



# ClickHouse для инженеров и архитекторов БД

Шардирование и распределенные запросы



**Проверить, идет ли запись**

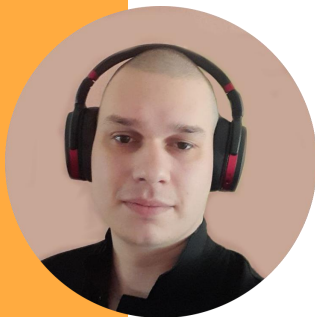
# **Меня хорошо видно && слышно?**



Ставим "+", если все хорошо  
"-", если есть проблемы

Тема вебинара

# Шардирование и распределенные запросы



**Константин Трофимов**

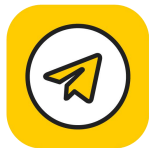
**Senior SRE / ClickHouse DBA** в [VK](#)

Занимаюсь эксплуатацией ClickHouse с первых версий: 5 лет в VK, до этого в AdNow, до этого занимался Vertica. Сотни серверов, десятки кластеров, десятки петабайт данных.

# Правила вебинара



Активно  
участвуем



Off-topic обсуждаем  
в учебной группе  
**#OTUS ClickHouse-2024-04**



Задаем вопрос  
в чат или голосом



Вопросы вижу в чате,  
могу ответить не сразу

## Условные обозначения



Индивидуально



Время, необходимое  
на активность



Пишем в чат



Говорим голосом

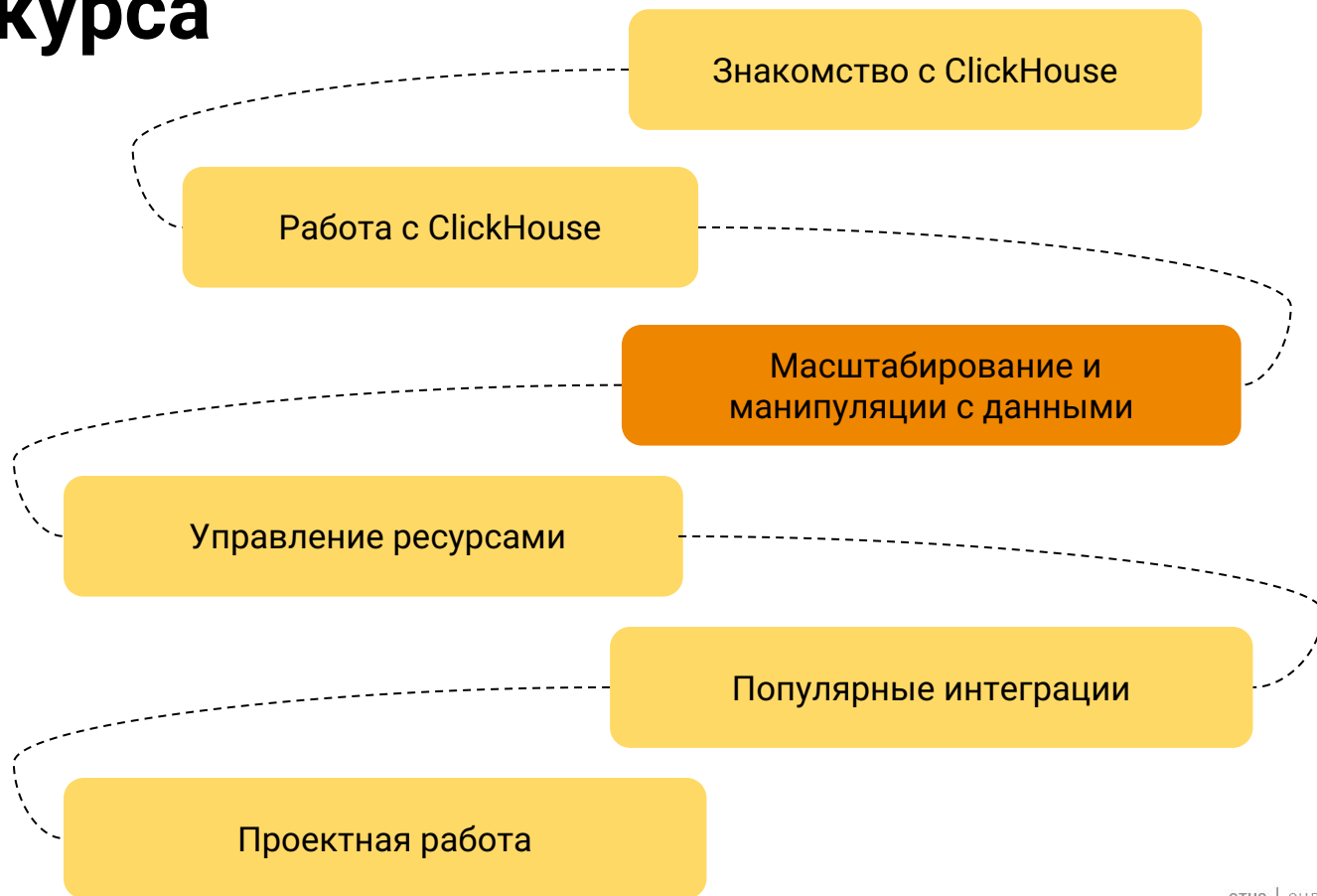


Документ



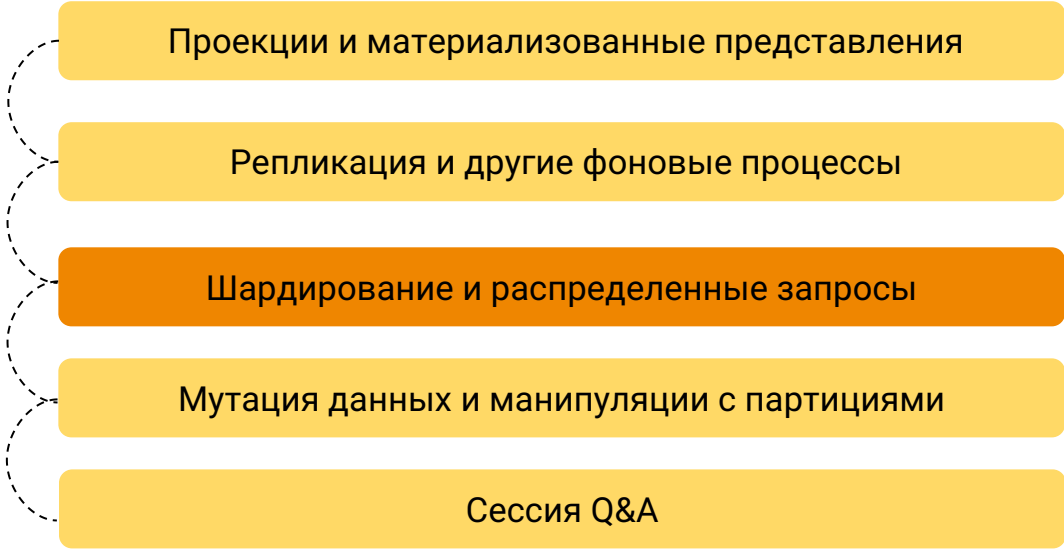
Ответьте себе или  
задайте вопрос

# Карта курса



# Темы модуля

## Масштабирование и манипуляции с данными



Проекции и материализованные представления

Репликация и другие фоновые процессы

Шардирование и распределенные запросы

Мутация данных и манипуляции с партициями

Сессия Q&A

# Маршрут вебинара



Шардирование

Распределенные запросы

Особенности шардирования

Примеры

Рефлексия

# Цели вебинара

К концу занятия вы сможете

1. научиться описывать топологию кластера и создавать Distributed-таблицы
2. заменять отказавшие реплики в кластере



# Смысл

## Зачем вам это уметь

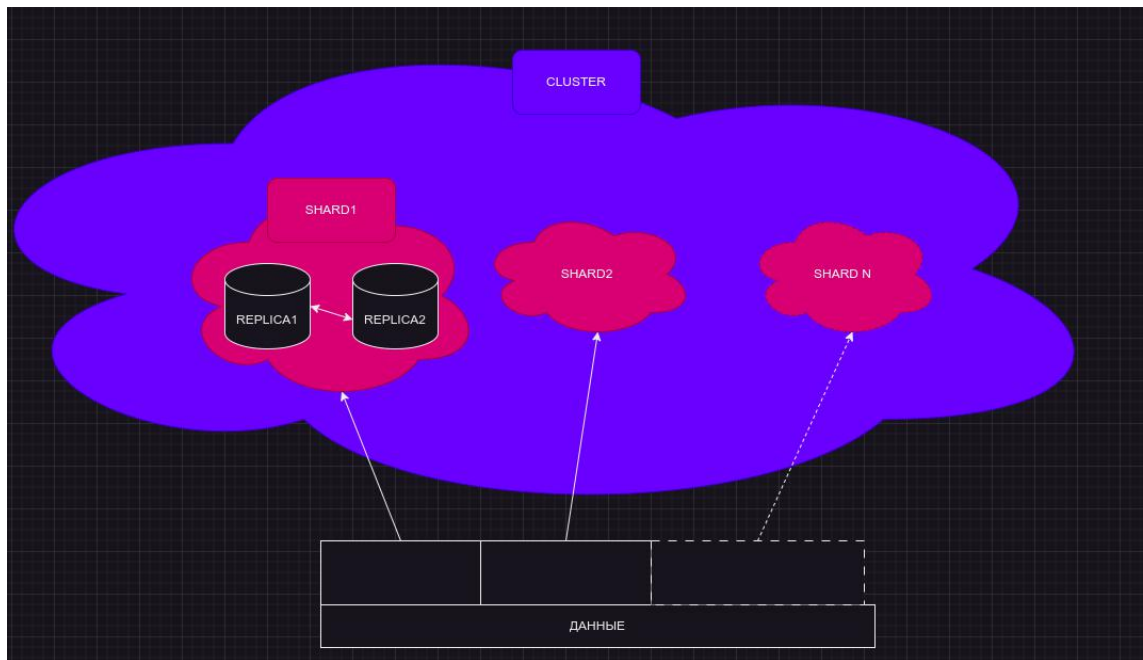
1. горизонтально масштабироваться
  2. для поддержки работоспособности кластера
-

# Шардирование

# Что такое шардирование?

Шардирование - расширение доступного места под данные, за счет расселения данных на несколько экземпляров приложения с независимым хранением.

Как правило за счет размещения экземпляров приложения на разные сервера.



## Engine=Distributed

Виртуальная таблица выступающая в роли «агрегирующего прокси»:

- Принимает запросы
- Повторяет запросы в таблицы с данными согласно топологии кластера
- Объединяет результаты с шардов
- Возвращает результат объединения

Создается:

```
CREATE TABLE имя таблицы ( колонки ) Engine=Distributed( аргументы )
```

## Engine=Distributed аргументы

Distributed( cluster\_name, database, table, sharding\_key )

cluster\_name - топология описанная в конфигурации в секции <remote\_servers> как секция <cluster\_name>, можно выбрать любое имя, можно описывать несколько топологий и использовать с разными Distributed-таблицами

database, table - таблица, в которую будет повторен запрос в одну реплику каждого шарда

sharding\_key - необязательный для SELECT-запросов к таблице параметр, можно не указывать, необходим для INSERT-запросов. Должен быть числовым типом, можно использовать хеширующие функции от других колонок. Нарезает данные на шарды по номеру шарда получаемого как остаток от деления ключа шардирования на количество шардов.

# Engine=Distributed описание топологии кластера

Выполняется через конфигурационный файл:

```
<remote_servers>
  <cluster_name>
    <shard>
      <replica> <host>ch-server-1.zone</host> <port>9000</port> </replica>
      ... replica 2 ...
      ... replica 3 ...
    </shard>
    ... one more shard ...
    ... more shards ...
  </cluster_name>
  ... more clusters ...
</remote_servers>
```

на серверах с Distributed-таблицами

## Доступы

Если Distributed-таблица расположена не на том же сервере (на отдельном), что и таблица с данными, дублировать пользователей на сервера с данными не обязательно.

Distributed-таблица породит запросы в реплики с пользователем default, или с указанным `<user>пользователь</user>` в секции `<replica>...</replica>` конфигурации кластера.

Однако для row-policy всё ещё необходимо наличие пользователя, он передается как `initital_user` на реплики и проверяется по row-policy (подробнее в модуле «Управление ресурсами», лекции «RBAC контроль доступа, квоты и ограничения»).

## Удаление и добавление шардов/реплик

Достигается редактированием конфигурационного файла.

На всех серверах с Distributed-таблицами.

В конфигурационный файл добавляются/удаляются готовые к эксплуатации сервера с нужным набором таблиц, подготовленные заранее.

Важно, чтобы до раскатки конфигурации, были заранее созданы целевые таблицы, к которым будет обращаться Distributed-таблица, ClickHouse не создаст их самостоятельно.

В противном случае будет получен DB::Exception: нет таблицы на remote реплика на запрос к Distributed таблице.



## Замена отказавшей реплики

- 1) в `system.replicas` живой реплики посмотреть имя мертвой реплики и список таблиц
- 2) запрос `SYSTEM DROP REPLICA 'отказавшая реплика' FROM table` таблица, для каждой таблицы
- 3) введение в эксплуатацию новой реплики на замену старой, создание на ней всех таблиц.
- 4) дождаться репликации таблиц
- 5) заменить реплику в `remote_servers`

# Вопросы?



Ставим "+",  
если вопросы есть



Ставим "-",  
если вопросов нет

# Распределенные запросы

# GLOBAL IN/JOIN

Запрос вида

```
SELECT ... FROM distributed_table WHERE IN ( SELECT ... )
```

Будет передан на каждый шард как

```
SELECT ... FROM local_table WHERE IN ( SELECT ... )
```

Где local\_table это таблица заданная в аргументах distributed\_table.

Таким образом, например при выполнении на кластере из 100 шардов, подзапрос ( SELECT ... ) будет выполнен 100 раз.

Модификатор **GLOBAL** меняет это поведение

GLOBAL IN ( подзапрос ) сначала выполнит подзапрос, потом передаст его результат на все шарды как временную таблицу переиспользуемую для основного запроса.

Аналогично работает с JOIN.

## distributed\_product\_mode

Настройка сервера, меняет поведение по умолчанию для IN/JOIN запросов, следующим образом:

- deny - по умолчанию, возвращает DB::EXCEPTION при попытке использовать GLOBAL-запросы

- local - всё ещё запрашивает GLOBAL, но в подзапросах отправляемых на шарды заменяет distributed-таблицы на их local-таблицы

- global - заменяет IN/JOIN на GLOBAL IN/JOIN

- allow - разрешает пользователю выбирать самостоятельно

## **prefer\_global\_in\_and\_join**

при использовании `distributed_product_mode=global` не учитываются таблицы с Engine для доступа к внешним ресурсам, например `Engine=mysql`.

`prefer_global_in_and_join=1` включает такое же поведение для таких Engine.  
`prefer_global_in_and_join=0` по умолчанию, отключено

## использование нескольких реплик одного шарда для запроса

Включается ручкой ***max\_parallel\_replicas>1***

Требует наличия ключа семплирования “SAMPLE BY ключ” на MergeTree таблицах.

Ускоряет выполнения запроса, разбивая его на N реплик, используя ключ семплирования

***SAMPLE 1/N OFFSET (N-1)/N***

### ***Важно!***

Если таблицы не имеют ключа семплирования, будет получен некорректный результат (выборка будет осуществлена по задублированным данным).

# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет



# Особенности шардирования

## Взаимосвязь с репликацией

Взаимосвязи **нет**.

В описании топологии кластера вы можете назначать сервера репликами, а группы серверов шардами, но ClickHouse никак не будет проверять что ваша топология соответствует заданной при создании таблиц репликации.

Набор взаимореплицируемых таблиц шарда, это самостоятельный набор таблиц, никак не зависящий от такого же набора для другого шарда.

Для того, чтобы на разных шардах эти наборы были самостоятельными, удобно использовать макрос `<shard>` в `*keeper`-пути при создании таблицы, для уникализации этого пути для шарда.

## Очередь Distributed таблиц

Под капотом очень не оптимальный недокументированный одноименный формат Distributed, на него нарезается приходящая в Engine=Distributed таблицы вставка. Может быть узким местом и причиной поддержки топологии кластера в клиентах, с собственной реализацией шардирования, записью в обход Distributed-таблиц.

Отслеживать очередь можно в таблице `system.distribution_queue`, важно рисовать метрики на основании этой таблицы.

## Не досхлопывать результаты агрегации с реплик

Некоторые запросы, выполняющие агрегацию по ключу шардирования, при условии что данные действительно шардированы по этому ключу, не требуют дополнительной агрегации.

Можно выставить настройку `distributed_group_by_no_merge=1`, значительно ускорив скорость таких запросов.

Например, `uniq()` в группировке по ключу, при условии что на разных шардах нет повторяющихся между шардами ключей, даст корректный результат даже при пропуске операции дополнительной агрегации.

Если такие повторения между шардами есть, вернется несколько результатов по каждому такому повторению (по результату с шарда).

# Решардинг

Его **НЕТ**.

Предлагаемый в документации способ - создание нового кластера и переливка данных при помощи clickhouse-copier, под капотом которого конструктор "INSERT SELECT" запросов из конфига утилиты.

Альтернативы:

1)

- копирование таблиц как "CREATE TABLE AS"+"ALTER TABLE ATTACH PARTITION FROM"
- дублирование данных по репликации на новые шарды
- спил лишнего через "ALTER TABLE DELETE WHERE"
- переключение на новые таблицы

2) на скриптах DETACH PART/PARTITION, перенос в новое место, ATTACH

3) переливка inplace через INSERT SELECT

## А как шардируются в yandex?

- Много маленьких кластеров используя макрос shard, сверху них distributed таблицы, кластера называются layer.
- Общий кластер используя макрос layer, ещё distributed таблицы сверху distributed таблиц уровня layer.
- в такой концепции становится применима переливка layer-ов утилитой clickhouse-copier

# Вопросы?



Ставим "+",  
если вопросы есть



Ставим "-",  
если вопросов нет

# Примеры



## Distributed и Local таблицы

```
CREATE TABLE default.events (  
  date Date MATERIALIZED toDate(timestamp),  
  ts DateTime,  
  event_id UInt64,  
  host IPv4,  
  response_time_ms UInt32,  
  headers Map(String, String),  
  another_column String,  
  one_more_column Array(String)  
)  
ENGINE=Distributed(cluster_name,default.events  
_local)
```

```
CREATE TABLE default.events_local (  
  date Date MATERIALIZED toDate(timestamp),  
  ts DateTime,  
  event_id UInt64,  
  host IPv4,  
  response_time_ms UInt32,  
  headers Map(String, String),  
  another_column String,  
  one_more_column Array(String)  
)  
ENGINE=ReplicatedMergeTree('/ch/{database}/{table}/{shard}', '{replica}')  
PARTITION BY date  
ORDER BY (date,event_id)  
SAMPLE by event_id
```

**<remote\_servers> 2 шарда 2 реплики, 1 шард 4 реплики, 3 шарда 3 реплики**

**<remote\_servers>**

<!-- 2 shards 2 replicas -->

<2sh2rep>

<shard>

<replica> <host>ch-server-1.zone</host> <port>9000</port> </replica>

<replica> <host>ch-server-2.zone</host> <port>9000</port> </replica>

</shard>

<shard>

<replica> <host>ch-server-3.zone</host> <port>9000</port> </replica>

<replica> <host>ch-server-4.zone</host> <port>9000</port> </replica>

</shard>

</2sh2rep>

```
<!-- 1 shards 4 replicas -->
```

```
<1sh4rep>
```

```
<shard>
```

```
<replica> <host>ch-server-1.zone</host> <port>9000</port> </replica>
```

```
<replica> <host>ch-server-2.zone</host> <port>9000</port> </replica>
```

```
<replica> <host>ch-server-3.zone</host> <port>9000</port> </replica>
```

```
<replica> <host>ch-server-4.zone</host> <port>9000</port> </replica>
```

```
</shard>
```

```
</1sh4rep>
```

```
<!-- 3 shards 3 replicas -->
<3sh3rep>
  <shard>
    <replica> <host>ch-server-1.zone</host> <port>9000</port> </replica>
    <replica> <host>ch-server-2.zone</host> <port>9000</port> </replica>
    <replica> <host>ch-server-3.zone</host> <port>9000</port> </replica>
  </shard>
  <shard>
    <replica> <host>ch-server-4.zone</host> <port>9000</port> </replica>
    <replica> <host>ch-server-5.zone</host> <port>9000</port> </replica>
    <replica> <host>ch-server-6.zone</host> <port>9000</port> </replica>
  </shard>
  <shard>
    <replica> <host>ch-server-7.zone</host> <port>9000</port> </replica>
    <replica> <host>ch-server-8.zone</host> <port>9000</port> </replica>
    <replica> <host>ch-server-9.zone</host> <port>9000</port> </replica>
  </shard>
</3sh3rep>
</remote_servers>
```



# bash-скрипт для создания таблиц как на другой реплике

```
#!/bin/bash
```

```
source_replica=один хост
```

```
dest_replica=другой хост
```

```
for database in $(
```

```
  clickhouse-client --host "${source_replica}" -q "show databases"
```

```
); do
```

```
  clickhouse-client --host "${dest_replica}" -q "create database if not exists ${database}"
```

```
  for table in $(
```

```
    clickhouse-client --host "${source_replica}" -d "${database}" -q "show tables"
```

```
); do
```

```
  table_sql="$(
```

```
    clickhouse-client --host "${source_replica}" -d "${database}" -q "show create table ${table} format TSVRaw"
```

```
  )"
```

```
  clickhouse-client --host "${dest_replica}" -n -d "${database}" <<<"${table_sql}"
```

```
done
```

```
done
```

# Домашнее задание

- 1) запустить N экземпляров clickhouse-server
- 2) описать несколько (2 или более) топологий объединения экземпляров в шарды в конфигурации clickhouse на одном из экземпляров. Фактор репликации и количество шардов можно выбрать на свой вкус.
- 3) предоставить xml-секцию <remote\_servers> для проверки текстовым файлом
- 4) создать DISTRIBUTED-таблицу на каждую из топологий. Можно использовать системную таблицу system.one, содержащую одну колонку dummy типа UInt8, в качестве локальной таблицы.
- или 5) предоставить вывод запроса `SELECT *,hostname(),_shard_num from distributed-table` для каждой distributed-таблицы, можно добавить group by и limit по вкусу если тестовых данных много.
- или 5) предоставить `SELECT * FROM system.clusters; SHOW CREATE TABLE` для каждой Distributed-таблицы.

п.5 можно любой из на ваш выбор из «или», можно оба

# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет



# Рефлексия

# Цели вебинара

## Проверка достижения целей

1. научиться описывать топологию кластера и создавать Distributed-таблицы
2. заменять отказавшие реплики в кластере

# Вопросы для проверки

1. как добавить новый шард в топологию кластера
  2. как заменить отказавшую реплику в кластере
- 
-

# Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то, что узнали на вебинаре?

# Следующий вебинар



27 июня 2024

## Мутация данных и манипуляции с партициями



Ссылка на вебинар  
будет в ЛК за 15 минут



Материалы  
к занятию в ЛК —  
можно изучать



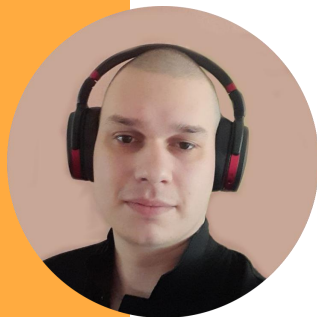
Обязательный  
материал обозначен  
красной лентой



**Заполните, пожалуйста,  
опрос о занятии  
по ссылке в чате**

Спасибо за внимание!

# Приходите на следующие вебинары



**Senior SRE / ClickHouse DBA в [VK](#)**

Занимаюсь эксплуатацией ClickHouse с первых версий: 5 лет в VK, до этого в AdNow, до этого занимался Vertica. Сотни серверов, десятки кластеров, десятки петабайт данных.