5. Другие способы оптимизации запросов

Фильтрация с помощью IN

В некоторых случаях запрос можно сильно ускорить, если предварительно отфильтровать данные с помощью запроса IN.

Рассмотрим пример:

Пример: запрос на вывод данных за день. Улучшение в 400 раз.

```
SELECT count()
FROM analytics.new_app_funnel_table
INNER JOIN analytics.int_spree_orders ON new_app_funnel_table.context_order_id = int_spree_orders.number
--Elapsed: 2259.573 sec.


-- select *   !
WITH data_from_orders AS (
    SELECT number
    FROM analytics.int_spree_orders
    WHERE number IN (
        SELECT context_order_id
        FROM analytics.new_app_funnel_table
    )
)
SELECT count()
FROM analytics.new_app_funnel_table
INNER JOIN data_from_orders ON new_app_funnel_table.context_order_id = data_from_orders.number
--Elapsed: 725.149 sec.
```

Второй запрос отработал в 3 раза быстрее, т.к. перед джойном мы предварительно отфильтровали данные.

Такая оптимизация стала возможной в данном случае, т.к. в *analytics.new_app_funnel_table* всего 56 млн. уникальных ключей, а в *analytics.int_spree_orders* 223 млн. уникальных ключей, т.е. если мы предварительно отфильтруем, то нам не нужно будет читать/обрабатывать 75% данных из *analytics.int_spree_orders*.

Проверка требований для использования IN

```
SELECT countDistinct(context_order_id)
FROM analytics.new_app_funnel_table

uniqExact(context_order_id)
                  56.198.397


SELECT countDistinct(number)
FROM analytics.int_spree_orders

uniqExact(number)
       223.075.186


--,  analytics.int_spree_orders  ,     (223 . > 56 .).
--     .

SELECT countDistinct(number)
FROM analytics.int_spree_orders
WHERE number IN (
    SELECT context_order_id
    FROM analytics.new_app_funnel_table
)

uniqExact(number)
        53.562.059
```

Когда имел смысл попробовать данную оптимизацию

Данную оптимизацию следует попробовать применить, если пересечение таблиц по ключу джойна небольшое (например, меньше половины от таблицы).

Если же две таблицы почти полностью пересекаются по ключу джоина, то отфильтруется мало данных и запрос будет работать наоборот медленнее. В любом случае всегда лучше попробовать запустить оба варианта (с фильтрацией и без) и сравнить результаты производительности.

Кроме того, нужно быть аккуратным и не засовывать в *IN* слишком много данных, т.к. построение множества ключей достаточно дорогая операции как по процессорному времени, так и по используемой оперативной памяти. У нас был случай, когда при попытке засунуть большое количество данных в *IN* умирал сервер клика.

Использовать *distinct* при выборе уникальных ключей для фильтра не нужно, т.к. при построении множества для IN клик и так оставляет только уникальные ключи.

Код

```
--
SELECT countDistinct(number)
FROM analytics.int_spree_orders
WHERE number IN (
    SELECT DISTINCT context_order_id
    FROM analytics.new_app_funnel_table
)

--
SELECT countDistinct(number)
FROM analytics.int_spree_orders
WHERE number IN (
    SELECT context_order_id
    FROM analytics.new_app_funnel_table
)
```

Регулирование max_threads

Если в вашем запросе много группировок/джойнов, которые съедают много оперативной памяти или просто запрос ест много памяти, можно попробовать уменьшить для запроса настройку ***max_threads.***

Данная настройка отвечает на каком максимальном количестве потоков (по сути процессоров) будет работать ваш запрос.

По умолчанию ***max_threads=32***. Если уменьшить количество потоков, тогда запрос будет есть меньше памяти, но немного дольше работать (время расчета может увеличиться не сильно).

Переопределяется в секции SETTINGS в конце запроса. **Больше 32 ставить запрещено!**

Пример переопределения:

Код

```
SELECT ...
FROM ...
SETTINGS max_threads=4
```

Разбиение запроса на несколько частей по исходным данным

В некоторых случая, если запрос потребляет много памяти, можно попробовать разбить весь расчет на несколько частей по данным.

Примеры:

- рассчитать по отдельности требуемые периоды (например, вместо 3-х месяцев разом считаем отдельно каждый месяц).
- разбить исходные датасеты по остатку от деления от hash-функции набора колонок.

Общий алгоритм следующий:

1. Разбиваем исходные данные по некоторому признаку(-ам) на несколько частей.
2. Рассчитываем каждую часть по отдельности, результаты складываем во временную stage-таблицу.
3. Последним шагом объединяем все части вместе.

Рассмотрим пример подсчета уникальных пар offer_id/uuid за некоторый период:

Код

```
SELECT
    offer_id,
    uuid,
    toDate(created_at) AS dt,
    count()
FROM analytics.int_spree_line_items
WHERE created_at >= toDate('2023-10-01') AND created_at < toDate('2023-12-01')
GROUP BY
    offer_id,
    uuid,
    dt
```

Данный запрос отработал за 95 секунд, используя при этом 66ГБ оперативной памяти.

Попробуем разбить исходный запрос на 4 части по остатку от деления *cityHash64(offer_id)* на 4, т.е. отдельно посчитаем запросы:

1. cityHash64(offer_id) % 4 = 0
2. cityHash64(offer_id) % 4 = 1
3. cityHash64(offer_id) % 4 = 2
4. cityHash64(offer_id) % 4 = 3

Код

```
SELECT
    offer_id,
    uuid,
    toDate(created_at) AS dt,
    count()
FROM analytics.int_spree_line_items
WHERE created_at >= toDate('2023-10-01') AND created_at < toDate('2023-12-01') AND cityHash64(offer_id) % 4 = 0
GROUP BY
    offer_id,
    uuid,
    dt
```

Первый из запросов работал 50 секунд, остальные по 18 секунд, т.е. сумарное время выполнения – 104 секунды. Потребление оперативной памяти составило 25ГБ.

Резюмируем: разбив запрос на 4 части, мы в 2.5 раза сократили потребление оперативной памяти, но проиграли 9 секунд во времени выполнения (было 95, стало 104), что в данном случае совсем некритично.