

# 1. Базовые правила написания запросов

- Не используем `select *`
- Все таблицы должны быть определены вместе с БД
- Фильтруем данные
- Используем CTE
  - Пример использования cte
  - Как работает CTE и как переиспользовать CTE
    - Что делать если CTE сложная, но ее нужно переиспользовать в нескольких других CTE?
- Именование столбцов в `SELECT` и `WHERE`
  - Пример того, как можно выстрелить себе в ногу



## Главное правило

Выбор только нужных колонок и правильная фильтрация данных составляют большую часть оптимального выполнения запроса!

P.S. В кликхаусе очень простой оптимизатор – в большинстве случаев как написано, так он и выполнится.

## Не используем `select *`

В запросах/cte/подзапросах не используем `select *`, всегда перечисляем все используемые поля.

Clickhouse колоночная БД (данные каждой колонке хранятся отдельно друг от друга), т.е. если вы выбираете все колонки из таблицы, это сильно нагружает БД (распаковка данных). Также, например, при джойнах оптимизатор может не догадаться, что используются не все колонки и поднять все в память, что сильно сократит объем свободной памяти и нагрузит процессоры лишней работой. Кроме того, очень сложно понять какие колонки (и откуда) используются в запросе, когда читаешь чужой код.

### Код

```
--
SELECT * FROM analytics.line_items LIMIT 10

--
SELECT line_item_id, li_created_at FROM analytics.line_items LIMIT 10
```

## Все таблицы должны быть определены вместе с БД

По дефолту в большинстве случаев в коннектах к клику зашита БД analytics, но баз становится все больше и не всегда очевидно откуда таблица. Добавление БД к имени таблицы облегчает понимание кода, возможный рефакторинг и поиск в коде, где используется таблица.

### Код

```
--
SELECT line_item_id, li_created_at FROM line_items LIMIT 10

--
SELECT line_item_id, li_created_at FROM analytics.line_items LIMIT 10
```

## Фильтруем данные

Максимально возможно фильтруем все исходные данные.

Особенно важно использовать при фильтрации колонки, которые участвуют в партицировании таблицы.

Партицирование таблицы можно посмотреть с помощью запроса `SHOW CREATE TABLE`. Партицирование будет определено после выражения `PARTITION BY`.

В примере ниже, в партицировании участвует одна колонка `li_created_at`.

## Код

```
SHOW CREATE TABLE analytics.line_items;

-- :
CREATE TABLE analytics.line_items
(
  `line_item_id` UInt32,
  ...
  `order_number` String
)
ENGINE = ReplicatedMergeTree('/tables/{shard}/analytics/line_items', '{replica}')
PARTITION BY toYYYYMM(li_created_at)
PRIMARY KEY tuple()
ORDER BY (retailer_id, sku)
SETTINGS index_granularity = 8192
```

## Используем CTE

**Во всех запросах, где используется больше одной таблицы, используем CTE:** чтение сырых данных делаем с **обязательным** выбором по каждой таблице нужных колонок и фильтрацией самих данных.

Название cte должны отражать смысл содержащихся в них данных и ни в коем случае не должны совпадать один в один с названиями таблицы.

Имена CTE должны начинаться с приставки *cte\_*. Например, *cte\_line\_items*, *cte\_new\_app\_grouped\_by\_user*.

Плюсы данного решения:

- повышается читаемость и понятность кода.
- уменьшаем возможные ошибки оптимизатора при фильтрации данных (например, фильтрация может произойти после джойна, а не до, как вы хотели) и выборе только необходимых колонок.

## Пример использования cte

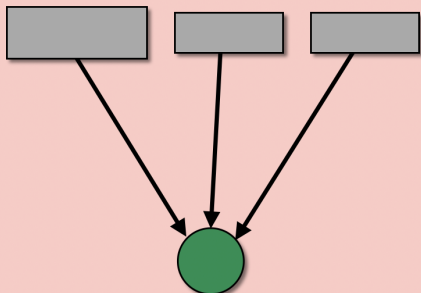
Рассмотрим пример джойна трех таблиц. Для примера предположим, что таблица *table\_a* большая, а таблицы *table\_b* и *table\_c* меньше.

Правильно в данном случае будет сделать по шагам так:

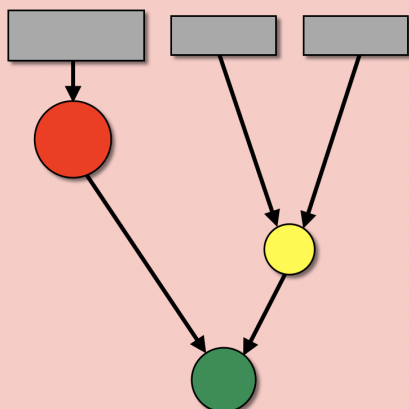
1. фильтруем все сырые данные и выбираем необходимые столбцы в cte'шках
2. джойним данные из двух меньших таблиц
3. результат джойна из предыдущего шага джойним с данными из большой таблицы

- таблица с сырыми данными
- cte с фильтрованными данными из таблицы
- промежуточная cte
- результат

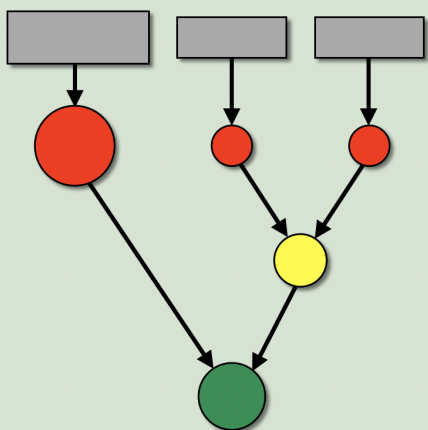
### Неправильно #1



### Неправильно #2



Правильно



Пример кода:

Код

```

-- #1
SELECT
    [ ]
FROM sandbox.table_a
INNER JOIN sandbox.table_b ON table_a.id = table_b.id
INNER JOIN sandbox.table_c ON table_a.id = table_c.id
WHERE 1 = 1
    AND [ sandbox.table_a]
    AND [ sandbox.table_b]
    AND [ sandbox.table_c]

-- #2.
WITH
    cte_data_from_table_a AS
    (
        SELECT
            [ ]
        FROM sandbox.table_a
        WHERE [ sandbox.table_a]
    ),
    cte_table_b_join_table_c AS
    (
        SELECT
            [ ]
        FROM sandbox.table_b
        INNER JOIN sandbox.table_c ON table_b.id = table_c.id
        WHERE [ sandbox.table_b] AND [ sandbox.table_c] AND [ ]
    )
SELECT
    [ '_cte._']
FROM cte_data_from_table_a
INNER JOIN cte_table_b_join_table_c ON cte_data_from_table_a.id = cte_table_b_join_table_c.id

--
WITH
    cte_data_from_table_a AS
    (
        SELECT
            [ ]
        FROM sandbox.table_a
        WHERE [ sandbox.table_a]
    ),
    cte_data_from_table_b AS
    (
        SELECT
            [ ]
        FROM sandbox.table_b
        WHERE [ sandbox.table_b]
    ),
    cte_data_from_table_c AS
    (
        SELECT
            [ ]
        FROM sandbox.table_c
        WHERE [ sandbox.table_c]
    ),
    cte_table_b_join_table_c AS
    (
        SELECT
            [ '_cte._']
        FROM cte_data_from_table_b
        INNER JOIN cte_data_from_table_c ON cte_data_from_table_b.id = cte_data_from_table_c.id
        WHERE [ ( )]
    )
SELECT
    [ '_cte._']
FROM cte_data_from_table_a
INNER JOIN cte_table_b_join_table_c ON cte_data_from_table_a.id = cte_table_b_join_table_c.id

```

## Как работает CTE и как переиспользовать CTE

На самом деле cte это alias для подзапросов, когда клик начинает выполнять запрос с CTE он просто заменяет алиасы на соответствующие подзапросы, в вашем итоговом запросе.

 Любой подзапрос в клике считается отдельно, его результаты никуда НЕ сохраняются и НЕ кешируются.

Всякий раз когда вы обращаетесь к CTE, клик вычисляет ее заново!

### Пример


```
EXPLAIN --
WITH
  line_items_raw AS -- cte      , line_items_raw - alias  `AS`
  (
    SELECT
      shipment_id,
      li_created_at
    FROM analytics.line_items
    WHERE (shipment_id = '684096249') AND (li_created_at >= '2024-07-15')
  ),
  line_items_raw_2 AS --      CTE      , ,      line_items_raw  !
  (
    SELECT
      shipment_id,
      li_created_at
    FROM line_items_raw AS lir
    INNER JOIN line_items_raw AS lir2 ON lir.shipment_id = lir2.shipment_id
  )
SELECT --      CTE      , ,      line_items_raw_2  !
  shipment_id,
  li_created_at
FROM line_items_raw_2 AS lir
INNER JOIN line_items_raw_2 AS lir2 ON lir.shipment_id = lir2.shipment_id

CTE
line_items_raw_2 -> 2
line_items_raw -> 2*2 = 4

4  analytics.line_items
2   line_items_raw_2
1

explain
Expression ((Projection + Before ORDER BY))
Join (JOIN FillRightFirst)                -- 3
Expression ((Before JOIN + (Projection + Before ORDER BY)))
Join (JOIN FillRightFirst)                -- 2
Expression ((Before JOIN + (Projection + Before ORDER BY)))
ReadFromMergeTree (analytics.line_items)  -- 4
Expression ((Joined actions + (Rename joined columns + (Projection + Before ORDER BY))))
ReadFromMergeTree (analytics.line_items)  -- 3
Expression ((Joined actions + (Rename joined columns + (Projection + Before ORDER BY))))
Join (JOIN FillRightFirst)                -- 1
Expression ((Before JOIN + (Projection + Before ORDER BY)))
ReadFromMergeTree (analytics.line_items)  -- 2
Expression ((Joined actions + (Rename joined columns + (Projection + Before ORDER BY))))
ReadFromMergeTree (analytics.line_items)  -- 1
```

В примере выше мы дважды обратились к CTE с джойном и дважды выполнили ее, то есть дважды выполнили джойн, отсюда возникает правило

 Переиспользовать CTE можно, но смотрите на сложность этой CTE:

- Если CTE простая и там, например, забор не большого сырья с фильтрами по партиции, то к такой cte можно обращаться и 2 и 5 раз.
- Если в CTE есть GROUP BY или JOIN, то тут нужно подумать можно ли обойтись в запросе без избыточного обращения к этой CTE, возможно хватит обращения к CTE с сырьем.

## Что делать если CTE сложная, но ее нужно переиспользовать в нескольких других CTE?

В таком случае, нужно оценить `cpu_score` такого запроса, если он превышает пороговое значение (его можно посмотреть по ссылке в [доке](#)), то ваш запрос нужно поделить на два запроса:

- В 1-м вычисляем ту, сложную CTE и кладем результат в [stage таблицу](#).
- Во 2-м достаём данные из этой [stage таблицы](#) и досчитываем изначальный запрос до конца.

Этот прием может сэкономить огромное количество ресурсов.

## Именованние столбцов в SELECT и WHERE

Нужно быть осторожным с именованнием столбцов в SELECT.

В классических БД сначала выполняется фильтрация WHERE, т.е. все столбцы он ищет в колонках таблицы (или в результатах выражения FROM) и на конечном шаге БД уже формирует SELECT.

ClickHouse же сначала ищет колонки в SELECT и только потом ищет в самой таблице (или в результатах выражения FROM).

Рассмотрим запрос:

### Код

```
SELECT
...
< dt> as dt,
...
FROM sandbox.table_a
WHERE dt=toDate('2023-09-01') AND ...
```

В этом случае, даже если в таблице `sandbox.table_a` есть колонка `dt`, то фильтроваться он будет по колонке определенной в селекте "`<выражение для вычисления dt> as dt`".

## Пример того, как можно выстрелить себе в ногу

Приведу пример из реальной практики, когда положили ноды с событийкой.

Событийка `event.new_app` хранится с партицирование по `dwh_dt` и обязательно нужно использовать фильтр по этому полю, также в событийки есть поле `ts` (таймстемп произошедшего события). Была задача посчитать кол-во уникальных пользователей, у которых есть событие `'Product Rendered'`, за каждый день. Сотрудник написал запрос, где переопределил `dwh_dt`:

Код
<pre> WITH     toDate('2023-09-01') AS start_date,     toDate('2023-09-10') AS end_date SELECT     toDate(ts) AS dwh_dt,     uniqExact(anonymous_id) as uniq_anonymous_id FROM event.new_app WHERE dwh_dt &gt;= start_date AND       dwh_dt &lt;= end_date AND       toDate(ts) &gt;= start_date AND       toDate(ts) &lt;= end_date AND       event = 'Product Rendered' GROUP BY dwh_dt </pre>

Данный запрос положил ноды по процессору, т.к. начал по сути вычитывать все исторические данные.

Как следовало написать:

Вариант 1. Как нужно было написать.	Вариант 2. Если важно использовать именно имя dwh_dt
<div> <div>Код</div> <pre> WITH     toDate('2023-09-01') AS start_date,     toDate('2023-09-10') AS end_date SELECT     toDate(ts) AS dt,     uniqExact(anonymous_id) as uniq_anonymous_id FROM event.new_app WHERE dwh_dt &gt;= start_date AND       dwh_dt &lt;= end_date AND       toDate(ts) &gt;= start_date AND       toDate(ts) &lt;= end_date AND       event = 'Product Rendered' GROUP BY dt </pre> </div>	<div> <div>Код</div> <pre> WITH     toDate('2023-09-01') AS start_date,     toDate('2023-09-10') AS end_date , preresult_data AS (     SELECT         toDate(ts) AS tmp_dwh_dt,         uniqExact(anonymous_id) as uniq_anonymous_id     FROM event.new_app     WHERE dwh_dt &gt;= start_date AND           dwh_dt &lt;= end_date AND           toDate(ts) &gt;= start_date AND           toDate(ts) &lt;= end_date AND           event = 'Product Rendered'     GROUP BY tmp_dwh_dt ) SELECT     tmp_dwh_dt AS dwh_dt,     uniq_anonymous_id FROM preresult_data </pre> </div>

Последние две реализации используют в 10 раз меньше вычислительных ресурсов и работают также быстрее в 10 раз.