



ClickHouse для инженеров и архитекторов БД

Профилирование запросов



Проверить, идет ли запись

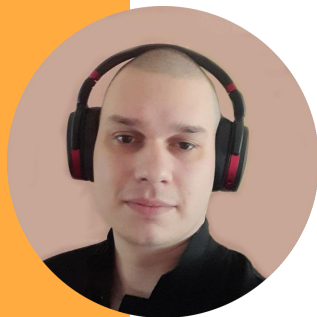
Меня хорошо видно && слышно?



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

Тема вебинара

Метрики и мониторинг, логирование



Константин Трофимов

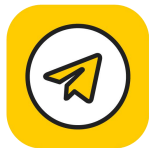
Senior SRE / ClickHouse DBA в [VK](#)

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



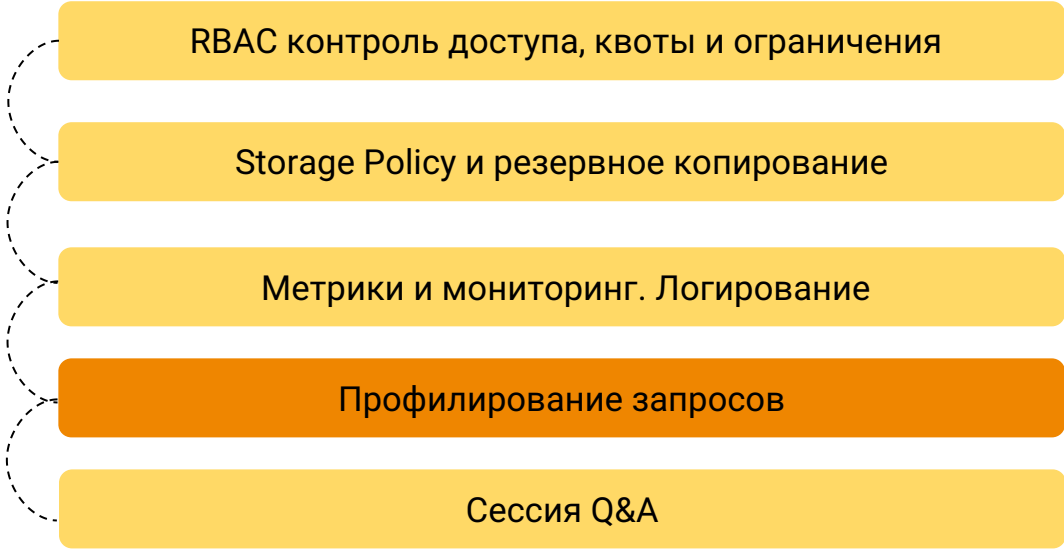
Ответьте себе или
задайте вопрос

Карта курса



Темы модуля

Управление ресурсами



RBAC контроль доступа, квоты и ограничения


Storage Policy и резервное копирование

Метрики и мониторинг. Логирование

Профилирование запросов

Сессия Q&A

Маршрут вебинара



Текстовый лог запроса

Встроенный профайлер

План запроса

Домашнее задание

Рефлексия

Текстовый лог запроса

На примере тестового датасета youtube

text_log сервера, в логе файлом или в логе таблицей, содержит по идентификатору запроса много полезной информации.

При использовании стандартного клиента, можно получить текстовый лог для конкретного запроса прямо в консоль клиента.

в clickhouse-client:

```
SET send_logs_level='trace'; запрос  
... текстовый лог запроса и результаты ...
```

бывает удобно указать ***format Null*** для запроса, если нас не интересует результат, а только причины почему запрос выполняется долго.

На примере тестового датасета youtube

```
3cf83966013 :) SET send_logs_level='trace'; SELECT count() FROM default.youtube WHERE upload_date = '2011-05-06' FORMAT Null;

SET send_logs_level = 'trace'

Query id: 8f87c0d4-baed-4204-ac7d-50f3e655bf44

[3cf83966013] 2024.07.22 13:51:38.645760 [ 48 ] [8f87c0d4-baed-4204-ac7d-50f3e655bf44] <Debug> executeQuery: (from 127.0.0.1:58196) SET send_logs_level='trace'; (stage: Complete)
[3cf83966013] 2024.07.22 13:51:38.646572 [ 48 ] [8f87c0d4-baed-4204-ac7d-50f3e655bf44] <Debug> TCPHandler: Processed in 0.002002540 sec.
Ok.

0 rows in set. Elapsed: 0.003 sec.

SELECT count()
FROM default.youtube
WHERE upload_date = '2011-05-06'
FORMAT 'Null'

Query id: a76382ce-dbb2-45f0-8ce6-f668c59b6a4e

[3cf83966013] 2024.07.22 13:51:38.649646 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> executeQuery: (from 127.0.0.1:58196) SELECT count() FROM default.youtube WHERE upload_date = '2011-05-06' FORMAT Null; (stage: Complete)
[3cf83966013] 2024.07.22 13:51:38.650549 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> InterpreterSelectQuery: The min valid primary key position for moving to the tail of PREWHERE is -1
[3cf83966013] 2024.07.22 13:51:38.650892 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> ContextAccess (default): Access granted: SELECT(upload_date) ON default.youtube
[3cf83966013] 2024.07.22 13:51:38.651022 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> InterpreterSelectQuery: FetchColumns -> Complete
[3cf83966013] 2024.07.22 13:51:38.652269 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> default.youtube (6bbfd70b-e999-4a35-9e86-c935470a2b9c) (SelectExecutor): Key condition: {column 1 in [5100, 5100]}
[3cf83966013] 2024.07.22 13:51:38.653290 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> default.youtube (6bbfd70b-e999-4a35-9e86-c935470a2b9c) (SelectExecutor): Used generic exclusion search over index for part all_1_6_1 with 458 steps
[3cf83966013] 2024.07.22 13:51:38.654389 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> default.youtube (6bbfd70b-e999-4a35-9e86-c935470a2b9c) (SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 393/393 marks by primary key, 393 marks to read from 1 ranges
[3cf83966013] 2024.07.22 13:51:38.654497 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> default.youtube (6bbfd70b-e999-4a35-9e86-c935470a2b9c) (SelectExecutor): Spreading mark ranges among streams (default reading)
[3cf83966013] 2024.07.22 13:51:38.654791 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> MergeTreeReadPool: min_marks_for_concurrent_read=24
[3cf83966013] 2024.07.22 13:51:38.654973 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> default.youtube (6bbfd70b-e999-4a35-9e86-c935470a2b9c) (SelectExecutor): Reading approx. 3204006 rows with 4 streams
[3cf83966013] 2024.07.22 13:51:38.659088 [ 725 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> AggregatingTransform: Aggregating
[3cf83966013] 2024.07.22 13:51:38.659129 [ 725 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: An entry for key=12279238563879453859 found in cache: sum_of_sizes=4, median_size=1
[3cf83966013] 2024.07.22 13:51:38.659178 [ 725 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: Aggregation method: without_key
[3cf83966013] 2024.07.22 13:51:38.659323 [ 717 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> AggregatingTransform: Aggregating
[3cf83966013] 2024.07.22 13:51:38.659493 [ 717 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: An entry for key=12279238563879453859 found in cache: sum_of_sizes=4, median_size=1
[3cf83966013] 2024.07.22 13:51:38.659453 [ 717 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: Aggregation method: without_key
[3cf83966013] 2024.07.22 13:51:38.660559 [ 715 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> AggregatingTransform: Aggregating
[3cf83966013] 2024.07.22 13:51:38.660646 [ 715 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: An entry for key=12279238563879453859 found in cache: sum_of_sizes=4, median_size=1
[3cf83966013] 2024.07.22 13:51:38.660700 [ 715 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: Aggregation method: without_key
[3cf83966013] 2024.07.22 13:51:38.663528 [ 719 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> AggregatingTransform: Aggregating
[3cf83966013] 2024.07.22 13:51:38.663654 [ 719 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: An entry for key=12279238563879453859 found in cache: sum_of_sizes=4, median_size=1
[3cf83966013] 2024.07.22 13:51:38.663782 [ 719 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: Aggregation method: without_key
[3cf83966013] 2024.07.22 13:51:38.664253 [ 725 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> AggregatingTransform: Aggregated: 92 to 1 rows (from 0.00 B) in 0.008845691 sec. (10400.544 rows/sec., 0.00 B/sec.)
[3cf83966013] 2024.07.22 13:51:38.665187 [ 715 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> AggregatingTransform: Aggregated: 44 to 1 rows (from 0.00 B) in 0.009801238 sec. (4489.229 rows/sec., 0.00 B/sec.)
[3cf83966013] 2024.07.22 13:51:38.665595 [ 717 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> AggregatingTransform: Aggregated: 61 to 1 rows (from 0.00 B) in 0.010017430 sec. (6089.361 rows/sec., 0.00 B/sec.)
[3cf83966013] 2024.07.22 13:51:38.666632 [ 719 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: An entry for key=12279238563879453859 found in cache: sum_of_sizes=4, median_size=1
[3cf83966013] 2024.07.22 13:51:38.666696 [ 719 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Trace> Aggregator: Aggregation method: without_key
[3cf83966013] 2024.07.22 13:51:38.668358 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> executeQuery: Read 3204006 rows, 6.11 MiB in 0.018869 sec., 169802639.24956277 rows/sec., 323.87 MiB/sec.
[3cf83966013] 2024.07.22 13:51:38.669744 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> MemoryTracker: Peak memory usage (for query): 636.41 KiB.
[3cf83966013] 2024.07.22 13:51:38.668831 [ 48 ] [a76382ce-dbb2-45f0-8ce6-f668c59b6a4e] <Debug> TCPHandler: Processed in 0.019988677 sec.

0 rows in set. Elapsed: 0.019 sec. Processed 3.20 million rows, 6.41 MB (165.48 million rows/s., 330.95 MB/s.)
Peak memory usage: 636.41 KiB.
```

Так выглядит полный лог уровня trace по запросу с *count()* за конкретный *upload_date*, являющийся основным ключом.



На примере тестового датасета youtube

```
<Debug> executeQuery: (from 127.0.0.1:58196) SELECT count() FROM default.youtube WHERE upload_date = '2011-05-06' FORMAT Null; (stage: Complete)
<Trace> InterpreterSelectQuery: The min valid primary key position for moving to the tail of PREWHERE is -1
<Trace> ContextAccess (default): Access granted: SELECT(upload_date) ON default.youtube
<Trace> InterpreterSelectQuery: FetchColumns -> Complete
```

Начальная стадия запроса:

- принят на исполнение запрос
- оптимизатор запроса сдвинул выражение для основного ключа в PREWHERE
- проверены права доступа
- определили какие будем читать колонки

```
<Debug> default.youtube (6bbfd70b-e999-4a35-9686-c935470a2b9c) (SelectExecutor): Key condition: (column 1 in [15100, 15100])
<Trace> default.youtube (6bbfd70b-e999-4a35-9686-c935470a2b9c) (SelectExecutor): Used generic exclusion search over index for part all_1_6_1 with 458 steps
<Debug> default.youtube (6bbfd70b-e999-4a35-9686-c935470a2b9c) (SelectExecutor): Selected 1/1 parts by partition key, 1 parts by primary key, 393/393 marks by primary key, 393 marks to read from 1 ranges
<Trace> default.youtube (6bbfd70b-e999-4a35-9686-c935470a2b9c) (SelectExecutor): Spreading mark ranges among streams (default reading)
<Debug> MergeTreeReadPool: min_marks_for_concurrent_read=24
<Debug> default.youtube (6bbfd70b-e999-4a35-9686-c935470a2b9c) (SelectExecutor): Reading approx. 3204006 rows with 4 streams
```

Работа с индексом:

- преобразовано выражение из WHERE (PREWHERE) к выражению поиска по индексу
- используем стандартный алгоритм поиска по индексу для единственного подходящего part
- отсеяли один диапазон в 393 засечки
- начали читать в 4 потока выбранные по индексу диапазоны (один найденный диапазон)



На примере тестового датасета youtube

```
<Trace> AggregatingTransform: Aggregating
<Trace> Aggregator: An entry for key=12279238563879453059 found in cache: sum_of_sizes=4, median_size=1
<Trace> Aggregator: Aggregation method: without_key
<Trace> AggregatingTransform: Aggregating
<Trace> Aggregator: An entry for key=12279238563879453059 found in cache: sum_of_sizes=4, median_size=1
<Trace> Aggregator: Aggregation method: without_key
<Trace> AggregatingTransform: Aggregating
<Trace> Aggregator: An entry for key=12279238563879453059 found in cache: sum_of_sizes=4, median_size=1
<Trace> Aggregator: Aggregation method: without_key
<Trace> AggregatingTransform: Aggregating
<Trace> Aggregator: An entry for key=12279238563879453059 found in cache: sum_of_sizes=4, median_size=1
<Trace> Aggregator: Aggregation method: without_key
<Trace> AggregatingTransform: Aggregated. 92 to 1 rows (from 0.00 B) in 0.008845691 sec. (10400.544 rows/sec., 0.00 B/sec.)
<Trace> AggregatingTransform: Aggregated. 44 to 1 rows (from 0.00 B) in 0.009801238 sec. (4489.229 rows/sec., 0.00 B/sec.)
<Trace> AggregatingTransform: Aggregated. 61 to 1 rows (from 0.00 B) in 0.010017439 sec. (6089.381 rows/sec., 0.00 B/sec.)
<Trace> AggregatingTransform: Aggregated. 17 to 1 rows (from 0.00 B) in 0.011240832 sec. (1512.344 rows/sec., 0.00 B/sec.)
<Trace> Aggregator: Merging aggregated data
<Debug> executeQuery: Read 3204006 rows, 6.11 MiB in 0.018869 sec., 169802639.24956277 rows/sec., 323.87 MiB/sec.
<Debug> MemoryTracker: Peak memory usage (for query): 636.41 KiB.
```

Выборка и агрегация данных

- читаем диапазон в 4 потока
- агрегируем полученные в каждом потоке данные
- выводим итоговые счетчики сколько потратили памяти и как быстро выполнили запрос

На что обращать внимание?

!!! метки времени !!!

у каждой строки лога есть метка времени, это позволяет выяснить сколько ClickHouse затратил времени на какую операцию, например, в текущем примере:

1) с 13:51:38.649646 по 13:51:38.651022, т.е. 0.001376 сек мы занимались только интерпретацией запроса.

когда это происходит слишком долго, значит запрос тяжелый для парсера и следует переписать длинные выражения WHERE column IN (.....) в `_external_data` передаваемый с запросом, или использовать временные таблицы.

2) с 13:51:38.651022 по 13:51:38.654973, т.е. 0.003951 сек происходил поиск по индексу, если индексы слишком тяжелые (например вторичные `set(0)`), стоит пересмотреть имеющиеся индексы

3) аналогично со стадией агрегации запроса

!!! использование индексов !!!

обратная история к п.2., когда индексов нет совсем или они не отсеивают достаточно узкие диапазоны строка «Selected 1/1 parts by partition key, 1 parts by primary key, 393/393 marks by primary key, 393 marks to read from 1 ranges» или её отсутствие если индекса нет, позволит выявить фуллскан-обращения к таблице

!!! медленные реплики !!!

в строке лога есть идентификатор хостнейма «[3cfe83966013]» в текущем примере, это хостнейм докера-контейнера. при обращении к Distributed-таблицам, будет приноситься лог с реплик, можно сравнивать метки времени с разных реплик.

Вопросы?



Ставим "+",
если вопросы есть



Ставим "-",
если вопросов нет

Встроенный профайлер

Встроенный профайлер

Представляет собой встроенный сэмплирующий профайлер который внутри ClickHouse снимает трейсы со всех тредов запросов.

Под семплированием понимается запуск профайлера раз в N период времени.

Периоды времени задаются через settings default-пользователя:

```
SET query_profiler_cpu_time_period_ns = 1000000; /* каждые N наносекунд процессорного времени */  
SET query_profiler_real_time_period_ns = 1000000; /* каждые N наносекунд реального времени */  
(+ memory_profiler в новых версиях, см. memory_profiler_% в system.settings)
```

Работа профайлера требует объявления секции **<trace_log>** в конфигурации.

В system.trace_log сохраняются трейсы снимаемые профайлером.

Для сбора и чтения trace_log должен быть установлен **пакет clickhouse-common-static-dbg**

Внимание!

Снятие трейсов - CPU-интенсивная операция; частый профайлинг, даже на дефолтных настройках на нагруженном прод-инстансе, приведет к большому потреблению CPU и как следствие деградации скорости запросов.



Как читать trace_log

1) Включить набор необходимых функций

SET allow_introspection_functions = 1

2) Распарсить колонку trace функциями

`demangle(addressToSymbol(trace))`

для каждого trace в колонке **trace**, (там массив на каждый тред по трейсу), например так:

```
SELECT arrayStringConcat(arrayMap(x -> demangle(addressToSymbol(x)), trace), '\n')
FROM system.trace_log
LIMIT 1
FORMAT Vertical

Query id: 45458b46-13bf-4f63-af43-7e178b048204

Row 1:

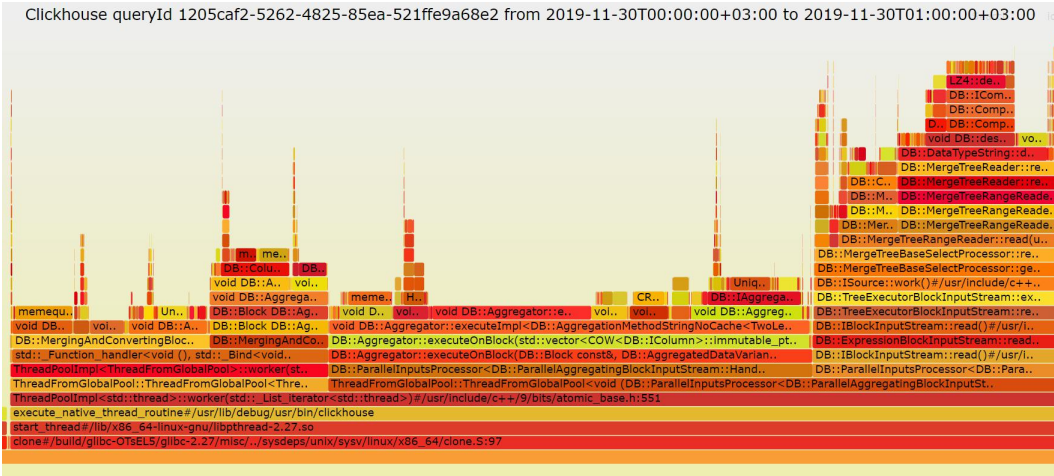
arrayStringConcat(arrayMap(lambda(tuple(x), demangle(addressToSymbol(x))), trace), '\n'): MemoryTracker::allocImpl(long, bool, MemoryTracker*, double)
operator new[](unsigned long)
DB::ThreadStatus::ThreadStatus(bool)
void std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<true>::ThreadFromGlobalPoolImpl<void (DB::TraceCollector::*)(), DB::TraceCollector*>(void (DB::TraceCollector::*&&)(), DB::TraceCollector*&&):'lambda'(), void ()>>(std::__1::__function::__policy_storage const*)
void* std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void ThreadPoolImpl<std::__1::thread>::scheduleImpl<void>(std::__1::function<void ()>, Priority, std::__1::optional<unsigned long>, bool):'lambda0'()>>(void*)
```

Что делать с трейсами дальше?

- 1) Это семплирующий профайлер, если выставить достаточно часто, можно в пределах одного запроса выявить наиболее часто встречающиеся в трейсе строки, это и будет самым долгим местом выполнения запроса.
- 2) То же самое по группе однотипных запросов.
- 3) Можно матчить query_id и thread_id на query_log и thread_log
- 4) Можно использовать сторонние инструменты для анализа трейса, например в виде flamegraph:

<https://github.com/Slach/clickhouse-flamegraph>

или визуализировать самостоятельно
в любом удобном формате
(личное предпочтение преподавателя:
«cat собранное-из-лога|sort|uniq -c» в bash)



Вопросы?



Ставим "+",
если вопросы есть



Ставим "-",
если вопросов нет

План запроса

План запроса

Синтаксис

EXPLAIN [AST | SYNTAX | PLAN | PIPELINE] [setting = value, ...] SELECT ... [FORMAT ...]

Пример:

```
EXPLAIN
SELECT
    upload_date,
    uniq(uploader_id),
    argMax(view_count, fetch_date)
FROM youtube
WHERE (upload_date >= '2020-12-20') AND (upload_date <= '2020-12-27') AND (uploader_id NOT IN ('UC9Fjgb3aZvr-vfyecs3AutA'))
GROUP BY upload_date
HAVING argMax(view_count, fetch_date) > 100
ORDER BY upload_date ASC
```

Query id: 529ad486-4723-40d2-8447-5e01faf0be1f

```
explain
Expression (Projection)
  Sorting (Sorting for ORDER BY)
    Expression (Before ORDER BY)
      Filter (HAVING)
        Aggregating
          Expression (Before GROUP BY)
            ReadFromMergeTree (default.youtube)
```



EXPLAIN AST

EXPLAIN возвращает разбор синтаксиса запроса на выполняемые действия (Abstract Syntax Tree)

EXPLAIN SYNTAX

Обрабатывает запрос встроенным оптимизатором ClickHouse. Самый частый кейс - сдвигает выражение по ПК в PREWHERE.

EXPLAIN ESTIMATE

Показывает сколько будет прочитано строк, гранул, партов. К сожалению не показывает ожидаемое время.

EXPLAIN PLAN

То же, что и AST без указания аргументов. Через аргументы можно достать информацию об использовании индексов

```
EXPLAIN indexes = 1
SELECT
  upload_date,
  uniq(uploader_id),
  argMax(view_count, fetch_date)
FROM youtube
WHERE (upload_date >= '2020-12-20') AND (upload_date <= '2020-12-27') AND (uploader_id NOT IN ('UC9Fjgb3aZvr-vfyecs3AutA'))
GROUP BY upload_date
HAVING argMax(view_count, fetch_date) > 100
ORDER BY upload_date ASC
```

Query id: 7ca9b885-2dc2-49df-addd-7150f06f83f2

```
explain
Expression (Projection)
  Sorting (Sorting for ORDER BY)
    Expression (Before ORDER BY)
      Filter (HAVING)
        Aggregating
          Expression (Before GROUP BY)
            ReadFromMergeTree (default.youtube)
              Indexes:
                PrimaryKey
                  Keys:
                    upload_date
                  Condition: and((upload_date in (-Inf, 18623]), (upload_date in [18616, +Inf]))
                  Parts: 1/1
                  Granules: 393/393
```

14 rows in set. Elapsed: 0.002 sec.



Вопросы?



Ставим “+”,
если вопросы есть



Ставим “-”,
если вопросов нет

Домашнее задание

Домашнее задание

- 1) Выполнить запрос с WHERE не использующим ПК. Выполнить запрос с WHERE использующим ПК. Сравнить text_log запросов, предоставить строки лога относящиеся к пробегу основного индекса.
- 2) Показать тот же индекс через EXPLAIN

Вопросы?



Ставим "+",
если вопросы есть



Ставим "-",
если вопросов нет

Рефлексия

Рефлексия



С какими впечатлениями уходите с вебинара?



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

Следующий вебинар



25 июля 2024

Сессия Q&A



Ссылка на вебинар
будет в ЛК за 15 минут



Материалы
к занятию в ЛК —
можно изучать



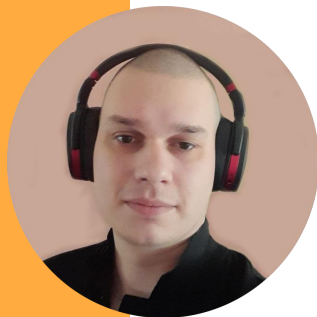
Обязательный
материал обозначен
красной лентой



**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.