Про JOIN’ы (в ClickHouse)

Что нового и чего ожидать?

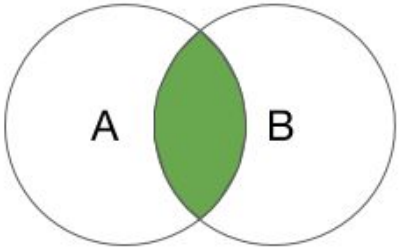
Артем Зуйков, разработчик ClickHouse

Что было год назад?

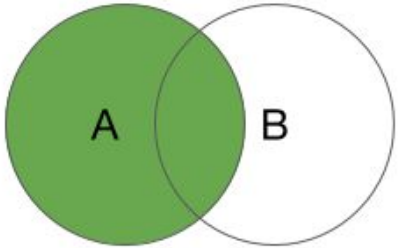
Join двух таблиц:

- [ANY] LEFT | INNER | RIGHT | FULL [OUTER] JOIN
- CROSS JOIN
- [LEFT] ARRAY JOIN

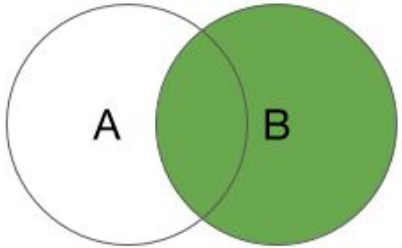
HashJoin (в памяти)



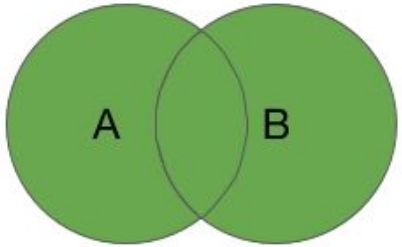
INNER JOIN



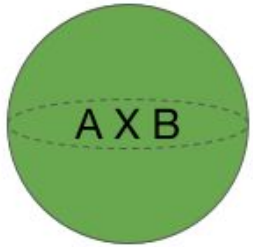
LEFT OUTER JOIN



RIGHT OUTER JOIN



FULL OUTER JOIN



CARTESIAN (CROSS) JOIN

Картинка с сайта thisisdata.ru

Equi-join

Equi-join - группа алгоритмов JOIN, допускающих только равенства в предикатах ON-секции (или USING)

Противопоставляется алгоритмам, допускающим неравенстова

```
SELECT *
FROM left_table AS l
LEFT JOIN right_table AS r
ON l.key = r.key AND l.str = r.str

SELECT *
FROM left_table AS l
LEFT JOIN right_table AS r
USING(key, str)
```

```
SELECT *
FROM left_table AS l
LEFT JOIN right_table AS r
ON l.key = r.key AND l.str > r.str
```

Как работает HashJoin?

```
SELECT *
FROM left_table
INNER JOIN right_table USING (key)
```

1. Берем все строки из right_table (многопоточно)
2. Вставляем их в hash-таблицу по ключу key (лок на запись)
3. Читаем все строки из left_table (многопоточно)
4. Каждой строке из left_table ищем соответствие в hash-таблице (лок на чтение)
5. Объединяем результат (в зависимости от варианта)

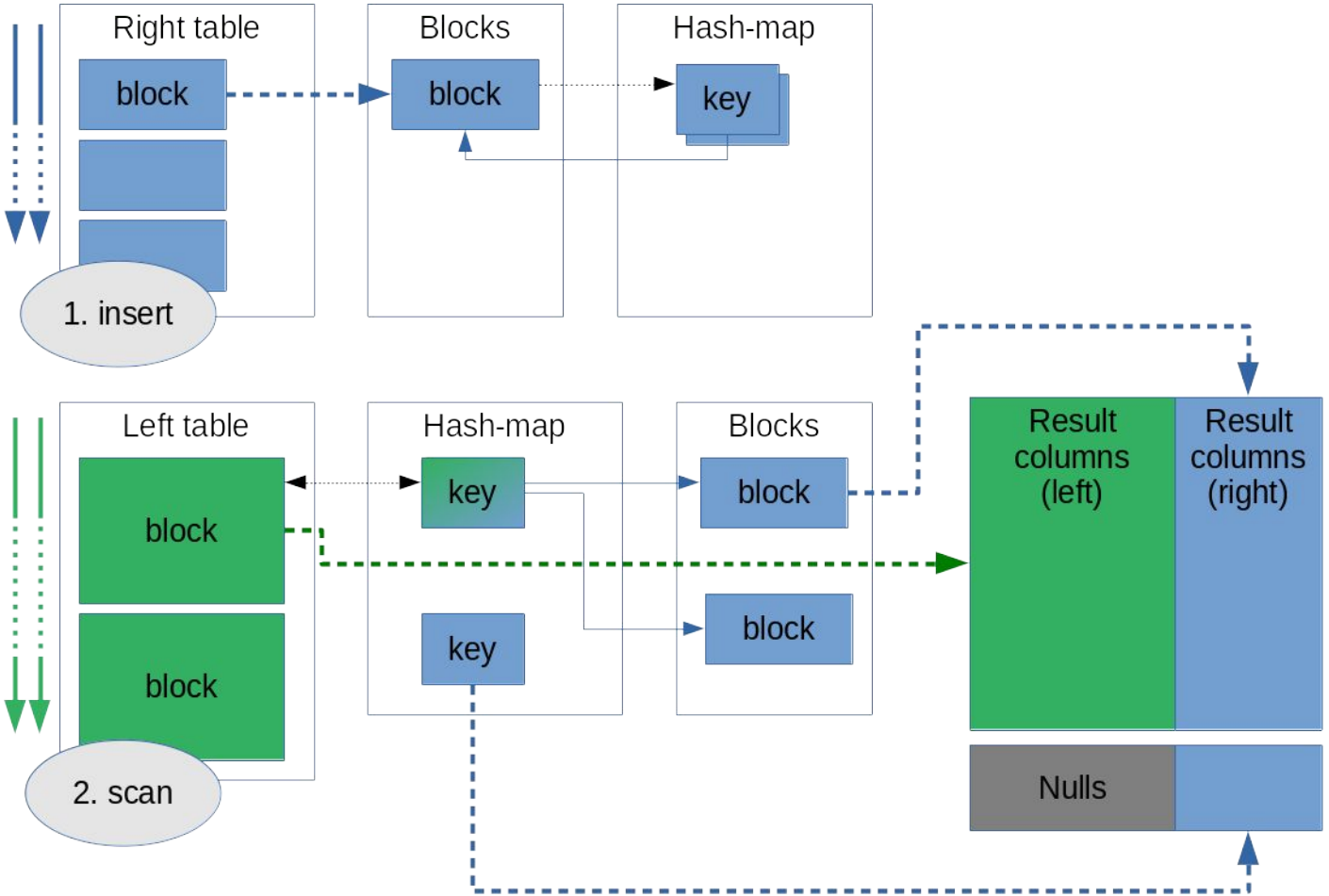
Не совсем так

Данные из right_table сохраняем блоками много_строк * нужные_столбцы

В hash-таблицу кладем ссылку на строку в блоке

Для RIGHT и FULL вариантов (однопоточно) дописываем не встретившиеся строки из hash-таблицы

Для этого в ссылке храним флаг, что key встретился


Right table
block
Blocks
block
Hash-map
key
1. insert
Left table
block
block
2. scan
Hash-map
key
key
Blocks
block
block
Result columns (left)
Result columns (right)
Nulls

Опция join_use_nulls

```
SELECT *
FROM
(
   SELECT 1 AS key
) AS l
LEFT JOIN
(
   SELECT 2 AS key
) AS r ON l.key = r.key
```

```
┌─key─┬─r.key─┐
│   1 │     0 │
└─────┴───────┘
```

```
SELECT *
FROM
(
   SELECT 1 AS key
) AS l
LEFT JOIN
(
   SELECT 2 AS key
) AS r ON l.key = r.key
SETTINGS join_use_nulls = 1
```

```
┌─key─┬─r.key─┐
│   1 │  NULL │
└─────┴───────┘
```

Составные ключи

```
SELECT *
FROM left_table AS l
INNER JOIN right_table AS r ON (l.key = r.key) AND (l.str = r.str)
```

Сериализуем ключи {l.key, l.str}, {r.key, r.str}

По возможности упаковываем в больший тип (2 * Type::key64 -> Type::key128)

Остальное решает hash-таблица

Множественные JOIN’ы

```
SELECT *
FROM left_table
```

```
┌─key─┬─str─┐
│   0 │ l1  │
│   1 │ l2  │
│   1 │ l3  │
│   2 │ l4  │
└─────┴─────┘
```

```
SELECT *
FROM right_table
```

```
┌─key─┬─str─┐
│   1 │ r1  │
│   2 │ r2  │
│   2 │ r3  │
│   3 │ r4  │
└─────┴─────┘
```

```
SELECT *
FROM left_table AS l
LEFT JOIN right_table AS r ON l.key = r.key
LEFT JOIN right_table AS r2 ON l.key = r2.key
```

```
┌─l.key─┬─l.str─┬─r.key─┬─r.str─┬─r2.key─┬─r2.str─┐
│     0 │ l1    │       │       │        │        │
│     1 │ l2    │     1 │ r1    │      1 │ r1     │
│     1 │ l3    │     1 │ r1    │      1 │ r1     │
│     2 │ l4    │     2 │ r2    │      2 │ r2     │
│     2 │ l4    │     2 │ r2    │      2 │ r3     │
│     2 │ l4    │     2 │ r3    │      2 │ r2     │
│     2 │ l4    │     2 │ r3    │      2 │ r3     │
└───────┴───────┴───────┴───────┴────────┴────────┘
```

Множественные JOIN’ы

```
SELECT *
FROM left_table AS l
LEFT JOIN right_table AS r
      ON l.key = r.key
LEFT JOIN right_table AS r2
      ON l.key = r2.key
```

```
SELECT
    `--l.key` AS `l.key`,
    `--l.str` AS `l.str`,
    `--r.key` AS `r.key`,
    `--r.str` AS `r.str`,
    key AS `r2.key`,
    str AS `r2.str`
FROM
(
    SELECT
        key AS `--l.key`,
        str AS `--l.str`,
        r.key AS `--r.key`,
        r.str AS `--r.str`
    FROM left_table AS l
    ALL LEFT JOIN right_table AS r
    ON `--l.key` = `--r.key`
)
ALL LEFT JOIN right_table AS r2 ON `--l.key` = key
```

Переписывание COMMA и CROSS JOIN

```
SELECT *
FROM left_table AS l
, right_table AS r
WHERE l.key > r.key
```

```
┌─key─┬─str─┬─r.key─┬─r.str─┐
│   2 │ l4  │     1 │ r1    │
└─────┴─────┴───────┴───────┘
```

```
SELECT
   key,
   str,
   r.key,
   r.str
FROM left_table AS l
CROSS JOIN right_table AS r
WHERE key > r.key
```

Переписывание COMMA и CROSS JOIN

```
SELECT *
FROM left_table AS l
, right_table AS r
WHERE l.key = r.key
```

```
┌─key─┬─str─┬─r.key─┬─r.str─┐
│   1 │ l2  │     1 │ r1    │
│   1 │ l3  │     1 │ r1    │
│   2 │ l4  │     2 │ r2    │
│   2 │ l4  │     2 │ r3    │
└─────┴─────┴───────┴───────┘
```

```
SELECT
   key,
   str,
   r.key,
   r.str
FROM left_table AS l
ALL INNER JOIN right_table AS r ON key = r.key
WHERE key = r.key
```

ANY JOIN (раньше)

ANY LEFT | INNER | RIGHT | FULL [OUTER] JOIN

ANY JOIN - особенность ClickHouse (нестандартный SQL)

Берем одну (any) строку с нужным ключом из правой таблицы, за счет этого не храним “дубликаты” в hash-таблице (быстрее)

Полезно в аналитических запросах, где нам часто достаточно одного представителя из семейства строк, или если заранее известна уникальность

ANY JOIN проблема 1

Неэквивалентность операций RIGHT и LEFT JOIN при перестановке таблиц

```
SELECT * FROM left_table
```

key	str
0	l1
1	l2
1	l3
2	l4

```
SELECT * FROM right_table
```

key	str
1	r1
2	r2
2	r3
3	r4

```
SELECT *
FROM left_table AS l
ANY LEFT JOIN right_table AS r
USING (key)
```

key	str	r.str
0	l1	
1	l2	r1
1	l3	r1
2	l4	r2

```
SELECT *
FROM right_table AS r
ANY RIGHT JOIN left_table AS l
USING (key)
```

key	str	l.str
1	r1	l2
2	r2	l4
2	r3	l4

key	str	l.str
0		l1

ANY JOIN проблема 2

Некоммутативный INNER JOIN

```
SELECT * FROM left_table
┌─key─┬─str─┐
│   0 │ l1  │
│   1 │ l2  │
│   1 │ l3  │
│   2 │ l4  │
└─────┴─────┘
SELECT * FROM right_table
┌─key─┬─str─┐
│   1 │ r1  │
│   2 │ r2  │
│   2 │ r3  │
│   3 │ r4  │
└─────┴─────┘
```

```
SELECT *
FROM left_table AS l
ANY INNER JOIN right_table AS r
USING (key)

┌─key─┬─str─┬─r.str─┐
│   1 │ l2  │ r1    │
│   1 │ l3  │ r1    │
│   2 │ l4  │ r2    │
└─────┴─────┴───────┘
```

```
SELECT *
FROM right_table AS r
ANY INNER JOIN left_table AS l
USING (key)

┌─key─┬─str─┬─l.str─┐
│   1 │ r1  │ l2    │
│   2 │ r2  │ l4    │
│   2 │ r3  │ l4    │
└─────┴─────┴───────┘
```

ANY JOIN (новый)

ANY LEFT | INNER | RIGHT [OUTER] JOIN

LEFT - any-строка из правой таблицы (как было)

RIGHT - any-строка из левой таблицы (поменялось)

INNER - any-строка из каждой таблицы (поменялось)

FULL - не ясна семантика (запрещен)

Старое поведение доступно под опцией any_join_distinct_right_table_keys

ANY JOIN (новый)

Эквивалентность операций RIGHT и LEFT JOIN при перестановке таблиц

```
SELECT *
FROM left_table AS l
ANY LEFT JOIN right_table AS r
USING (key)
```

key	str	r.str
0	l1	
1	l2	r1
1	l3	r1
2	l4	r2

```
SELECT *
FROM right_table AS r
ANY RIGHT JOIN left_table AS l
USING (key)
```

key	str	l.str
1	r1	l2
1	r1	l3
2	r2	l4

key	str	l.str
0		l1

```
SELECT *
FROM right_table AS r
ANY RIGHT JOIN left_table AS l
USING (key)
SETTINGS
any_join_distinct_right_table_
keys = 1
```

key	str	l.str
1	r1	l2
2	r2	l4
2	r3	l4

key	str	l.str
0		l1

ANY JOIN (новый)

Коммутативный INNER JOIN

```
SELECT *
FROM left_table AS l
ANY INNER JOIN right_table AS r
USING (key)
```

```
┌─key─┬─str─┬─r.str─┐
│   1 │ l2  │ r1    │
│   2 │ l4  │ r2    │
└─────┴─────┴───────┘
```

```
SELECT *
FROM right_table AS r
ANY INNER JOIN left_table AS l
USING (key)
```

```
┌─key─┬─str─┬─l.str─┐
│   1 │ r1  │ l2    │
│   2 │ r2  │ l4    │
└─────┴─────┴───────┘
```

```
SELECT *
FROM left_table AS l
ANY INNER JOIN right_table AS r
USING (key)
SETTINGS
any_join_distinct_right_table_keys = 1
```

```
┌─key─┬─str─┬─r.str─┐
│   1 │ l2  │ r1    │
│   1 │ l3  │ r1    │
│   2 │ l4  │ r2    │
└─────┴─────┴───────┘
```

SEMI JOIN

SEMI LEFT | RIGHT JOIN

SEMI JOIN - продвинутый вариант операции EXISTS: выдаем те строки, ключи которых присутствуют во второй таблице

LEFT - фильтруем левую таблицу по ключам правой
RIGHT - фильтруем правую таблицу по ключам левой

Для фильтрующей таблицы извлекаем any-строки (особенность ClickHouse)

SEMI JOIN

Старый ANY INNER JOIN стал SEMI LEFT JOIN

```
SELECT * FROM left_table
┌─key─┬─str─┐
│   0 │ l1  │
│   1 │ l2  │
│   1 │ l3  │
│   2 │ l4  │
└─────┴─────┘
SELECT * FROM right_table
┌─key─┬─str─┐
│   1 │ r1  │
│   2 │ r2  │
│   2 │ r3  │
│   3 │ r4  │
└─────┴─────┘
```

```
SELECT *
FROM left_table AS l
SEMI LEFT JOIN right_table AS r
USING (key)

┌─key─┬─str─┬─r.str─┐
│   1 │ l2  │ r1    │
│   1 │ l3  │ r1    │
│   2 │ l4  │ r2    │
└─────┴─────┴───────┘
```

```
SELECT *
FROM right_table AS r
SEMI RIGHT JOIN left_table AS l
USING (key)

┌─key─┬─str─┬─l.str─┐
│   1 │ r1  │ l2    │
│   1 │ r1  │ l3    │
│   2 │ r2  │ l4    │
└─────┴─────┴───────┘
```

ANTI JOIN

ANTI LEFT | RIGHT JOIN

ANTI JOIN - продвинутый вариант операции NOT EXISTS

Работает аналогично SEMI JOIN. Возвращает строки, ключи которых отсутствуют в другой таблице

ANTI JOIN

SEMI и ANTI JOIN сохраняют эквивалентность LEFT и RIGHT вариантов

```
SELECT * FROM left_table
┌─key─┬─str─┐
│   0 │ l1  │
│   1 │ l2  │
│   1 │ l3  │
│   2 │ l4  │
└─────┴─────┘
SELECT * FROM right_table
┌─key─┬─str─┐
│   1 │ r1  │
│   2 │ r2  │
│   2 │ r3  │
│   3 │ r4  │
└─────┴─────┘
```

```
SELECT *
FROM left_table AS l
ANTI LEFT JOIN right_table AS r
USING (key)

┌─key─┬─str─┬─r.str─┐
│   0 │ l1  │       │
└─────┴─────┴───────┘
```

```
SELECT *
FROM right_table AS r
ANTI RIGHT JOIN left_table AS l
USING (key)

┌─key─┬─str─┬─l.str─┐
│   0 │     │ l1    │
└─────┴─────┴───────┘
```

ASOF JOIN

ASOF LEFT | INNER JOIN

```
SELECT
   l.dt,
   r.dt
FROM left_table AS l
ASOF LEFT JOIN right_table AS r
ON (l.currency = r.currency)
AND (l.dt >= r.dt)
```

dt	r.dt
1970-01-01 03:00:10	0000-00-00 00:00:00
1970-01-01 03:00:15	1970-01-01 03:00:11
1970-01-01 03:00:17	1970-01-01 03:00:17
1970-01-01 03:00:21	1970-01-01 03:00:20

```
SELECT
   l.dt,
   r.dt
FROM left_table AS l
ASOF LEFT JOIN right_table AS r
ON (l.currency = r.currency)
AND (l.dt > r.dt)
```

dt	r.dt
1970-01-01 03:00:10	0000-00-00 00:00:00
1970-01-01 03:00:15	1970-01-01 03:00:11
1970-01-01 03:00:17	1970-01-01 03:00:11
1970-01-01 03:00:21	1970-01-01 03:00:20

ASOF JOIN

Для каждой левой строки возвращается не более одной правой строки

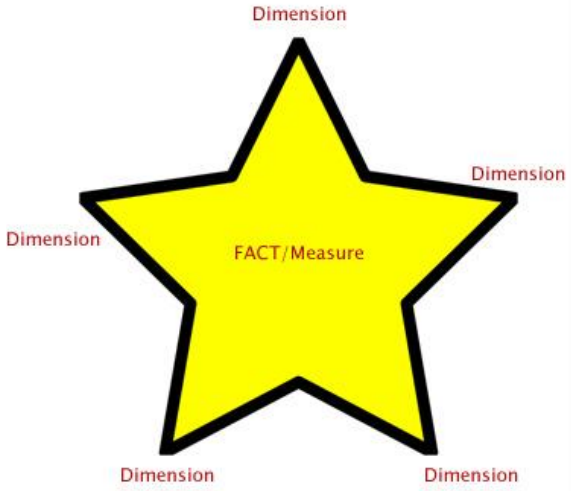
Возвращается ближайшее значение, удовлетворяющее условию

Похоже на ANY JOIN с поддержкой условия на одну колонку: **>**, **<**, **>=**, **<=**

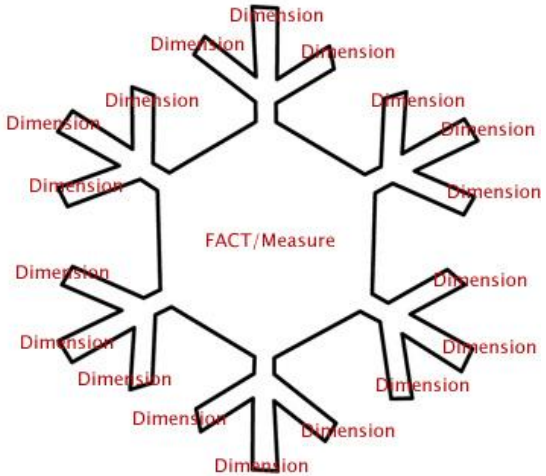
По сути, делаем ordered map по ASOF-колонке внутри hash-таблицы:

1. Делаем обычный HashJoin по всем колонкам, кроме одной
2. Для нее сохраняем в ключе все значения со ссылками на строки
3. Делаем бинарный поиск по сохраненным значениям

Ограничения HashJoin



Star Schema



SnowFlake Schema

Неявно предполагаем, что left_table - таблица фактов

Размер right_table много меньше размера left_table

Проблема: right_table должна влезать в память

Картинка с сайта www.learnmsbitutorials.net

Как сделать снежинку?

Переносим JOIN dimension’ов в правый подзапрос

```
SELECT *
FROM fact
JOIN dim1 ON fact.x = dim1.x
JOIN dim2 ON fact.y = dim2.y
```

```
SELECT *
FROM fact
JOIN
(
   SELECT *
   FROM dim1
   JOIN dim2 ON dim1.z = dim2.z
) AS dmix
ON fact.x = dmix.x AND fact.y = dmix.y
```

Альтернатива: MergeJoin

1. Сортируем по ключу и сохраняем на диск left_table
2. Сортируем по ключу и сохраняем на диск right_table
3. Идем по отсортированным данным двумя курсорами
4. Объединяем строки с одинаковыми ключами

Проблема:

Сортировать и сохранять left_table дорого (и по времени и по месту)

Partial MergeJoin

1. Сортируем и сохраняем на диск right_table
2. Читаем left_table крупными порциями
3. Сортируем каждую порцию left_table
4. Объединяем каждую порцию left_table с right_table по алгоритму MergeJoin отдельно

Проблема:

Плохая асимптотика алгоритма в общем случае

Асимптотика Patrial MergeJoin

При Partial MergeJoin нужно перечитывать всю right_table на каждую порцию left_table (квадратичная сложность при фиксированной памяти)

Можно показать, что при фиксированном размере left_table и фиксированном объеме памяти до некоторого размера right_table быстрее Partial вариант, после - обычный MergeJoin

Если размер правой таблицы в несколько раз больше располагаемой памяти, эффективней использовать Partial MergeJoin, если в сотни - MergeJoin

Partial MergeJoin тормозит

```
SELECT count(1)
FROM catalog_sales
┌─count(1)─┐
│ 144005443 │
└──────────┘
```

```
SELECT count(1)
FROM inventory
┌─count(1)─┐
│ 399330000 │
└──────────┘
```

```
SELECT count(1) FROM catalog_sales AS cs
ANY LEFT JOIN inventory AS inv ON cs.cs_item_sk = inv.inv_item_sk
WHERE (inv.inv_quantity_on_hand >= 100)
      AND (inv.inv_quantity_on_hand <= 500)
SETTINGS partial_merge_join = 1,
      max_rows_in_join = 3200000,
      partial_merge_join_rows_in_right_blocks = 32000

┌─count(1)─┐
│ 54732602 │
└──────────┘

1 rows in set. Elapsed: 302.142 sec. Processed 543.34 million rows, 5.94
GB (1.80 million rows/s., 19.67 MB/s.)
```

```
Insert: ~0.33 time. CPU utilization: 1 core
Scan:   ~0.66 time. CPU utilization: 25 cores (25 * 300 = 7500)
```

Partial MergeJoin тормозит-2

```
SELECT count(1)
FROM catalog_sales
┌─count(1)─┐
│ 144005443 │
└──────────┘

SELECT count(1)
FROM inventory
┌─count(1)─┐
│ 399330000 │
└──────────┘

SELECT count(1)
FROM item
┌─count(1)─┐
│   204000 │
└──────────┘
```

```
SELECT count(1) FROM catalog_sales AS cs
ANY LEFT JOIN inventory AS inv ON cs.cs_item_sk = inv.inv_item_sk
ANY LEFT JOIN item AS i ON inv.inv_item_sk = i.i_item_sk
WHERE ((i.i_current_price >= 20)
      AND (i.i_current_price <= 50))
      AND ((inv.inv_quantity_on_hand >= 100)
      AND (inv.inv_quantity_on_hand <= 500))
SETTINGS partial_merge_join = 1,
      max_rows_in_join = 3200000,
      partial_merge_join_rows_in_right_blocks = 32000

┌─count(1)─┐
│  1817292 │
└──────────┘

1 rows in set. Elapsed: 375.839 sec. Processed 543.54 million rows, 5.95
GB (1.45 million rows/s., 15.82 MB/s.)

Scan CPU utilization: 20 cores (20 * 375 = 7500)
```

Что стало?

Join нескольких таблиц:

- [ANY | ASOF | SEMI | ANTI] LEFT | INNER | RIGHT | FULL JOIN
- CROSS JOIN
- COMMA JOIN
- [LEFT] ARRAY JOIN

HashJoin (в памяти), Partial MergeJoin (в памяти или со сбросом на диск)

Оптимизация COMMA и CROSS JOIN на уровне AST

Что дальше?

1. Не тормозящий Partial MergeJoin
2. Оптимизация Partial MergeJoin для отсортированных данных
3. RIGHT и FULL варианты Partial MergeJoin
4. LookupJoin:
   - HashJoin с выгружаемой hash-таблицей
   - просто реализовать как JOIN со словарем
5. Выбор стратегии JOIN-а: хинты, зачатки оптимизатора
6. Прокидывание предикатов в JOIN
7. Distributed JOIN (развитие и замена GLOBAL JOIN)