куски достаточно большого размера при наличии некоторой нагрузки на сеть или диск
- .(попытки скачивания кусков циклически повторяются). Ошибка возникла в версии 1.1.54388
- Исправлена работа при использовании chroot в ZooKeeper, в случае вставки дублирующихся блоков
- .данных в таблицу
- .(Исправлена работа функции `has` для случая массива с Nullable элементами (#2115

- Исправлена работа таблицы `system.tables` при её использовании в распределённых запросах; столбцы `metadata_modification_time` и `engine_full` сделаны не виртуальными; исправлена ошибка в случае, если из таблицы были запрошены только эти столбцы
- Исправлена работа пустой таблицы типа `TinyLog` после вставки в неё пустого блока данных ([#2563](#))
- Таблица `system.zookeeper` работает в случае, если значение узла в ZooKeeper равно NULL

## ClickHouse release 1.1.54390, 2018-07-06

### :Новые возможности

- Возможность отправки запроса в формате `multipart/form-data` (в поле `query`), что полезно, если при этом также отправляются внешние данные для обработки запроса ([Ольга Хвостикова](#))
- Добавлена возможность включить или отключить обработку одинарных или двойных кавычек при чтении данных в формате CSV. Это задаётся настройками `format_csv_allow_single_quotes` и `format_csv_allow_double_quotes` ([Amos Bird](#))
- Возможность использования `OPTIMIZE TABLE ... FINAL` без указания партиции для не реплицированных вариантов `MergeTree` ([Amos Bird](#))

### :Улучшения

- Увеличена производительность, уменьшено потребление памяти, добавлен корректный учёт потребления памяти, при использовании оператора `IN` в случае, когда для его работы может использоваться индекс таблицы ([#2584](#))
- Убраны избыточные проверки чексумм при добавлении куска. Это важно в случае большого количества реплик, так как в этом случае суммарное количество проверок было равно  $N^2$
- Добавлена поддержка аргументов типа `Array(Tuple(...))` для функции `arrayEnumerateUniq` ([#2573](#))
- Добавлена поддержка `Nullable` для функции `runningDifference`. ([#2594](#))
- Увеличена производительность анализа запроса в случае очень большого количества выражений ([#2572](#))
- Более быстрый выбор кусков для слияния в таблицах типа `ReplicatedMergeTree`. Более быстрое восстановление сессии с ZooKeeper. ([#2597](#))
- Файл `format_version.txt` для таблиц семейства `MergeTree` создаётся заново при его отсутствии, что имеет смысл в случае запуска ClickHouse после копирования структуры директорий без файлов ([Ciprian Hacman](#))

### :Исправление ошибок

- Исправлена ошибка при работе с ZooKeeper, которая могла приводить к невозможности восстановления сессии и readonly состояниям таблиц до перезапуска сервера
- Исправлена ошибка при работе с ZooKeeper, которая могла приводить к не удалению старых узлов при разрыве сессии
- Исправлена ошибка в функции `quantileTDigest` для Float аргументов (ошибка появилась в версии 1.1.54388) ([Михаил Сурин](#))
- Исправлена ошибка работы индекса таблиц типа `MergeTree`, если в условии, столбец первичного ключа расположен внутри функции преобразования типов между знаковым и беззнаковым целым одного размера ([#2603](#))
- Исправлен `segfault`, если в конфигурационном файле нет `macros`, но они используются ([#2570](#))
- Исправлено переключение на базу данных по-умолчанию при переподключении клиента ([#2583](#))
- Исправлена ошибка в случае отключенной настройки `use_index_for_in_with_subqueries`

### :Исправления безопасности

- При соединениях с MySQL удалена возможность отправки файлов (`LOAD DATA LOCAL INFILE`)

## ClickHouse release 1.1.54388, 2018-06-28

### :Новые возможности

- Добавлена поддержка запроса `ALTER TABLE t DELETE WHERE` для реплицированных таблиц и таблица `system.mutations`

- Добавлена поддержка запроса `ALTER TABLE t [REPLACE|ATTACH] PARTITION` для \*MergeTree-таблиц
- (Добавлена поддержка запроса `TRUNCATE TABLE` (Winter Zhang
- Добавлено несколько новых `SYSTEM`-запросов для реплицированных таблиц (`RESTART REPLICAS`, `SYNC`
- `[(REPLICA, [STOP|START] [MERGES|FETCHES|REPLICATED SENDS|REPLICATION QUEUES`
- Добавлена возможность записи в таблицу с движком MySQL и соответствующую табличную функцию
- `((sundy-li`
- (Добавлена табличная функция `url()` и движок таблиц `URL` (Александр Сапин
- (Добавлена агрегатная функция `windowFunnel` (sundy-li
- (Добавлены функции `startsWith` и `endsWith` для строк (Вадим Плахтинский
- (В табличной функции `numbers()` добавлена возможность указывать offset (Winter Zhang
- Добавлена возможность интерактивного ввода пароля в `clickhouse-client`
- (Добавлена возможность отправки логов сервера в syslog (Александр Крашенинников
- (Добавлена поддержка логирования в словарях с источником shared library (Александр Сапин
- (Добавлена поддержка произвольного разделителя в формате CSV (Иван Жуков
- Добавлена настройка `date_time_input_format`. Если переключить эту настройку в значение 'best\_effort',
- значения DateTime будут читаться в широком диапазоне форматов
- Добавлена утилита `clickhouse-obfuscator` для обфускации данных. Пример использования: публикация
- данных, используемых в тестах производительности

## :Экспериментальные возможности

- Добавлена возможность вычислять аргументы функции `and` только там, где они нужны (Анастасия
- (Царькова
- (Добавлена возможность JIT-компиляции в нативный код некоторых выражений (pyos

## :Исправление ошибок

- Исправлено появление дублей в запросе с `DISTINCT` и `ORDER BY`
- Запросы с `ARRAY JOIN` и `arrayFilter` раньше возвращали некорректный результат
- (Исправлена ошибка при чтении столбца-массива из Nested-структуры (#2066
- (...) Исправлена ошибка при анализе запросов с секцией HAVING вида `HAVING tuple IN`
- Исправлена ошибка при анализе запросов с рекурсивными алиасами
- Исправлена ошибка при чтении из ReplacingMergeTree с условием в PREWHERE, фильтрующим все
- строки (#2525
- Настройки профиля пользователя не применялись при использовании сессий в HTTP-интерфейсе
- Исправлено применение настроек из параметров командной строки в программе clickhouse-local
- Клиентская библиотека ZooKeeper теперь использует таймаут сессии, полученный от сервера
- Исправлена ошибка в клиентской библиотеке ZooKeeper, из-за которой ожидание ответа от сервера
- могло длиться дольше таймаута
- Исправлено отсечение ненужных кусков при запросе с условием на столбцы ключа
- (партиционирования (#2342
- (После `CLEAR COLUMN IN PARTITION` в соответствующей партии теперь возможны слияния (#2315
- (Исправлено соответствие типов в табличной функции ODBC (sundy-li
- (Исправлено некорректное сравнение типов `DateTime` с таймзоной и без неё (Александр Бочаров
- Исправлен синтаксический разбор и форматирование оператора `CAST`
- Исправлена вставка в материализованное представление в случае, если движок таблицы
- (представления - Distributed (Babacar Diassé
- Исправлен race condition при записи данных из движка `Kafka` в материализованные представления
- ((Yangkuan Liu
- (Исправлена SSRF в табличной функции `remote`
- (Исправлен выход из `clickhouse-client` в multiline-режиме (#2510

## :Улучшения

- Фоновые задачи в реплицированных таблицах теперь выполняются не в отдельных потоках, а в пуле
- (потоков (Silviu Caragea
- Улучшена производительность разжатия LZ4
- Ускорен анализ запроса с большим числом JOIN-ов и подзапросов

- .DNS-кэш теперь автоматически обновляется при большом числе сетевых ошибок
- Вставка в таблицу теперь не происходит, если вставка в одно из её материализованных представлений невозможна из-за того, что в нём много кусков
- .Исправлено несоответствие в значениях счётчиков событий [Query](#), [SelectQuery](#), [InsertQuery](#)
- .Разрешены выражения вида [tuple IN \(SELECT tuple\)](#), если типы кортежей совпадают
- Сервер с реплицированными таблицами теперь может стартовать, даже если не сконфигурирован
- .ZooKeeper
- .(При расчёте количества доступных ядер CPU теперь учитываются ограничения cgroups ([Atri Sharma](#)
- .(Добавлен shown директорий конфигов в конфигурационном файле systemd ([Михаил Ширяев](#)

## :Изменения сборки

- .Добавлена возможность сборки компилятором gcc8
- .Добавлена возможность сборки llvm из submodule
- .Используемая версия библиотеки librdkafka обновлена до v0.11.4
- Добавлена возможность использования библиотеки libspuid из системы, используемая версия библиотеки обновлена до 0.4.0
- .(Исправлена сборка с использованием библиотеки vectorclass ([Babacar Diassé](#)
- .(Стаке теперь по умолчанию генерирует файлы для ninja (как при использовании [-G Ninja](#)
- .(Добавлена возможность использования библиотеки libtinfo вместо libtermcap ([Георгий Кондратьев](#)
- .(Исправлен конфликт заголовочных файлов в Fedora Rawhide ([#2520](#)

## :Обратно несовместимые изменения

- .Убран escaping в форматах [Vertical](#) и [Pretty\\*](#), удалён формат [VerticalRaw](#)
- Если в распределённых запросах одновременно участвуют серверы версии 1.1.54388 или новее и более старые, то при использовании выражения [cast\(x, 'Type'\)](#), записанного без указания [AS](#), если слово [cast](#) указано не в верхнем регистре, возникает ошибка вида [Not found column cast\(0, 'UInt8'\) in block](#).
- .Решение: обновить сервер на всём кластере

## ClickHouse release 1.1.54385, 2018-06-01

### :Исправление ошибок

- .Исправлена ошибка, которая в некоторых случаях приводила к блокировке операций с ZooKeeper

## ClickHouse release 1.1.54383, 2018-05-22

### :Исправление ошибок

- Исправлена деградация скорости выполнения очереди репликации при большом количестве реплик

## ClickHouse release 1.1.54381, 2018-05-14

### :Исправление ошибок

- Исправлена ошибка, приводящая к "утеканию" метаданных в ZooKeeper при потере соединения с сервером ZooKeeper

## ClickHouse release 1.1.54380, 2018-04-21

### :Новые возможности

- Добавлена табличная функция [file\(path, format, structure\)](#). Пример, читающий байты из [/dev/urandom](#): [ln -s /dev/urandom /var/lib/clickhouse/user\\_files/random clickhouse-client -q "SELECT \\* FROM file\('random', 'RowBinary', 'd UInt8'\)"](#) [."LIMIT 10](#)

### :Улучшения

- Добавлена возможность оборачивать подзапросы скобками [\(\)](#) для повышения читаемости запросов.
- .(Например: [\(SELECT 1\) UNION ALL \(SELECT 1](#)
- Простые запросы [SELECT](#) из таблицы [system.processes](#) не учитываются в ограничении [.max\\_concurrent\\_queries](#)

## :Исправление ошибок

- Исправлена неправильная работа оператора **IN** в **MATERIALIZED VIEW**
- Исправлена неправильная работа индекса по ключу партиционирования в выражениях типа **(...) partition\_key\_column IN**
- Исправлена невозможность выполнить **OPTIMIZE** запрос на лидирующей реплике после выполнения **RENAME** таблицы
- Исправлены ошибки авторизации возникающие при выполнении запросов **OPTIMIZE** и **ALTER** на нелидирующей реплике
- Исправлены зависания запросов **KILL QUERY**
- Исправлена ошибка в клиентской библиотеке ZooKeeper, которая при использовании непустого префикса **chroot** в конфигурации приводила к потере watch'ей, остановке очереди distributed DDL запросов и замедлению репликации

## :Обратно несовместимые изменения

- Убрана поддержка выражений типа **(a, b) IN (SELECT (a, b))** (можно использовать эквивалентные выражение **(a, b) IN (SELECT a, b)**). Раньше такие запросы могли приводить к недетерминированной фильтрации в **WHERE**

## ClickHouse release 1.1.54378, 2018-04-16

### :Новые возможности

- Возможность изменения уровня логгирования без перезагрузки сервера
- Добавлен запрос **SHOW CREATE DATABASE**
- (Возможность передать **query\_id** в **clickhouse-client** (elBroom
- Добавлена настройка **max\_network\_bandwidth\_for\_all\_users**
- Добавлена поддержка **ALTER TABLE ... PARTITION ...** для **MATERIALIZED VIEW**
- Добавлена информация о размере кусков данных в несжатом виде в системные таблицы
- Поддержка межсерверного шифрования для distributed таблиц (**<secure>1</secure>** в конфигурации **(<реплики в <remote\_servers**
- Добавлена настройка уровня таблицы семейства **ReplicatedMergeTree** для уменьшения объема данных, хранимых в zookeeper: **use\_minimalistic\_checksums\_in\_zookeeper = 1**
- Возможность настройки приглашения **clickhouse-client**. По-умолчанию добавлен вывод имени сервера в приглашение. Возможность изменить отображаемое имя сервера. Отправка его в HTTP заголовке **X-ClickHouse-Display-Name** (Kirill Shvakov
- (Возможность указания нескольких **topics** через запятую для движка **Kafka** (Tobias Adamson
- При остановке запроса по причине **KILL QUERY** или **replace\_running\_query**, клиент получает исключение **Query was cancelled** вместо неполного результата

### :Улучшения

- Запросы вида **ALTER TABLE ... DROP/DETACH PARTITION** выполняются впереди очереди репликации
- Возможность использовать **SELECT ... FINAL** и **OPTIMIZE ... FINAL** даже в случае, если данные в таблице представлены одним куском
- Пересоздание таблицы **query\_log** налету в случае если было произведено её удаление вручную (Kirill (Shvakov
- (Ускорение функции **lengthUTF8** (zhang2014
- Улучшена производительность синхронной вставки в **Distributed** таблицы (**insert\_distributed\_sync = 1**) в случае очень большого количества шардов
- Сервер принимает настройки **send\_timeout** и **receive\_timeout** от клиента и применяет их на своей стороне для соединения с клиентом (в переставленном порядке: **send\_timeout** у сокета на стороне сервера (выставляется в значение **receive\_timeout** принятое от клиента, и наоборот
- Более надёжное восстановление после сбоев при асинхронной вставке в **Distributed** таблицы
- (Возвращаемый тип функции **countEqual** изменён с **UInt32** на **UInt64** (谢磊

## :Исправление ошибок

- Исправлена ошибка с **IN** где левая часть выражения **Nullable**



- Исправлен неправильный результат при использовании кортежей с **IN** в случае, если часть компонентов кортежа есть в индексе таблицы
- Исправлена работа ограничения **max\_execution\_time** с распределенными запросами
- Исправлены ошибки при вычислении размеров составных столбцов в таблице **system.columns**
- Исправлена ошибка при создании временной таблицы **CREATE TEMPORARY TABLE IF NOT EXISTS**
- (Исправлены ошибки в **StorageKafka** (#2075)
- Исправлены падения сервера от некорректных аргументов некоторых агрегатных функций
- Исправлена ошибка, из-за которой запрос **DETACH DATABASE** мог не приводить к остановке фоновых задач таблицы типа **ReplicatedMergeTree**
- Исправлена проблема с появлением **Too many parts** в агрегирующих материализованных представлениях (#2084)
- Исправлена рекурсивная обработка подстановок в конфиге, если после одной подстановки, требуется другая подстановка на том же уровне
- Исправлена ошибка с неправильным синтаксисом в файле с метаданными при создании **VIEW**, использующих запрос с **UNION ALL**
- Исправлена работа **SummingMergeTree** в случае суммирования вложенных структур данных с составным ключом
- Исправлена возможность возникновения race condition при выборе лидера таблиц **ReplicatedMergeTree**

## :Изменения сборки

- Поддержка **ninja** вместо **make** при сборке. **ninja** используется по-умолчанию при сборке релизов
- Переименованы пакеты **clickhouse-server-base** в **clickhouse-common-static**; **clickhouse-server-common** в **clickhouse-server**; **clickhouse-common-dbg** в **clickhouse-common-static-dbg**. Для установки используйте **clickhouse-server** **clickhouse-client**. Для совместимости, пакеты со старыми именами продолжают загружаться в репозиторий

## :Обратно несовместимые изменения

- Удалена специальная интерпретация выражения **IN**, если слева указан массив. Ранее выражение вида **arr IN (set)** воспринималось как "хотя бы один элемент **arr** принадлежит множеству **set**". Для получения (такого же поведения в новой версии, напишите **arrayExists(x -> x IN (set), arr)**
- Отключено ошибочное использование опции сокета **SO\_REUSEPORT** (которая по ошибке включена по-умолчанию в библиотеке POCO). Стоит обратить внимание, что на Linux системах теперь не имеет смысла указывать одновременно адреса **::** и **0.0.0.0** для **listen** - следует использовать лишь адрес **::**, который (с настройками ядра по-умолчанию) позволяет слушать соединения как по IPv4 так и по IPv6. Также вы можете вернуть поведение старых версий, указав в конфиге **<listen\_reuse\_port>1</listen\_reuse\_port>**

## ClickHouse release 1.1.54370, 2018-03-16

### :Новые возможности

- Добавлена системная таблица **system.macros** и автоматическое обновление макросов при изменении конфигурационного файла
- Добавлен запрос **SYSTEM RELOAD CONFIG**
- Добавлена агрегатная функция **maxIntersections(left\_col, right\_col)**, возвращающая максимальное количество одновременно пересекающихся интервалов **[left; right]**. Функция **maxIntersectionsPosition(left, (right))** возвращает начало такого "максимального" интервала. (**Michael Furmur**)

### :Улучшения

- При вставке данных в **Replicated**-таблицу делается меньше обращений к **ZooKeeper** (также из лога **ZooKeeper** исчезло большинство user-level ошибок)
- Добавлена возможность создавать алиасы для множеств. Пример: **WITH (1, 2, 3) AS set SELECT number IN set FROM system.numbers LIMIT 10**

### :Исправление ошибок

- Исправлена ошибка **Illegal PREWHERE** при чтении из Merge-таблицы над **Distributed**-таблицами

- Добавлены исправления, позволяющие запускать clickhouse-server в IPv4-only docker-контейнерах
- Исправлен race condition при чтении из системной таблицы `system.parts_columns`
- Убрана двойная буферизация при синхронной вставке в `Distributed`-таблицу, которая могла приводить к `.timeout`-ам соединений
- Исправлена ошибка, вызывающая чрезмерно долгое ожидание недоступной реплики перед началом выполнения `SELECT`
- Исправлено некорректное отображение дат в таблице `system.parts`
- Исправлена ошибка, приводящая к невозможности вставить данные в `Replicated`-таблицу, если в конфигурации кластера `ZooKeeper` задан непустой `chroot`
- Исправлен алгоритм вертикального мержа при пустом ключе `ORDER BY` таблицы
- Возвращена возможность использовать словари в запросах к удаленным таблицам, даже если этих словарей нет на сервере-инициаторе. Данная функциональность была потеряна в версии 1.1.54362
- Восстановлено поведение, при котором в запросах типа `SELECT * FROM remote('server2', default.table) WHERE col IN (SELECT col2 FROM default.table)` в правой части `IN` должна использоваться удаленная таблица `.default.table`, а не локальная. данное поведение было нарушено в версии 1.1.54358
- Устранено ненужное Error-level логирование `Not found column ... in block`

## Релиз ClickHouse 1.1.54362, 2018-03-11

### :Новые возможности

- Агрегация без `GROUP BY` по пустому множеству (как например, `SELECT count(*) FROM table WHERE 0`) теперь возвращает результат из одной строки с нулевыми значениями агрегатных функций, в соответствии со стандартом SQL. Вы можете вернуть старое поведение (возвращать пустой результат), выставив настройку `empty_result_for_aggregation_by_empty_set` в значение 1
- Добавлено приведение типов при `UNION ALL`. Допустимо использование столбцов с разными алиасами в соответствующих позициях `SELECT` в `UNION ALL`, что соответствует стандарту SQL
- Поддержка произвольных выражений в секции `LIMIT BY`. Ранее было возможно лишь использование столбцов - результата `SELECT`
- Использование индекса таблиц семейства `MergeTree` при наличии условия `IN` на кортеж от выражений от столбцов первичного ключа. Пример `WHERE (UserID, EventDate) IN ((123, '2000-01-01'), ...)` (Anastasiya .(Tsarkova)
- Добавлен инструмент `clickhouse-copier` для межкластерного копирования и перешардирования данных .((бета
- Добавлены функции консистентного хэширования `yandexConsistentHash`, `jumpConsistentHash`, `sumburConsistentHash`. Их можно использовать в качестве ключа шардирования для того, чтобы уменьшить объём сетевого трафика при последующих перешардированиях
- Добавлены функции `arrayAny`, `arrayAll`, `hasAny`, `hasAll`, `arrayIntersect`, `arrayResize`
- .(Добавлена функция `arrayCumSum` (Javi Santana
- Добавлена функция `parseDateTimeBestEffort`, `parseDateTimeBestEffortOrZero`, `parseDateTimeBestEffortOrNull`, позволяющая прочитать `DateTime` из строки, содержащей текст в широком множестве возможных форматов
- Возможность частичной перезагрузки данных внешних словарей при их обновлении (загрузка лишь .(записей со значением заданного поля большим, чем при предыдущей загрузке) (Arsen Hakobyan
- Добавлена табличная функция `cluster`. Пример: `cluster(cluster_name, db, table)`. Табличная функция `remote` может принимать имя кластера в качестве первого аргумента, если оно указано в виде идентификатора
- .Возможность использования табличных функций `remote`, `cluster` в `INSERT` запросах
- Добавлены виртуальные столбцы `create_table_query`, `engine_full` в таблице `system.tables`. Столбец `.metadata_modification_time` сделан виртуальным
- Добавлены столбцы `data_path`, `metadata_path` в таблицы `system.tables` и `system.databases`, а также столбец `.path` в таблицы `system.parts` и `system.parts_columns`
- .Добавлена дополнительная информация о слияниях в таблице `system.part_log`
- Возможность использования произвольного ключа партиционирования для таблицы `system.query_log` .((Kirill Shvakov
- Запрос `SHOW TABLES` теперь показывает также и временные таблицы. Добавлены временные таблицы и .(столбец `is_temporary` в таблице `system.tables` (zhang2014



- .(Добавлен запрос `DROP TEMPORARY TABLE, EXISTS TEMPORARY TABLE` (zhang2014
- .(Поддержка `SHOW CREATE TABLE` для временных таблиц (zhang2014
- Добавлен конфигурационный параметр `system_profile` для настроек, используемых внутренними процессами
- Поддержка загрузки `object_id` в качестве атрибута в словарях с источником `MongoDB` (Павел Литвиненко
- .(Литвиненко
- Возможность читать `null` как значение по-умолчанию при загрузке данных для внешнего словаря с .(источником `MongoDB` (Павел Литвиненко
- Возможность чтения значения типа `DateTime` в формате `Values` из unix timestamp без одинарных кавычек
- Поддержан failover в табличной функции `remote` для случая, когда на части реплик отсутствует запрошенная таблица
- Возможность переопределять параметры конфигурации в параметрах командной строки при запуске `.clickhouse-server`, пример: `clickhouse-server -- --logger.level=information`
- Реализована функция `empty` от аргумента типа `FixedString`: функция возвращает 1, если строка состоит .(полностью из нулевых байт (zhang2014
- Добавлен конфигурационный параметр `listen_try`, позволяющий слушать хотя бы один из listen адресов и не завершать работу, если некоторые адреса не удаётся слушать (полезно для систем с .(выключенной поддержкой IPv4 или IPv6
- .Добавлен движок таблиц `VersionedCollapsingMergeTree`
- .Поддержка строк и произвольных числовых типов для источника словарей `library`
- Возможность использования таблиц семейства `MergeTree` без первичного ключа (для этого необходимо .(указать `ORDER BY tuple`
- Добавлена возможность выполнить преобразование (`CAST`) `Nullable` типа в не `Nullable` тип, если аргумент .не является `NULL`
- .Возможность выполнения `RENAME TABLE` для `VIEW`
- .Добавлена функция `throwIf`
- Добавлена настройка `odbc_default_field_size`, позволяющая расширить максимальный размер значения, .(загружаемого из ODBC источника (по-умолчанию - 1024
- .В таблицу `system.processes` и в `SHOW PROCESSLIST` добавлены столбцы `is_cancelled` и `peak_memory_usage`

## :Улучшения

- Ограничения на результат и квоты на результат теперь не применяются к промежуточным данным .для запросов `INSERT SELECT` и для подзапросов в `SELECT`
- Уменьшено количество ложных срабатываний при проверке состояния `Replicated` таблиц при запуске .сервера, приводивших к необходимости выставления флага `force_restore_data`
- .Добавлена настройка `allow_distributed_ddl`
- Запрещено использование недетерминированных функций в выражениях для ключей таблиц .семейства `MergeTree`
- .Файлы с подстановками из `config.d` директорий загружаются в алфавитном порядке
- Увеличена производительность функции `arrayElement` в случае константного многомерного массива с .[пустым массивом в качестве одного из элементов. Пример: `[[1], []][x]`
- Увеличена скорость запуска сервера при использовании конфигурационных файлов с очень большими .(подстановками (например, очень большими списками IP-сетей
- При выполнении запроса, табличные функции выполняются один раз. Ранее табличные функции `remote`, `mysql` дважды делали одинаковый запрос на получение структуры таблицы с удалённого .сервера
- .Используется генератор документации `MkDocs`
- При попытке удалить столбец таблицы, от которого зависят `DEFAULT/MATERIALIZED` выражения других .(столбцов, кидается исключение (zhang2014
- Добавлена возможность парсинга пустой строки в текстовых форматах как числа 0 для `Float` типов .данных. Эта возможность присутствовала раньше, но была потеряна в релизе 1.1.54342
- Значения типа `Enum` можно использовать в функциях `min`, `max`, `sum` и некоторых других - в этих случаях используются соответствующие числовые значения. Эта возможность присутствовала ранее, но была .потеряна в релизе 1.1.54337

Добавлено ограничение `max_expanded_ast_elements` действующее на размер AST после рекурсивного раскрытия алиасов

## :Исправление ошибок

Исправлены случаи ошибочного удаления ненужных столбцов из подзапросов, а также отсутствие удаления ненужных столбцов из подзапросов, содержащих `UNION ALL`

Исправлена ошибка в слияниях для таблиц типа `ReplacingMergeTree`

(Исправлена работа синхронного режима вставки в `Distributed` таблицы (`insert_distributed_sync = 1`

Исправлены `segfault` при некоторых случаях использования `FULL` и `RIGHT JOIN` с дублирующимися столбцами в подзапросах

Исправлены `segfault`, которые могут возникать при использовании функциональности

`.replace_running_query` и `KILL QUERY`

Исправлен порядок столбцов `source` и `last_exception` в таблице `system.dictionaries`

Исправлена ошибка - запрос `DROP DATABASE` не удалял файл с метаданными

Исправлен запрос `DROP DATABASE` для базы данных типа `Dictionary`

Исправлена неоправданно низкая точность работы функций `uniqHLL12` и `uniqCombined` для (кардинальностей больше 100 млн. элементов (Alex Bocharov

Исправлено вычисление неявных значений по-умолчанию при необходимости одновременного

(вычисления явных выражений по-умолчанию в запросах `INSERT` (zhang2014

Исправлен редкий случай, в котором запрос к таблице типа `MergeTree` мог не завершаться (chenxing-.xc

Исправлено падение при выполнении запроса `CHECK` для `Distributed` таблиц, если все шарды локальные ((chenxing.xc

Исправлена незначительная регрессия производительности при работе функций, использующих регулярные выражения

Исправлена регрессия производительности при создании многомерных массивов от сложных выражений

Исправлена ошибка, из-за которой в `.sql` файл с метаданными может записываться лишняя секция `.FORMAT`

Исправлена ошибка, приводящая к тому, что ограничение `max_table_size_to_drop` действует при попытке удаления `MATERIALIZED VIEW`, смотрящего на явно указанную таблицу

Исправлена несовместимость со старыми клиентами (на старые клиенты могли отправляться данные (с типом `DateTime('timezone')`), который они не понимают

Исправлена ошибка при чтении столбцов-элементов `Nested` структур, которые были добавлены с помощью `ALTER`, но являются пустыми для старых партиций, когда условия на такие столбцы переносятся в `PREWHERE`

Исправлена ошибка при фильтрации таблиц по условию на виртуальных столбец `_table` в запросах к таблицам типа `Merge`

Исправлена ошибка при использовании `ALIAS` столбцов в `Distributed` таблицах

Исправлена ошибка, приводящая к невозможности динамической компиляции запросов с агрегатными функциями из семейства `quantile`

Исправлен race condition в конвейере выполнения запроса, который мог проявляться в очень редких случаях при использовании `Merge` таблиц над большим количеством таблиц, а также при использовании `GLOBAL` подзапросов

Исправлено падение при передаче массивов разных размеров в функцию `arrayReduce` при использовании агрегатных функций от нескольких аргументов

.Запрещено использование запросов с `UNION ALL` в `MATERIALIZED VIEW`

Исправлена ошибка, которая может возникать при инициализации системной таблицы `part_log` при (старте сервера (по-умолчанию `part_log` выключен

## :Обратно несовместимые изменения

Удалена настройка `distributed_ddl_allow_replicated_alter`. Соответствующее поведение включено по-умолчанию

Удалена настройка `strict_insert_defaults`. Если вы использовали эту функциональность, напишите на [clickhouse-feedback@yandex-team.com](mailto:clickhouse-feedback@yandex-team.com)

.Удалён движок таблиц `UnsortedMergeTree`

## Релиз ClickHouse 1.1.54343, 2018-02-05

Добавлена возможность использовать макросы при задании имени кластера в распределенных DLL запросах и создании Distributed-таблиц: `CREATE TABLE distr ON CLUSTER '{cluster}' (...) ENGINE =`

`('Distributed('{cluster}', 'db', 'table`

Теперь при вычислении запросов вида `SELECT ... FROM table WHERE expr IN (subquery)` используется индекс таблицы `table`

Улучшена обработка дубликатов при вставке в Replicated-таблицы, теперь они не приводят к излишнему замедлению выполнения очереди репликации

## Релиз ClickHouse 1.1.54342, 2018-01-22

Релиз содержит исправление к предыдущему релизу 1.1.54337: \* Исправлена регрессия в версии 1.1.54337: если пользователь по-умолчанию имеет readonly доступ, то сервер отказывался стартовать с сообщением `Cannot create database in readonly mode`. \* Исправлена регрессия в версии 1.1.54337: на системах под управлением systemd, логи по ошибке всегда записываются в syslog; watchdog скрипт по ошибке использует init.d. \* Исправлена регрессия в версии 1.1.54337: неправильная конфигурация по-умолчанию в Docker образе. \* Исправлена недетерминированная работа GraphiteMergeTree (в логах видно по сообщениям `Data after merge is not byte-identical to data on another replica`). \* Исправлена ошибка, в связи с которой запрос OPTIMIZE к Replicated таблицам мог приводить к неконсистентным мержам (в логах видно по сообщениям `Part ... intersects previous part`). \* Таблицы типа Buffer теперь работают при наличии MATERIALIZED столбцов в таблице назначения (by zhang2014). \* Исправлена одна из ошибок в реализации .NULL

## Релиз ClickHouse 1.1.54337, 2018-01-18

### :Новые возможности

.Добавлена поддержка хранения многомерных массивов и кортежей (тип данных `Tuple`) в таблицах  
Поддержка табличных функций для запросов `DESCRIBE` и `INSERT`. Поддержка подзапроса в запросе `DESCRIBE`. Примеры: `DESC TABLE remote('host', default.hits); DESC TABLE (SELECT 1); INSERT INTO TABLE FUNCTION remote('host', default.hits)`. Возможность писать `INSERT INTO TABLE` вместо `INSERT INTO`

Улучшена поддержка часовых поясов. В типе `DateTime` может быть указана таймзона, которая используется для парсинга и отображения данных в текстовом виде. Пример: `DateTime('Europe/Moscow')`. При указании таймзоны в функциях работы с `DateTime`, тип возвращаемого значения будет запоминать таймзону, для того, чтобы значение отображалось ожидаемым образом

Добавлены функции `toTimeZone`, `timeDiff`, `toQuarter`, `toRelativeQuarterNum`. В функцию `toRelativeHour/Minute/Second` можно передать аргумент типа `Date`. Имя функции `now` воспринимается без учёта регистра

.(Добавлена функция `toStartOfFifteenMinutes` (Kirill Shvakov

.Добавлена программа `clickhouse format` для переформатирования запросов

Добавлен конфигурационный параметр `format_schema_path` (Marek Vavruša). Он используется для задания схемы для формата `Cap'n'Proto`. Файлы со схемой могут использоваться только из указанной директории

.(Добавлена поддержка `incl` и `conf.d` подстановок для конфигурации словарей и моделей (Pavel Yakunin

.(В таблице `system.settings` появилось описание большинства настроек (Kirill Shvakov

Добавлена таблица `system.parts_columns`, содержащая информацию о размерах столбцов в каждом куске данных `MergeTree` таблиц

.Добавлена таблица `system.models`, содержащая информацию о загруженных моделях `CatBoost`

Добавлены табличные функции `mysql` и `odbc` и соответствующие движки таблиц `MySQL`, `ODBC` для "обращения к удалённым базам данных. Функциональность в состоянии "бета

Для функции `groupArray` разрешено использование аргументов типа `AggregateFunction` (можно создать (массив из состояний агрегатных функций

Удалены ограничения на использование разных комбинаций комбинаторов агрегатных функций. Для примера, вы можете использовать как функцию `avgForEachIf`, так и `avgIfForEach`, которые имеют разный смысл

- Комбинатор агрегатных функций `-ForEach` расширен для случая агрегатных функций с более чем одним аргументом
- Добавлена поддержка агрегатных функций от `Nullable` аргументов, для случаев, когда функция всегда возвращает не `Nullable` результат (реализовано с участием Silviu Caragea). Пример: `groupArray`, `groupUniqArray`, `topK`
- (Добавлен параметр командной строки `max_client_network_bandwidth` для `clickhouse-client` (Kirill Shvakov
- Пользователям с доступом `readonly = 2` разрешено работать с временными таблицами (`CREATE`, `DROP`, `INSERT...`) (Kirill Shvakov
- Добавлена возможность указания количества `consumers` для `Kafka`. Расширена возможность (конфигурации движка `Kafka` (Marek Vavruša
- Добавлены функции `intExp2`, `intExp10`
- Добавлена агрегатная функция `sumKahan`
- Добавлены функции `toNumberOrNull`, где `Number` - числовой тип
- (Добавлена поддержка секции `WITH` для запроса `INSERT SELECT` (автор: zhang2014
- Добавлены настройки `http_connection_timeout`, `http_send_timeout`, `http_receive_timeout`. Настройки используются, в том числе, при скачивании кусков для репликации. Изменение этих настроек позволяет сделать более быстрый failover в случае перегруженной сети
- (Добавлена поддержка `ALTER` для таблиц типа `Null` (Anastasiya Tsarkova
- Функция `reinterpretAsString` расширена на все типы данных, значения которых хранятся в памяти непрерывно
- Для программы `clickhouse-local` добавлена опция `--silent` для подавления вывода информации о выполнении запроса в `stderr`
- Добавлена поддержка чтения `Date` в текстовом виде в формате, где месяц и день месяца могут быть (указаны одной цифрой вместо двух (Amos Bird

## :Увеличение производительности

- Увеличена производительность агрегатных функций `min`, `max`, `any`, `anyLast`, `anyHeavy`, `argMin`, `argMax` от строковых аргументов
- Увеличена производительность функций `isInfinite`, `isFinite`, `isNaN`, `roundToExp2`
- Увеличена производительность форматирования в текстовом виде и парсинга из текста значений типа `Date` и `DateTime`
- Увеличена производительность и точность парсинга чисел с плавающей запятой
- Уменьшено потребление памяти при `JOIN`, если левая и правая часть содержали столбцы с одинаковым именем, не входящие в `USING`
- Увеличена производительность агрегатных функций `varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr` за счёт уменьшения стойкости к вычислительной погрешности. Старые версии функций добавлены под именами `varSampStable`, `varPopStable`, `stddevSampStable`, `stddevPopStable`, `covarSampStable`, `covarPopStable`, `corrStable`

## :Исправления ошибок

- Исправлена работа дедупликации блоков после `DROP` или `DETACH PARTITION`. Раньше удаление партиции и вставка тех же самых данных заново не работала, так как вставленные заново блоки считались дубликатами
- Исправлена ошибка, в связи с которой может неправильно обрабатываться `WHERE` для запросов на создание `MATERIALIZED VIEW` с указанием `POPULATE`
- Исправлена ошибка в работе параметра `root_path` в конфигурации `zookeeper_servers`
- Исправлен неожиданный результат при передаче аргумента типа `Date` в функцию `toStartOfDay`
- Исправлена работа функции `addMonths`, `subtractMonths`, арифметика с `INTERVAL n MONTH`, если в результате получается предыдущий год
- Добавлена недостающая поддержка типа данных `UUID` для `DISTINCT`, `JOIN`, в агрегатных функциях `uniq` и во внешних словарях (Иванов Евгений). Поддержка `UUID` всё ещё остаётся не полной
- Исправлено поведение `SummingMergeTree` для строк, в которых все значения после суммирования равны нулю
- (Многочисленные доработки для движка таблиц `Kafka` (Marek Vavruša
- (Исправлена некорректная работа движка таблиц `Join` (Amos Bird

- Исправлена работа аллокатора под FreeBSD и OS X
- Функция `extractAll` теперь может доставать пустые вхождения
- Исправлена ошибка, не позволяющая подключить при сборке `libressl` вместо `openssl`
- Исправлена работа `CREATE TABLE AS SELECT` из временной таблицы
- Исправлена неатомарность обновления очереди репликации. Эта проблема могла приводить к
- рассинхронизации реплик и чинилась при перезапуске
- (Исправлено переполнение в функциях `gcd`, `lcm`, `modulo` (оператор `%`) (Maks Skorokhod
- Файлы `-preprocessed` теперь создаются после изменения `umask` (`umask` может быть задан в
- (конфигурационном файле
- Исправлена ошибка фоновой проверки кусков (`MergeTreePartChecker`) при использовании
- партиционирования по произвольному ключу
- Исправлен парсинг кортежей (значений типа `Tuple`) в текстовых форматах
- Исправлены сообщения о неподходящих типах аргументов для функций `multif`, `array` и некоторых
- других
- Переработана поддержка `Nullable` типов. Исправлены ошибки, которые могут приводить к падению
- сервера. Исправлено подавляющее большинство других ошибок, связанных с поддержкой `NULL`:
- неправильное приведение типов при `INSERT SELECT`, недостаточная поддержка `Nullable` в `HAVING` и в
- `PREWHERE`, режим `join_use_nulls`, `Nullable` типы в операторе `OR` и т. п
- Исправлена работа с внутренними свойствами типов данных, что позволило исправить проблемы
- следующего вида: ошибочное суммирование полей типа `Enum` в `SummingMergeTree`; значения типа `Enum`
- ошибочно выводятся с выравниванием по правому краю в `Pretty` форматах, и т. п
- Более строгие проверки для допустимых комбинаций составных столбцов - это позволило исправить
- ошибок, которые могли приводить к падениям
- Исправлено переполнение при задании очень большого значения параметра для типа `FixedString`
- Исправлена работа агрегатной функции `topK` для generic случая
- Добавлена отсутствующая проверка на совпадение размеров массивов для n-арных вариантов
- агрегатных функций с комбинатором `-Array`
- (Исправлена работа `--pager` для `clickhouse-client` (автор: ks1322
- Исправлена точность работы функции `exp10`
- Исправлено поведение функции `visitParamExtract` согласно документации
- Исправлено падение при объявлении некорректных типов данных
- Исправлена работа `DISTINCT` при условии, что все столбцы константные
- Исправлено форматирование запроса в случае наличия функции `tupleElement` со сложным константным
- выражением в качестве номера элемента
- Исправлена работа `Dictionary` таблиц для словарей типа `range_hashed`
- (Исправлена ошибка, приводящая к появлению лишних строк при `FULL` и `RIGHT JOIN` (Amos Bird
- Исправлено падение сервера в случае создания и удаления временных файлов в `config.d` директориях
- в момент перечитывания конфигурации
- Исправлена работа запроса `SYSTEM DROP DNS CACHE`: ранее сброс DNS кэша не приводил к повторному
- резолвингу имён хостов кластера
- Исправлено поведение `MATERIALIZED VIEW` после `DETACH TABLE` таблицы, на которую он смотрит (Marek
- (Vavruša

## :Улучшения сборки

- Для сборки используется `pbuilder`. Сборка максимально независима от окружения на сборочной
- машине
- Для разных версий систем выкладывается один и тот же пакет, который совместим с широким
- диапазоном Linux систем
- Добавлен пакет `clickhouse-test`, который может быть использован для запуска функциональных тестов
- Добавлена выкладка в репозиторий архива с исходниками. Этот архив может быть использован для
- воспроизведения сборки без использования GitHub
- Добавлена частичная интеграция с Travis CI. В связи с ограничениями на время сборки в Travis,
- запускается только ограниченный набор тестов на Debug сборке
- Добавлена поддержка `Cap'n'Proto` в сборку по-умолчанию
- Документация переведена с `Restructured Text` на `Markdown`



- Добавлена поддержка `systemd` (Vladimir Smirnov). В связи с несовместимостью с некоторыми образами, она выключена по-умолчанию и может быть включена вручную
- Для динамической компиляции запросов, `clang` и `lld` встроены внутрь `clickhouse`. Они также могут быть вызваны с помощью `clickhouse clang` и `clickhouse lld`
- Удалено использование расширений GNU из кода и включена опция `-Wextra`. При сборке с помощью `++clang` по-умолчанию используется `libc++` вместо `libstdc`
- Выделены библиотеки `clickhouse_parsers` и `clickhouse_common_io` для более быстрой сборки утилит

## :Обратно несовместимые изменения

- Формат засечек (marks) для таблиц типа `Log`, содержащих `Nullable` столбцы, изменён обратно-несовместимым образом. В случае наличия таких таблиц, вы можете преобразовать их в `TinyLog` до запуска новой версии сервера. Для этого в соответствующем таблице файле `.sql` в директории `metadata`, замените `ENGINE = Log` на `ENGINE = TinyLog`. Если в таблице нет `Nullable` столбцов или тип таблицы не `Log`, то ничего делать не нужно
- Удалена настройка `experimental_allow_extended_storage_definition_syntax`. Соответствующая функциональность включена по-умолчанию
- Функция `runningIncome` переименована в `runningDifferenceStartingWithFirstValue` во избежание путаницы
- (Удалена возможность написания `FROM ARRAY JOIN arr` без указания таблицы после `FROM` (Amos Bird
- Удалён формат `BlockTabSeparated`, использовавшийся лишь для демонстрационных целей
- Изменён формат состояния агрегатных функций `varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr`. Если вы использовали эти состояния для хранения в таблицах (тип данных `AggregateFunction` от этих функций или материализованные представления, хранящие эти состояния), напишите на [clickhouse-feedback@yandex-team.com](mailto:clickhouse-feedback@yandex-team.com)
- В предыдущих версиях существовала недокументированная возможность: в типе данных `AggregateFunction` можно было не указывать параметры для агрегатной функции, которая зависит от параметров. Пример: `AggregateFunction(quantiles, UInt64)` вместо `AggregateFunction(quantiles(0.5, 0.9), UInt64)`. Эта возможность потеряна. Не смотря на то, что возможность не документирована, мы собираемся вернуть её в ближайших релизах
- Значения типа данных `Enum` не могут быть переданы в агрегатные функции `min/max`. Возможность будет возвращена обратно в следующем релизе

## :На что обратить внимание при обновлении

- При обновлении кластера, на время, когда на одних репликах работает новая версия сервера, а на других - старая, репликация будет приостановлена и в логе появятся сообщения вида `unknown .parameter 'shard'`. Репликация продолжится после обновления всех реплик кластера
- Если на серверах кластера работают разные версии ClickHouse, то возможен неправильный результат распределённых запросов, использующих функции `varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr`. Необходимо обновить все серверы кластера

## Релиз ClickHouse 1.1.54327, 2017-12-21

Релиз содержит исправление к предыдущему релизу 1.1.54318: \* Исправлена проблема с возможным race condition при репликации, которая может приводить к потере данных. Проблема подвержены версии 1.1.54310 и 1.1.54318. Если вы их используете и у вас есть Replicated таблицы, то обновление обязательно. Понять, что эта проблема существует, можно по сообщениям в логе Warning вида `Part ... from own log doesn't .exist`. Даже если таких сообщений нет, проблема всё-равно актуальна

## Релиз ClickHouse 1.1.54318, 2017-11-30

Релиз содержит изменения к предыдущему релизу 1.1.54310 с исправлением следующих багов: \* Исправлено некорректное удаление строк при слияниях в движке `SummingMergeTree` \* Исправлена утечка памяти в нереплицированных `MergeTree`-движках \* Исправлена деградация производительности при частых вставках в `MergeTree`-движках \* Исправлена проблема, приводящая к остановке выполнения очереди репликации \* Исправлено ротирование и архивация логов сервера

## Релиз ClickHouse 1.1.54310, 2017-11-01

## :Новые возможности

- .Произвольный ключ партиционирования для таблиц семейства MergeTree
- .Движок таблиц **Kafka**
- .Возможность загружать модели **CatBoost** и применять их к данным, хранящимся в ClickHouse
- .Поддержка часовых поясов с нецелым смещением от UTC
- .Поддержка операций с временными интервалами
- .Диапазон значений типов Date и DateTime расширен до 2105 года
- Запрос **CREATE MATERIALIZED VIEW x TO y** (позволяет указать существующую таблицу для хранения (данных материализованного представления
- .Запрос **ATTACH TABLE** без аргументов
- Логика обработки Nested-столбцов в SummingMergeTree, заканчивающихся на -Map, вынесена в агрегатную функцию sumMap. Такие столбцы теперь можно задавать явно
- .Максимальный размер IP trie-словаря увеличен до 128M записей
- .Функция getSizeOfEnumType
- .Агрегатная функция sumWithOverflow
- .Поддержка входного формата Cap'n Proto
- .Возможность задавать уровень сжатия при использовании алгоритма zstd

## :Обратно несовместимые изменения

- .Запрещено создание временных таблиц с движком, отличным от Memory
- .Запрещено явное создание таблиц с движком View и MaterializedView
- .При создании таблицы теперь проверяется, что ключ сэмплирования входит в первичный ключ

## :Исправления ошибок

- .Исправлено зависание при синхронной вставке в Distributed таблицу
- .Исправлена неатомарность при добавлении/удалении кусков в реплицированных таблицах
- Данные, вставляемые в материализованное представление, теперь не подвергаются излишней дедупликации
- Запрос в Distributed таблицу, для которого локальная реплика отстаёт, а удалённые недоступны, теперь не падает
- .Для создания временных таблиц теперь не требуется прав доступа к БД **default**
- .Исправлено падение при указании типа Array без аргументов
- .Исправлено зависание при недостатке места на диске в разделе с логами
- .Исправлено переполнение в функции toRelativeWeekNum для первой недели Unix-эпохи

## :Улучшения сборки

- .Несколько сторонних библиотек (в частности, Poco) обновлены и переведены на git submodules

## Релиз ClickHouse 1.1.54304, 2017-10-19

### :Новые возможности

- (Добавлена поддержка TLS в нативном протоколе (включается заданием **tcp\_ssl\_port** в **config.xml**

### :Исправления ошибок

- ALTER** для реплицированных таблиц теперь пытается начать выполнение как можно быстрее
- Исправлены падения при чтении данных с настройкой **preferred\_block\_size\_bytes=0**
- Исправлено падение **clickhouse-client** при нажатии **Page Down**
- Корректная интерпретация некоторых сложных запросов с **GLOBAL IN** и **UNION ALL**
- Операция **FREEZE PARTITION** теперь работает всегда атомарно
- (Исправлено зависание пустых POST-запросов (теперь возвращается код 411
- ((Исправлены ошибки при интерпретации выражений типа **CAST(1 AS Nullable(UInt8**
- Исправлена ошибка при чтении колонок типа **Array(Nullable(String))** из **MergeTree** таблиц
- Исправлено падение при парсинге запросов типа **SELECT dummy AS dummy, dummy AS b**
- Корректное обновление пользователей при невалидном **users.xml**
- Корректная обработка случаев, когда executable-словарь возвращает ненулевой код ответа

# Релиз ClickHouse 1.1.54292, 2017-09-20

## :Новые возможности

- .Добавлена функция `pointInPolygon` для работы с координатами на плоскости
- Добавлена агрегатная функция `sumMap`, обеспечивающая суммирование массивов аналогично `.SummingMergeTree`
- Добавлена функция `trunc`. Увеличена производительность функций округления `round`, `floor`, `ceil`, `roundToExp2`. Исправлена логика работы функций округления. Изменена логика работы функции `.roundToExp2` для дробных и отрицательных чисел
- Ослаблена зависимость исполняемого файла ClickHouse от версии `libc`. Один и тот же исполняемый файл ClickHouse может запускаться и работать на широком множестве Linux систем. Замечание: зависимость всё ещё присутствует при использовании скомпилированных запросов (настройка `compile` `.(= 1`, по-умолчанию не используется
- .Уменьшено время динамической компиляции запросов

## :Исправления ошибок

- Исправлена ошибка, которая могла приводить к сообщениям `part ... intersects previous part` и нарушению консистентности реплик
- Исправлена ошибка, приводящая к блокировке при завершении работы сервера, если в это время `.ZooKeeper` недоступен
- .Удалено избыточное логгирование при восстановлении реплик
- .Исправлена ошибка в реализации `UNION ALL`
- Исправлена ошибка в функции `concat`, возникающая в случае, если первый столбец блока имеет тип `.Array`
- .Исправлено отображение прогресса в таблице `system.merges`

# Релиз ClickHouse 1.1.54289, 2017-09-13

## :Новые возможности

- Запросы `SYSTEM` для административных действий с сервером: `SYSTEM RELOAD DICTIONARY`, `SYSTEM RELOAD DICTIONARIES`, `SYSTEM DROP DNS CACHE`, `SYSTEM SHUTDOWN`, `SYSTEM KILL`
- Добавлены функции для работы с массивами: `concat`, `arraySlice`, `arrayPushBack`, `arrayPushFront`, `arrayPopBack`, `.arrayPopFront`
- Добавлены параметры `root` и `identity` для конфигурации ZooKeeper. Это позволяет изолировать разных пользователей одного ZooKeeper кластера
- Добавлены агрегатные функции `groupBitAnd`, `groupBitOr`, `groupBitXor` (для совместимости доступны также `.(под именами BIT_AND, BIT_OR, BIT_XOR`
- .Возможность загрузки внешних словарей из MySQL с указанием сокета на файловой системе
- Возможность загрузки внешних словарей из MySQL через SSL соединение (параметры `ssl_cert`, `ssl_key`, `.(ssl_ca`
- Добавлена настройка `max_network_bandwidth_for_user` для ограничения общего потребления сети для всех запросов одного пользователя
- .Поддержка `DROP TABLE` для временных таблиц
- .Поддержка чтения значений типа `DateTime` в формате unix timestamp из форматов `CSV` и `JSONEachRow`
- Включено по-умолчанию отключение отстающих реплик при распределённых запросах (по-умолчанию `.(порог равен 5 минутам`
- Используются FIFO блокировки при ALTER: выполнение ALTER не будет неограниченно блокироваться `.при непрерывно выполняющихся запросах`
- .Возможность задать `umask` в конфигурационном файле
- .Увеличена производительность запросов с `DISTINCT`

## :Исправления ошибок

- Более оптимальная процедура удаления старых нод в ZooKeeper. Ранее в случае очень частых вставок, старые ноды могли не успевать удаляться, что приводило, в том числе, к очень долгому завершению сервера



- .Исправлена рандомизация при выборе хостов для соединения с ZooKeeper
- Исправлено отключение отстающей реплики при распределённых запросах, если реплика является .localhost
- Исправлена ошибка, в связи с которой кусок данных таблицы типа **ReplicatedMergeTree** мог становиться .битым после выполнения **ALTER MODIFY** элемента **Nested** структуры
- .Исправлена ошибка приводящая к возможному зависанию SELECT запросов
- .Доработки распределённых DDL запросов
- .<Исправлен запрос **CREATE TABLE ... AS <materialized view**
- .Исправлен дедлок при запросе **ALTER ... CLEAR COLUMN IN PARTITION** для **Buffer** таблиц
- Исправлено использование неправильного значения по-умолчанию для **Enum**-ов (0 вместо .минимального) при использовании форматов **JSONEachRow** и **TSKV**
- .Исправлено появление zombie процессов при работе со словарём с источником **executable**
- .Исправлен segfault при запросе HEAD

## :Улучшения процесса разработки и сборки ClickHouse

- .Возможность сборки с помощью **pbuilder**
- .Возможность сборки с использованием **libc++** вместо **libstdc++** под Linux
- .Добавлены инструкции для использования статических анализаторов кода **Coverity**, **clang-tidy**, **cppcheck**

## :На что обратить внимание при обновлении

- Увеличено значение по-умолчанию для настройки MergeTree **max\_bytes\_to\_merge\_at\_max\_space\_in\_pool** (максимальный суммарный размер кусков в байтах для мержа) со 100 GiB до 150 GiB. Это может привести к запуску больших мержей после обновления сервера, что может вызвать повышенную нагрузку на дисковую подсистему. Если же на серверах, где это происходит, количество свободного места менее чем в два раза больше суммарного объёма выполняющихся мержей, то в связи с этим перестанут выполняться какие-либо другие мержи, включая мержи мелких кусков. Это приведёт к тому, что INSERT-ы будут отклоняться с сообщением "Merges are processing significantly slower than inserts". Для наблюдения, используйте запрос **SELECT \* FROM system.merges**. Вы также можете смотреть на метрику **DiskSpaceReservedForMerge** в таблице **system.metrics** или в Graphite. Для исправления этой ситуации можно ничего не делать, так как она нормализуется сама после завершения больших мержей. Если же вас это не устраивает, вы можете вернуть настройку **max\_bytes\_to\_merge\_at\_max\_space\_in\_pool** в старое значение, прописав в config.xml в секции **<merge\_tree>** **<max\_bytes\_to\_merge\_at\_max\_space\_in\_pool>107374182400</max\_bytes\_to\_merge\_at\_max\_space\_in\_pool>** и .перезапустить сервер

## Релиз ClickHouse 1.1.54284, 2017-08-29

- Релиз содержит изменения к предыдущему релизу 1.1.54282, которые исправляют утечку записей о кусках в ZooKeeper

## Релиз ClickHouse 1.1.54282, 2017-08-23

- Релиз содержит исправления к предыдущему релизу 1.1.54276: \* Исправлена ошибка **DB::Exception: Assertion violation: !\_path.empty()** при вставке в Distributed таблицу. \* Исправлен парсинг при вставке в формате RowBinary, если входные данные начинаются с ';'. \* Исправлена ошибка при рантайм-компиляции .(())некоторых агрегатных функций (например, **groupArray**)

## Релиз ClickHouse 1.1.54276, 2017-08-16

### :Новые возможности

- Добавлена опциональная секция WITH запроса SELECT. Пример запроса: **WITH 1+1 AS a SELECT a, a\*a**
- Добавлена возможность синхронной вставки в Distributed таблицу: выдается Ok только после того как все данные записались на все шарды. Активируется настройкой **insert\_distributed\_sync=1**
- Добавлен тип данных UUID для работы с 16-байтовыми идентификаторами
- Добавлены алиасы типов CHAR, FLOAT и т.д. для совместимости с Tableau
- Добавлены функции **toYYYYMM**, **toYYYYMMDD**, **toYYYYMMDDhhmmss** для перевода времени в числа

- Добавлена возможность использовать IP адреса (совместно с hostname) для идентификации сервера при работе с кластерными DDL запросами
- Добавлена поддержка неконстантных аргументов и отрицательных смещений в функции `substring(str, (pos, len`
- Добавлен параметр `max_size` для агрегатной функции `groupBy(max_size)(column)`, и оптимизирована её производительность

## :Основные изменения

- Улучшение безопасности: все файлы сервера создаются с правами 0640 (можно поменять, через `.(параметр в конфиге`
- Улучшены сообщения об ошибках в случае синтаксически неверных запросов
- Значительно уменьшен расход оперативной памяти и улучшена производительность слияний больших MergeTree-кусков данных
- Значительно увеличена производительность слияний данных для движка ReplacingMergeTree
- Улучшена производительность асинхронных вставок из Distributed таблицы за счет объединения нескольких исходных вставок. Функциональность включается настройкой `.distributed_directory_monitor_batch_inserts=1`

## :Обратно несовместимые изменения

- Изменился бинарный формат агрегатных состояний функции `groupBy(array_column)` для массивов

## :Полный список изменений

- Добавлена настройка `output_format_json_quote_denormals`, включающая вывод nan и inf значений в формате JSON
- Более оптимальное выделение потоков при чтении из Distributed таблиц
- Разрешено задавать настройки в режиме readonly, если их значение не изменяется
- Добавлена возможность считывать нецелые гранулы движка MergeTree для выполнения ограничений на размер блока, задаваемый настройкой `preferred_block_size_bytes` - для уменьшения потребления оперативной памяти и увеличения кэш-локальности при обработке запросов из таблиц со столбцами большого размера
- Эффективное использование индекса, содержащего выражения типа `toStartOfHour(x)`, для условий вида `toStartOfHour(x) op constexpr`
- : (Добавлены новые настройки для MergeTree движков (секция `merge_tree` в `config.xml`
- `replicated_deduplication_window_seconds` позволяет задать интервал дедупликации вставок в Replicated-таблицы в секундах
- `cleanup_delay_period` - периодичность запуска очистки неактуальных данных
- `(replicated_can_become_leader` - запретить реплике становиться лидером (и назначать мержи
- Ускорена очистка неактуальных данных из ZooKeeper
- Множественные улучшения и исправления работы кластерных DDL запросов. В частности, добавлена `.настройка distributed_ddl_task_timeout`, ограничивающая время ожидания ответов серверов кластера
- Улучшено отображение стэктрейсов в логах сервера
- Добавлен метод сжатия `none`
- Возможность использования нескольких секций `dictionaries_config` в `config.xml`
- Возможность подключения к MySQL через сокет на файловой системе
- В таблицу `system.parts` добавлен столбец с информацией о размере `marks` в байтах

## :Исправления багов

- Исправлена некорректная работа Distributed таблиц, использующих Merge таблицы, при SELECT с условием на поле `_table`
- Исправлен редкий race condition в ReplicatedMergeTree при проверке кусков данных
- Исправлено возможное зависание процедуры leader election при старте сервера
- Исправлено игнорирование настройки `max_replica_delay_for_distributed_queries` при использовании локальной реплики в качестве источника данных
- Исправлено некорректное поведение `ALTER TABLE CLEAR COLUMN IN PARTITION` при попытке очистить несуществующую колонку

- Исправлено исключение в функции `multif` при использовании пустых массивов или строк
- Исправлено чрезмерное выделение памяти при десериализации формата `Native`
- Исправлено некорректное автообновление `Trie` словарей
- Исправлено исключение при выполнении запросов с `GROUP BY` из `Merge`-таблицы при использовании `SAMPLE`
- Исправлено падение `GROUP BY` при использовании настройки `distributed_aggregation_memory_efficient=1`
- Добавлена возможность указывать `database.table` в правой части `IN` и `JOIN`
- Исправлено использование слишком большого количества потоков при параллельной агрегации
- Исправлена работа функции `if` с аргументами `FixedString`
- Исправлена некорректная работа `SELECT` из `Distributed`-таблицы для шардов с весом 0
- Исправлено падение запроса `CREATE VIEW IF EXISTS`
- Исправлено некорректное поведение при `input_format_skip_unknown_fields=1` в случае отрицательных чисел
- Исправлен бесконечный цикл в функции `dictGetHierarchy()` в случае некоторых некорректных данных словаря
- Исправлены ошибки типа `Syntax error: unexpected (...)` при выполнении распределенных запросов с подзапросами в секции `IN` или `JOIN`, в случае использования совместно с `Merge` таблицами
- Исправлена неправильная интерпретация `SELECT` запроса из таблиц типа `Dictionary`
- Исправлена ошибка "Cannot mmap" при использовании множеств в секциях `IN`, `JOIN`, содержащих более 2 млрд. элементов
- Исправлен `failover` для словарей с источником `MySQL`

## :Улучшения процесса разработки и сборки ClickHouse

- Добавлена возможность сборки в `Arcadia`
- Добавлена возможность сборки с помощью `gcc 7`
- Ускорена параллельная сборка с помощью `ccache+distcc`

## Релиз ClickHouse 1.1.54245, 2017-07-04

### :Новые возможности

- (Распределённые DDL (например, `CREATE TABLE ON CLUSTER`
- Реплицируемый запрос `ALTER TABLE CLEAR COLUMN IN PARTITION`
- (Движок таблиц `Dictionary` (доступ к данным словаря в виде таблицы
- Движок баз данных `Dictionary` (в такой базе автоматически доступны `Dictionary`-таблицы для всех (подключённых внешних словарей
- Возможность проверки необходимости обновления словаря путём отправки запроса в источник
- `Qualified` имена столбцов
- Квотирование идентификаторов двойными кавычками
- Сессии в `HTTP` интерфейсе
- Запрос `OPTIMIZE` для `Replicated` таблицы теперь можно выполнять не только на лидере

### :Обратно несовместимые изменения

- Убрана команда `SET GLOBAL`

### :Мелкие изменения

- Теперь после получения сигнала в лог печатается полный стектрейс
- Ослаблена проверка на количество повреждённых/лишних кусков при старте (было слишком много (ложных срабатываний

### :Исправления багов

- Исправлено залипание плохого соединения при вставке в `Distributed` таблицу
- `GLOBAL IN` теперь работает при запросе из таблицы `Merge`, смотрящей в `Distributed`
- Теперь правильно определяется количество ядер на виртуалках `Google Compute Engine`
- Исправления в работе `executable` источника кэшируемых внешних словарей

- Исправлены сравнения строк, содержащих нулевые символы
- Исправлено сравнение полей первичного ключа типа Float32 с константами
- Раньше неправильная оценка размера поля могла приводить к слишком большим аллокациям
- Исправлено падение при запросе Nullable столбца, добавленного в таблицу ALTER-ом
- Исправлено падение при сортировке по Nullable столбцу, если количество строк меньше LIMIT
- Исправлен ORDER BY подзапроса, состоящего только из константных значений
- Раньше Replicated таблица могла остаться в невалидном состоянии после неудавшегося DROP TABLE
- Алиасы для скалярных подзапросов с пустым результатом теперь не теряются
- Теперь запрос, в котором использовалась компиляция, не завершается ошибкой, если .so файл повреждается

## Fixed in ClickHouse Release 18.12.13, 2018-09-10

### CVE-2018-14672

.Functions for loading CatBoost models allowed path traversal and reading arbitrary files through error messages

Credits: Andrey Krasichkov of Yandex Information Security Team

## Fixed in ClickHouse Release 18.10.3, 2018-08-13

### CVE-2018-14671

unixODBC allowed loading arbitrary shared objects from the file system which led to a Remote Code Execution .vulnerability

Credits: Andrey Krasichkov and Evgeny Sidorov of Yandex Information Security Team

## Fixed in ClickHouse Release 1.1.54388, 2018-06-28

### CVE-2018-14668

remote" table function allowed arbitrary symbols in "user", "password" and "default\_database" fields which led to" .Cross Protocol Request Forgery Attacks

Credits: Andrey Krasichkov of Yandex Information Security Team

## Fixed in ClickHouse Release 1.1.54390, 2018-07-06

### CVE-2018-14669

ClickHouse MySQL client had "LOAD DATA LOCAL INFILE" functionality enabled that allowed a malicious MySQL .database read arbitrary files from the connected ClickHouse server

Credits: Andrey Krasichkov and Evgeny Sidorov of Yandex Information Security Team

## Fixed in ClickHouse Release 1.1.54131, 2017-01-10

### CVE-2018-14670

.Incorrect configuration in deb package could lead to unauthorized use of the database

(Credits: the UK's National Cyber Security Centre (NCSC