Стратегии оптимизации запросов и доступа к данным в ClickHouse

ClickHouse Meetup / Limassol, Cyprus

Я :)

- Не являюсь системным администратором
- Не являюсь администратором баз данных
- Я разработчик, восовном пишу на Go

О чем ?

Об опыте работы с ClickHouse

Что за синий квадрат?

Integros video platform

https://integros.com

ClickHouse не тормозит ©

Но это не точно.

~~ClickHouse не тормозит ©~~

Нет такой системы, которую в умелых руках нельзя сломать или испортить!

Традиционно

Что влияет на скорость выполнения запросов?

- CPU
- Память
- Сеть
- "RAID с батарейкой"
- "Хороший RAID с батарейкой"
- Попутный ветер

Это все не важно

- чем больше - тем лучше
- чем лучше - тем лучше

* А ещё я не системный администратор ;)

Менее традиционно

Если хочется улучшить - нужно переписывать код :)

- https://github.com/littledan/linux-aio
- https://blog.cloudflare.com/io_submit-the-epoll-alternative-youve-never-heard-about/
- https://blog.cloudflare.com/sockmap-tcp-splicing-of-the-future/
- https://github.com/search?l=C%2B%2B&q=fast&type=Repositories
- https://github.com/search?l=C%2B%2B&q=High-performance&type=Repositories

Важно, но сложно!

Есть теория

“Если какие-то вычисления занимают X секунд, то для того, чтобы нам потребовалось X/2 секунд, нужно вычислять X/2.”

- теория и практика эффективного менеджмента Т.П. Барсук. МСК 2003.

“Существует мнение, что для того, чтобы эффективно считать статистику, данные нужно агрегировать, так как это позволяет уменьшить объём данных.”

- выдержка из документации к ClickHouse

И это правда :)

"Сырые" события

В Facebook на этот митап "собирались пойти" или "интересовались" 250 человек.

Представим, что каждый раз когда человек нажимал на кнопку FB отправлял нам данные:

- тип ("пойду" или "интересно")
- время с точностью до секунды
- имя
- пол
- день рождения
- место рождения
- место работы
- должность

Мы записали данные как есть - это и будут "сырые" данные.

DDL ;)

```
CREATE TABLE exness_events (

      event_time    DateTime

    , event_type    Enum16('going' = 1, 'interested' = 2)

    , name          String

    , gender        Enum16('male'  = 1, 'female'     = 2)

    , date_of_birth Date

    , birthplace    String

    , company       String

    , position      String

) Engine MergeTree PARTITION BY toYYYYMM(event_time) ORDER BY (event_type, event_time);
```

Зачем вообще заранее агрегировать?

Чтоб не выполнять расчеты по всем данным каждый раз ;)

shut up and take my money!

Как агрегировать?

- Фоновый процесс
- MATERIALIZED VIEW

Materialized view

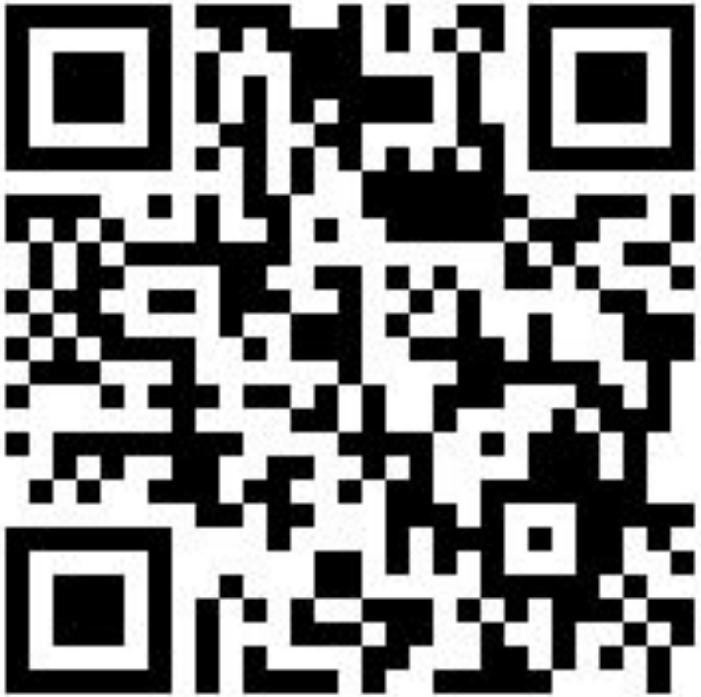
```
CREATE TABLE exness_events_agg (

    event_type    Enum16('going' = 1, 'interested' = 2)

    , total       Int16

) Engine SummingMergeTree PARTITION BY event_type ORDER BY (event_type);



-- создадим триггер который будет вызываться при каждой записи в exness_events и писать в
exness_events_agg

CREATE MATERIALIZED VIEW exness_events_mv TO exness_events_agg AS

    SELECT event_type, toInt16(1) AS total FROM exness_events;
```

Результат

```
SELECT

    event_type

    , SUM(total) AS total

FROM exness_events_agg

GROUP BY event_type

┌─event_type─┬─total─┐

│ going      │     4 │

│ interested │     1 │

└────────────┴───────┘
```

ClickHouse в фоне производит агрегацию в `SummingMergeTree` (вся "магия" MergeTree происходит во время слияний), поэтому мы не можем быть уверенными в том, что в момент времени T все данные уже были смержены. Для этого мы делаем финальную агрегацию в запросе.

Меньше слов - больше кода ;)



Если хочется большего - используй массивы

- Отличная поддержка на стороне ClickHouse
  - Nested + *Map
  - sumMap
  - ARRAY JOIN
  - -Array
- Но есть нюансы
  - index_granularity
  - GROUP BY и память
  - Нужны словари (sumMap не работает со строками)

Выводы

- Каждый решает сам что ему делать со своими данными
- Если вам “повезло”, то вам достаточно просто создать материализованное представление :)

Читайте документацию

Спрашивайте в https://t.me/clickhouse_ru

Часть вторая, когда просто пописать SQL может быть недостаточно.

Исходно: у нас есть сырые данные

```
CREATE TABLE video_events (

    event_time DateTime

    , user_id  Int32

    , video_id Int64

    , bytes    Int64

    , os       LowCardinality(String)

    , device   LowCardinality(String)

    , browser  LowCardinality(String)

    , country  LowCardinality(String)

    , domain   LowCardinality(String)

) Engine MergeTree PARTITION BY toYYYYMM(event_time) ORDER BY (event_time, user_id, video_id);
```

Задача

- Строить произвольные отчеты
- Показывать последние события (фильтрация произвольная)

В чем проблема?

1 000 000 000 событий в день

Агрегировать

```
CREATE TABLE video_events_agg (

    ...

    , bytes_total Int64

    , count       Int64

    , min_time    AggregateFunction(min, DateTime)

    , max_time    AggregateFunction(max, DateTime)

    , timeMap     Nested (

        hour    UInt8

        , count Int64

    )

) Engine SummingMergeTree PRIMARY KEY (date, user_id, video_id) ORDER BY (...)
```

В чем проблема?

Произвольные отчеты строятся, но не все одинаково быстро

```
SELECT count() FROM video_events_agg WHERE user_id = 7

┌─count()─┐
│  629508 │
└─────────┘

1 rows in set. Elapsed: 0.011 sec. Processed 1.12 million rows, 4.48 MB
(103.74 million rows/s., 414.95 MB/s.)

SELECT count() FROM video_events_agg WHERE video_id = 7000

┌─count()─┐
│     633 │
└─────────┘

1 rows in set. Elapsed: 0.033 sec. Processed 12.44 million rows, 99.54 MB
(371.37 million rows/s., 2.97 GB/s.)
```

В чем проблема №2 ?

```
SELECT * FROM video_events WHERE (video_id = 7000) AND (user_id = 7) ORDER BY event_time DESC LIMIT 10
```

```
┌──────────event_time─┬─user_id─┬─video_id─┬──────bytes─┬─os──────┬─device───┬─browser─┬─country─┬─domain──────────┐
│ 2019-05-05 19:07:23 │       7 │     7000 │ 2947257000 │ OS X    │ Tablet   │ Chrome  │ UA      │ exness.com      │
│ 2019-05-05 18:42:43 │       7 │     7000 │ 2015503000 │ Windows │ Desktop  │ Firefox │ CY      │ clickhouse.yandex │
│ 2019-05-05 18:38:30 │       7 │     7000 │  950649000 │ OS X    │ Tablet   │ Chrome  │ CY      │ exness.com      │
└─────────────────────┴─────────┴──────────┴────────────┴─────────┴──────────┴─────────┴─────────┴─────────────────┘
```

```
10 rows in set. Elapsed: 3.327 sec. Processed 341.07 million rows, 1.54 GB (102.52 million rows/s., 463.40 MB/s.)
```

Пойдем в Postgres Pro

Планировщик/оптимизатор

https://postgrespro.ru/docs/postgrespro/11/planner-optimizer

Спасибо ребятам из Postgres Professional за их работу

Идеи хорошие

- Но ClickHouse не умеет выбирать планы выполнения
- В ClickHouse   нет индексов (не в каждом ClickHouse есть индексы;))

Мы инженеры - давайте сами все сделаем

Scientia potentia est

- Никто не знает ваши данные лучше вас
- Если вы знаете о связях - вы можете оптимизировать запросы

```
SELECT count()
FROM video_events_agg
WHERE (video_id = 7000) AND (user_id = 7)

┌─count()─┐
│     633 │
└─────────┘

1 rows in set. Elapsed: 0.023 sec. Processed 506.20 thousand rows, 3.13 MB
(22.27 million rows/s., 137.55 MB/s.)
```

Экспериментальные возможности

```
SET allow_experimental_data_skipping_indices = 1;

ALTER TABLE video_events_agg ADD INDEX idx_video_id video_id TYPE minmax GRANULARITY 3;

SELECT count() FROM video_events_agg WHERE video_id = 7000

┌─count()─┐
│     633 │
└─────────┘

1 rows in set. Elapsed: 0.009 sec. Processed 663.55 thousand rows, 5.31 MB (75.90 million rows/s., 607.20 MB/s.)
```

Проблема №2

“ClickHouse очень эффективно сортирует, но для этого он сканирует все данные”

Алексей Миловидов :)

У нас есть статистика!

- Мы знаем распределение данных по времени
- Мы можем написать запрос для извлечения нужных диапазонов дат
- Я не смог запихнуть этот SQL запрос в экран (см GitHub)

```
SELECT toDateTime(date) + INTERVAL hour HOUR AS from, max_time AS to,
runningAccumulate(state) AS count FROM (...) WHERE count > 10 LIMIT 1

┌────────────────from─┬──────────────────to─┬─count─┐
│ 2019-05-05 18:00:00 │ 2019-05-05 19:07:23 │    12 │
└─────────────────────┴─────────────────────┴───────┘

1 rows in set. Elapsed: 0.051 sec. Processed 983.04 thousand rows, 13.33
MB (19.41 million rows/s., 263.27 MB/s.)
```

“Оптимизируем” исходный запрос

```
SELECT * FROM video_events WHERE (video_id = 7000) AND (user_id = 7)
AND event_time >= '2019-05-05 18:00:00' AND event_time <= '2019-05-05 19:07:23'
ORDER BY event_time DESC LIMIT 10
```

```
┌──────────event_time─┬─user_id─┬─video_id─┬──────bytes─┬─os──────┬─device──┬─browser─┬─country─┬─domain──────────┐
│ 2019-05-05 19:07:23 │       7 │     7000 │ 2947257000 │ OS X    │ Tablet  │ Chrome  │ UA      │ exness.com      │
│ 2019-05-05 18:42:43 │       7 │     7000 │ 2015503000 │ Windows │ Desktop │ Firefox │ CY      │ clickhouse.yandex │
│ 2019-05-05 18:38:30 │       7 │     7000 │  950649000 │ OS X    │ Tablet  │ Chrome  │ CY      │ exness.com      │
└─────────────────────┴─────────┴──────────┴────────────┴─────────┴─────────┴─────────┴─────────┴─────────────────┘
```

```
10 rows in set. Elapsed: 0.039 sec. Processed 409.60 thousand rows, 1.85 MB (10.55 million rows/s., 47.72 MB/s.)
```

Итого

0.051 sec для определения "стоимости" запроса и 0.039 sec на выполнение.
0.09 vs 3.327 sec

Выводы

У каждого свои

Читать

- MySQL. Оптимизация производительности (Зайцев Петр, Шварц Бэрон)
- Oracle. Оптимизация производительности (Кэри Миллсап, Джефф Хольт)
- SQL и реляционная теория. Как грамотно (писать код на SQL К. Дж. Дейт)

Спасибо

- вам
- https://github.com/yandex/ClickHouse/graphs/contributors
- Exness :)

Помощь по ClickHouse

- https://t.me/clickhouse_ru
- https://altinity.com