2. Использование JOIN

Как правильно джойнить таблицы

Подготовка данных в CTE

Всегда во всех запросах читайте сырые данные через CTE с фильтрами и выбором только нужных колонок!

Все фильтры применяются после джойна, никаких оптимизаций здесь нет, т.е. сначала произойдет джойн (секция *FROM*), а только потом фильтрация (секция *WHERE*).

Рассмотрим простой пример:

Код

```
    SELECT count()
FROM analytics.new_app_funnel_table AS wf
INNER JOIN analytics.int_spree_orders AS o ON wf.context_order_id = o.number
WHERE toDate(created_at) = '2022-10-10'

--Elapsed: 1874.032 sec.


WITH
    data_from_funnel AS
    (
        SELECT context_order_id
        FROM analytics.new_app_funnel_table
    ),
    data_from_orders AS
    (
        SELECT number
        FROM analytics.int_spree_orders
        WHERE toDate(created_at) = '2022-10-10'
    )
SELECT count()
FROM data_from_funnel
INNER JOIN data_from_orders ON data_from_funnel.context_order_id = data_from_orders.number

--Elapsed: 7.080 sec.
```

Мало того, что первый запрос выполнялся в 250 раз дольше, так он еще и загрузил процессоры сервера на 50%.

Общая рекомендация: данные необходимо фильтровать настолько рано, насколько это возможно:

- Перед запросом создать CTE, в которых будет происходить только фильтрация сырых данных и выбор нужных колонок. Затем использовать не таблицы, а результаты из CTE.
- На каждом этапе (CTE) в рамках расчета фильтровать полученные из соединения данные, если появляется такая возможность

Порядок таблиц в джойне

Правило № 1

При джоине меньшая по весу таблица должна быть справа, то есть важно не только количество строк, но и количество столбцов и их типы у таблицы справа.

Правило № 2

Если у вас джоинятся несколько таблиц, то указывайте (через CTE) в каком порядке их джоинить. Особенно это важно, если некоторые из джоинов сильно уменьшают количество записей.

Примечание

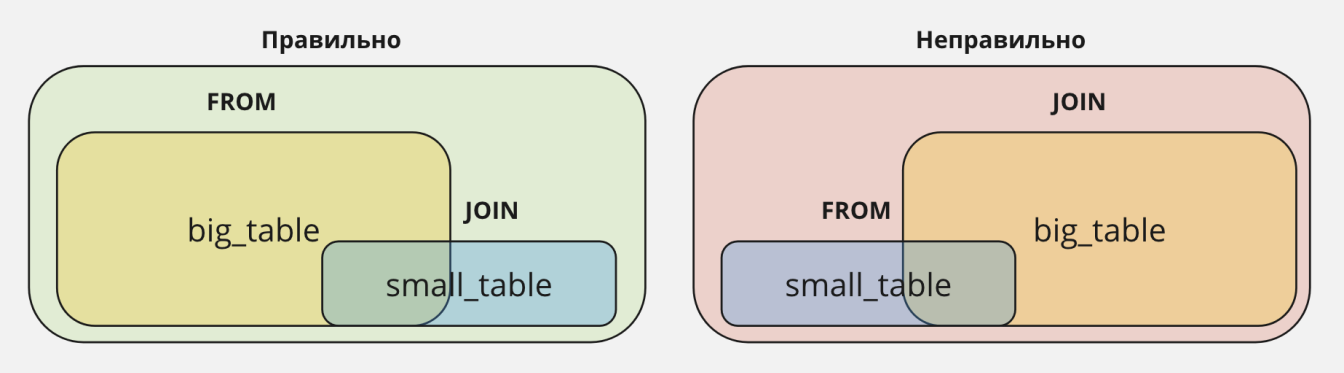
Это касается также и промежуточных таблиц, получаемых в результате CTE.

В ClickHouse отсутствует используемый во многих СУБД стоимостный оптимизатор.

Это значит, что порядок соединения определяется не размером таблиц (и некоторыми другими статистиками), а только синтаксическим видом запроса. Проще говоря, скобочками или структурой общих табличных выражений (CTE).

Поэтому оптимизация в CH проще и ее может сделать каждый: достаточно аккуратно написать запрос, согласно правилам ниже:

Соединение двух таблиц



Меньшая по весу таблица всегда должна быть справа.

Это необходимо для того, чтобы JOIN выполнялся максимально эффективно. Левая таблица считается основной и для каждой строки в ней ищутся соответствующие строки в правой, поэтому, чем меньше правая, тем проще в ней искать. В зависимости от алгоритма, который будет применяться в запросе, правильное расположение таблиц может уменьшить нагрузку на память, процессоры и снизить общее время выполнения в десятки раз.

По умолчанию можно сравнивать таблицы по количеству строк. Но если таблицы сопоставимы по размеру или, например, в одной мы берем одну колонку id и 1.000.000 строк, а в другой 15 колонок разных типов и 10.000 строк, то имеет смысл проверить размер таблиц с помощью запроса:

Запрос для проверки размера колонок

```
    SELECT
    format('{0}.{1}', database, table) AS table_name,
        formatReadableSize(SUM(data_uncompressed_bytes)) AS table_size
FROM system.columns
WHERE
        --
        (table = 'table_name_1' AND name IN ('column_1', 'column_2', 'column_3'))
        --
        OR (table = 'table_name_2' AND name IN ('column_1', 'column_2', 'column_3'))
GROUP BY
    table_name
```

Соединение нескольких таблиц

Если вам необходимо соединить большое количество таблиц (> 2), то

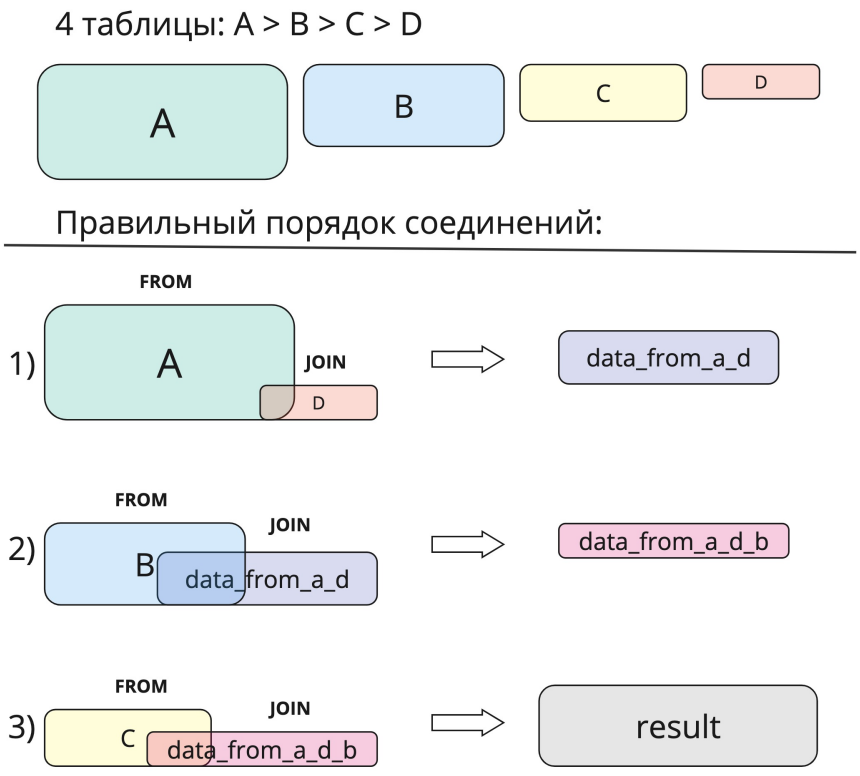
- **В рамках одного CTE желательно чтобы был только 1 join.**
- Если в рамках запроса вам нужно сделать больше одного соединения, то запрос желательно дробить на CTE так, чтобы выполнялся прошлый пункт.
- Последовательность таблиц для распределения по CTE нужно выбирать таким образом, чтобы в каждом CTE справа всегда оказывалась минимальная по размеру таблица из возможных.
- Для каждой отдельной пары необходимо придерживаться правила про соединение двух таблиц.

Рассмотрим пример: пусть у нас четыре таблицы разного размера такие, что A > B > C > D.
Из них мы хотим получить запрос вида

```
    SELECT [ ]
FROM
        A
        JOIN B ON [ ]
        JOIN C ON [ ]
        JOIN D ON [ ]
```

Тогда джойнить следует в следующем порядке:

- Берем самую крупную, **A**, и объединяем с самой маленькой **D**, чтобы максимально обрезать данные: **A join D** получаем cte **data_from_a_d**
- Для следующей по размеру **B** выбираем наименьшую возможную таблицу справа, то есть объединение A и D: **B join data_from_a_d** получаем cte **data_from_a_d_b**
- У нас остались 2 таблицы: C и результат объединения (A, D, B). Пусть C больше, тогда пишем финальное соединение: **C join data_from_a_d_b result**



Фильтрация с помощью IN

Описано в отдельной статье Фильтрация с помощью IN

Реальные примеры оптимизаций

Соединение двух таблиц

Соединим таблицу Big (содержит ~ 63 млн записей) и таблицу Small (содержит ~ 1 млн записей) двумя способами

1) Маленькая таблица слева

```
SELECT *
FROM sandbox_dwh.gorobets_test_join_order_small AS small
INNER JOIN sandbox_dwh.gorobets_test_join_order_big AS big ON small.id_1 = big.id_1
```

2) Большая таблица слева

```
SELECT *
FROM sandbox_dwh.gorobets_test_join_order_big AS big
INNER JOIN sandbox_dwh.gorobets_test_join_order_small AS small ON small.id_1 = big.id_1
```

В случае merge соединения второй запрос выигрывает. Это связано с тем, что min-max индексы строятся для блоков именно правой таблицы.

area	read_rows	written_rows	query_duration_ms	memory_usage	thread_ids
big_right	64 208 382	629 255	36 995	2 280 138 216	{1 769 545,2 636 521,2 635 321, ... [38]}
big_left	64 208 382	629 255	1 359	205 276 270	{1 769 545,2 635 889,2 635 486, ... [40]}

В случае соединения по хэшу второй запрос также выполняется быстрее и эффективнее по памяти в 20 раз. В этом случае query_duration_ms (время выполнения запроса в мс) лучше в 25 раз, memory_usage (использование оперативной памяти) лучше в 22 раза.

area	read_rows	written_rows	query_duration_ms	memory_usage	thread_ids
big_right	64 208 382	629 255	34 647	4 878 809 096	{1 769 545,2 634 768,2 636 256, ... [38]}
big_left	64 208 382	629 255	1 493	222 437 630	{1 769 545,2 633 115,2 699 033, ... [39]}

Это связано с тем, что хэш карта ВСЕГДА строится по правой таблице (даже в LEFT и RIGHT JOIN), а процесс этот не дешевый и выполнять его лучше на маленькой таблице.

Код для теста

```
    DROP TABLE IF EXISTS sandbox_dwh.gorobets_test_join_order_small
;

CREATE TABLE sandbox_dwh.gorobets_test_join_order_small
ENGINE = MergeTree
ORDER BY id_0
SETTINGS index_granularity = 8192 AS
SELECT
    rowNumberInAllBlocks() AS id_0,
    id_1
FROM
(
    SELECT DISTINCT rand(number) % 100000000 AS id_1
    FROM numbers(1000000)
)
;

DROP TABLE IF EXISTS sandbox_dwh.gorobets_test_join_order_big
;

CREATE TABLE sandbox_dwh.gorobets_test_join_order_big
ENGINE = MergeTree
ORDER BY id_0
SETTINGS index_granularity = 8192 AS
SELECT
    rowNumberInAllBlocks() AS id_0,
    id_1
FROM
(
    SELECT DISTINCT rand(number) % 100000000 AS id_1
    FROM numbers(100000000)
)
```

```
;

DROP TABLE IF EXISTS sandbox_dwh.gorobets_test_join_order_1
;

CREATE TABLE sandbox_dwh.gorobets_test_join_order_1
ENGINE = MergeTree
ORDER BY id_0
SETTINGS index_granularity = 8192 AS
SELECT *
FROM sandbox_dwh.gorobets_test_join_order_small AS small
INNER JOIN sandbox_dwh.gorobets_test_join_order_big AS big ON small.id_1 = big.id_1
SETTINGS join_algorithm = 'hash'
;

DROP TABLE IF EXISTS sandbox_dwh.gorobets_test_join_order_2
;

CREATE TABLE sandbox_dwh.gorobets_test_join_order_2
ENGINE = MergeTree
ORDER BY id_0
SETTINGS index_granularity = 8192 AS
SELECT *
FROM sandbox_dwh.gorobets_test_join_order_big AS big
INNER JOIN sandbox_dwh.gorobets_test_join_order_small AS small ON small.id_1 = big.id_1
SETTINGS join_algorithm = 'hash'
;

DROP TABLE IF EXISTS sandbox_dwh.gorobets_test_join_order_1
;

CREATE TABLE sandbox_dwh.gorobets_test_join_order_1
ENGINE = MergeTree
ORDER BY id_0
SETTINGS index_granularity = 8192 AS
SELECT *
FROM sandbox_dwh.gorobets_test_join_order_small AS small
INNER JOIN sandbox_dwh.gorobets_test_join_order_big AS big ON small.id_1 = big.id_1
SETTINGS join_algorithm = 'partial_merge'
;

DROP TABLE IF EXISTS sandbox_dwh.gorobets_test_join_order_2
;

CREATE TABLE sandbox_dwh.gorobets_test_join_order_2
ENGINE = MergeTree
ORDER BY id_0
SETTINGS index_granularity = 8192 AS
SELECT *
FROM sandbox_dwh.gorobets_test_join_order_big AS big
INNER JOIN sandbox_dwh.gorobets_test_join_order_small AS small ON small.id_1 = big.id_1
SETTINGS join_algorithm = 'partial_merge'
;
```

Соединение нескольких таблиц

В примере ниже удалось сократить время работы в 20 раз за счет правильного порядк джойнов.

Сделаем джоин трех таблиц, две из которых содержат числа от 1 до 10.000.000, а одна от 1 до 10.

Запрос вида:

```
    SELECT *
FROM t1
INNER JOIN t2 ON ...
INNER JOIN t3 ON ...
```

не выполнится эффективно, на какое бы место (t1, t2 или t3) вы не поставили маленькую таблицу. Нужно явно указать порядок соединения.

Т.е. любой из следующих примеров не выполняется эффективно

Вариант 1

```
    SELECT
    number_1,
    number_2,
    number_3
FROM
(
    SELECT number AS number_1
    FROM numbers(10)
) AS t1
INNER JOIN
(
    SELECT number AS number_2
    FROM numbers(10000000)
) AS t2 ON t1.number_1 = t2.number_2
INNER JOIN
(
    SELECT number AS number_3
    FROM numbers(10000000)
) AS t3 ON t2.number_2 = t3.number_3
;


time: 10sec
```

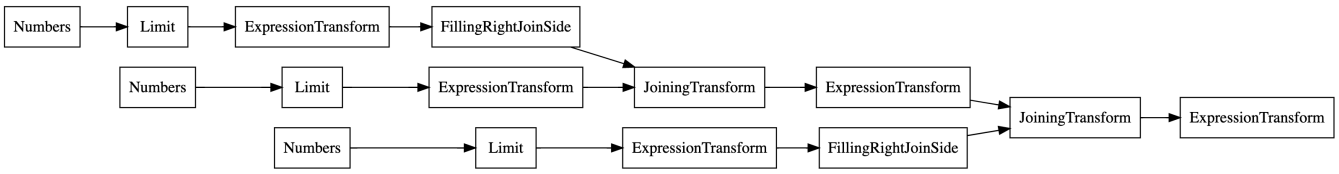
Вариант 2

```
    SELECT
    number_1,
    number_2,
    number_3
FROM
(
    SELECT number AS number_1
    FROM numbers(10000000)
) AS t1
INNER JOIN
(
    SELECT number AS number_2
    FROM numbers(10)
) AS t2 ON t1.number_1 = t2.number_2
INNER JOIN
(
    SELECT number AS number_3
    FROM numbers(10000000)
) AS t3 ON t2.number_2 = t3.number_3
;

time: 5sec
```

Вариант 3

```
    SELECT
    number_1,
    number_2,
    number_3
FROM
(
    SELECT number AS number_1
    FROM numbers(10000000)
) AS t1
INNER JOIN
(
    SELECT number AS number_2
    FROM numbers(10000000)
) AS t2 ON t1.number_1 = t2.number_2
INNER JOIN
(
    SELECT number AS number_3
    FROM numbers(10)
) AS t3 ON t2.number_2 = t3.number_3

 time: 6sec
```

План запроса (explain pipeline graph=1,compact=0) во всех случаях одинаковый:



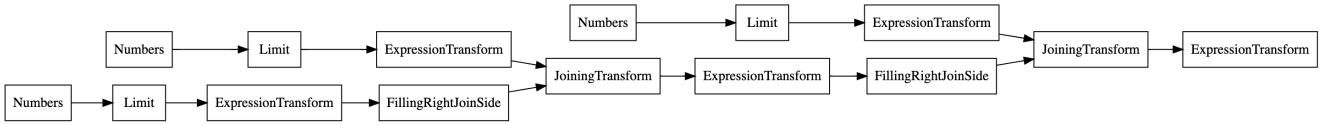
По плану видно, что хэш карта (filling right join side) строится два раза просто по изначальным таблицам. При этом, в первом варианте хэш карта оба раза строится по таблицам с 10 млн записями, поэтому этот вариант примерно в два раза хуже остальных, в которых хэш карта один раз строится по таблице 10 млн и 1 раз по таблице с 10 записями.

Если же явно указать порядок такой, что маленькая таблица каждый раз оказывается справа, то время выполнения будет в десятки раз меньше

Правильный порядок с CTE

```
    WITH
    t1 AS
    (
        SELECT number AS number_1
        FROM numbers(10000000)
    ),
    t1_join_t2 AS
    (
        SELECT
            number_1,
            number AS number_2
        FROM t1
        INNER JOIN numbers(10) AS t2 ON t1.number_1 = t2.number
    )
SELECT
    number_1,
    number_2,
    number AS number_3
FROM numbers(10000000) AS t3
INNER JOIN t1_join_t2 ON  t3.number = t1_join_t2.number_2

time: 0.3 sec
```

План запроса (explain pipeline graph=1,compact=0) в этом случае:



По плану видно, что хэш карта (filling right join side) строится один раз по изначальной таблице (самой маленькой), а второй раз по промежуточной.

Дополнительно. Физическая реализация join'ов.

Для объединения двух таблиц существует несколько разных алгоритмов соединения. Они отличаются скоростью выполнения и требованиями к вычислительным ресурсам (нагрузка процессора, оперативная память).

В ClickHouse используется 2 основных типа соединения: **Hash-join** и **Partial-merge-join** (= Merge-join). По умолчанию таблицы объединяются по Hash-join, но если превышается определенный порог по памяти, то используется Partial-merge-join.

Hash Join

> **Важно!**
>
> Это дорогостоящий по памяти алгоритм. ClickHouse использует этот тип настолько часто, насколько ему позволяют ресурсы.

Алгоритм: правая таблица целиком загружается в оперативную память в виде хэш-таблицы (по своей сути похоже на словарь), где ключом является ключ соединения, а значением — строка, соответствующая этому ключу либо список строк, если таких ключей несколько. Для соединения нужно для каждой строки слева сделать обращение в хэш-таблицу и получить соответствующую строку справа.

Данный подход работает прекрасным образом, если таблица справа маленькая (влезает в оперативную память), а ключ не является длинным — операция хэширования данных сравнительно дорогая штука.

blocked URL

Partial Merge Join (Merge Join)

Важно!

Это алгоритм использует гораздо меньше оперативной памяти, но в общем случае работает немного медленнее, чем Hash-join

Алгоритм работает следующим образом:

1. Обе таблицы сортируем по ключу соединения.
2. Далее идем по этим отсортированным таблицам и соединяем строки по ключу соединения, если он совпадает.

Такой алгоритм очень эффективен за счет того, что он смотрит на данные каждой из таблиц только один раз и мало хранит данных в оперативной памяти.

blocked URL