

Способы соединения

Соединение слиянием



Авторские права

© Postgres Professional, 2019–2024

Авторы: Егор Рогов, Павел Лузанов, Павел Толмачев, Илья Баштанов

Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

edu@postgrespro.ru

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Алгоритм соединения слиянием

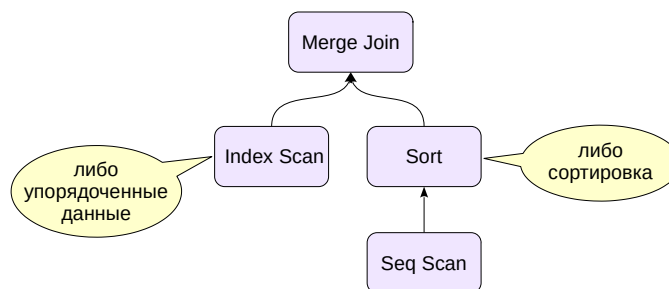
Вычислительная сложность

Соединение слиянием в параллельных планах

Соединение слиянием

Слияние двух отсортированных наборов строк

Результат соединения автоматически отсортирован



Третий, и последний, способ соединения — соединение слиянием.

Идея этого способа состоит в том, что два предварительно отсортированных набора данных нетрудно объединить в один общий — и точно так же отсортированный — набор. Похожим образом работает узел Gather Merge.

Подготовительным этапом для соединения слиянием служит сортировка обоих наборов строк. Сортировка — дорогая операция, она имеет сложность $O(N \log N)$.

Но иногда этого этапа удастся избежать, если можно сразу получить отсортированные по нужным столбцам наборы строк, например, за счет индексного доступа к таблице.

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year
1	Yellow Submarine	1969
3	Let It Be	1970
4	The Beatles	1968
6	Abbey Road	1969

album_id	name
1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life

Само слияние устроено просто. Сначала берем первые строки обоих наборов и сравниваем их между собой. В данном случае мы сразу нашли соответствие и можем вернуть первую строку результата: («Yellow Submarine», «All Together Now»).

Общий алгоритм таков: читаем следующую строку того набора, для которого значение поля, по которому происходит соединение, меньше (один набор «догоняет» другой). Если же значения одинаковы, как в нашем примере, то читаем следующую строку второго (внутреннего) набора.

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year
1	Yellow Submarine	1969
3	Let It Be	1970
4	The Beatles	1968
6	Abbey Road	1969

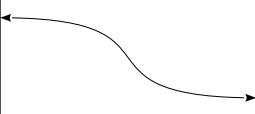
album_id	name
1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life

Вновь соответствие: («Yellow Submarine», «All You Need Is Love»).

Снова читаем следующую строку второго набора.

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year	album_id	name
1	Yellow Submarine	1969	1	All Together Now
3	Let It Be	1970	1	All You Need Is Love
4	The Beatles	1968	2	Another Girl
6	Abbey Road	1969	2	Act Naturally
			3	Across the Universe
			5	A Day in the Life



В данном случае соответствия нет.

Поскольку $1 < 2$, читаем следующую строку первого набора.

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year
1	Yellow Submarine	1969
3	Let It Be	1970
4	The Beatles	1968
6	Abbey Road	1969

album_id	name
1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life



Соответствия нет.

3 > 2, поэтому читаем следующую строку второго набора.

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year	album_id	name
1	Yellow Submarine	1969	1	All Together Now
3	Let It Be	1970	1	All You Need Is Love
4	The Beatles	1968	2	Another Girl
6	Abbey Road	1969	2	Act Naturally
			3	Across the Universe
			5	A Day in the Life

Снова нет соответствия, снова $3 > 2$, снова читаем строку второго набора.


```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year
1	Yellow Submarine	1969
3	Let It Be	1970
4	The Beatles	1968
6	Abbey Road	1969

album_id	name
1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life

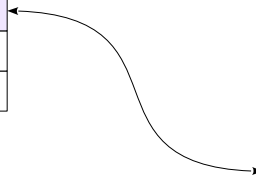
Есть соответствие: («Let It Be», «Across the Universe»).

3 = 3, читаем следующую строку второго набора.

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year
1	Yellow Submarine	1969
3	Let It Be	1970
4	The Beatles	1968
6	Abbey Road	1969

album_id	name
1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life



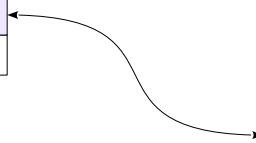
Соответствия нет.

3 < 5, читаем строку первого набора.

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year
1	Yellow Submarine	1969
3	Let It Be	1970
4	The Beatles	1968
6	Abbey Road	1969

album_id	name
1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life




Соответствия нет.

4 < 5, читаем строку первого набора.

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

id	title	year	album_id	name
1	Yellow Submarine	1969	1	All Together Now
3	Let It Be	1970	1	All You Need Is Love
4	The Beatles	1968	2	Another Girl
6	Abbey Road	1969	2	Act Naturally
			3	Across the Universe
			5	A Day in the Life



И последний шаг: снова нет соответствия.

На этом соединение слиянием окончено.

На самом деле алгоритм сложнее — если в первом (внешнем) наборе строк встречается несколько одинаковых значений, нужно иметь возможность перечитать строки второго (внутреннего) набора с тем же ключом соединения.

Псевдокод алгоритма можно посмотреть в файле <src/backend/executor/nodeMergejoin.c>.

Важно, что алгоритм слияния возвращает результат соединения в отсортированном виде. В частности, полученный набор строк можно использовать для следующего соединения слиянием без дополнительной сортировки.

Соединение слиянием

Если результат необходим в отсортированном виде, оптимизатор может предпочесть соединение слиянием. Особенно, если данные от дочерних узлов можно получить уже отсортированными — как в этом примере:

```
=> EXPLAIN (costs off) SELECT *
FROM tickets t
     JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no
ORDER BY t.ticket_no;
```

QUERY PLAN

```
-----
Merge Join
  Merge Cond: (t.ticket_no = tf.ticket_no)
    -> Index Scan using tickets_pkey on tickets t
    -> Index Scan using ticket_flights_pkey on ticket_flights tf
(4 rows)
```

Вот еще один пример с двумя соединениями слиянием, в котором один узел Merge Join получает отсортированный набор от другого узла Merge Join:

```
=> EXPLAIN (costs off) SELECT t.ticket_no, bp.flight_id, bp.seat_no
FROM tickets t
     JOIN ticket_flights tf ON t.ticket_no = tf.ticket_no
     JOIN boarding_passes bp ON bp.ticket_no = tf.ticket_no
     AND bp.flight_id = tf.flight_id
ORDER BY t.ticket_no;
```

QUERY PLAN

```
-----
Merge Join
  Merge Cond: ((t.ticket_no = tf.ticket_no) AND (bp.flight_id = tf.flight_id))
    -> Merge Join
      Merge Cond: (bp.ticket_no = t.ticket_no)
        -> Index Scan using boarding_passes_pkey on boarding_passes bp
        -> Index Only Scan using tickets_pkey on tickets t
    -> Index Only Scan using ticket_flights_pkey on ticket_flights tf
(7 rows)
```

Здесь соединяются перелеты (ticket_flights) и посадочные талоны (boarding_passes), и с этим, уже отсортированным по номерам билетов, набором строк соединяются билеты (tickets).

Для Merge Join также существуют модификации Left, Right, Semi, Anti и Full.

В настоящее время соединение слиянием поддерживает только эквисоединения, соединения по операциям «больше» или «меньше» не реализованы.

Таким образом, модификация Full существует только для соединений хешированием и слиянием, и оба варианта работают только с условием равенства. Если пренебречь производительностью, полное соединение можно получить, объединив результаты левого соединения и антисоединения — таким вариантом придется воспользоваться, если потребуются полное соединение по иному условию.

$\sim N + M$, где

N и M — число строк в первом и втором наборах данных,
если не требуется сортировка

$\sim N \log N + M \log M$,

если сортировка нужна

Возможные начальные затраты на сортировку

Эффективно для большого числа строк

В случае, когда не требуется сортировать данные, общая сложность соединения слиянием пропорциональна сумме числа строк в обоих наборах данных. Но, в отличие от соединения хешированием, здесь не требуются накладные расходы на построение хеш-таблицы.

Поэтому соединение слиянием может успешно применяться как в OLTP-, так и в OLAP-запросах.

Однако если сортировка требуется, то стоимость становится пропорциональной произведению количества строк на логарифм этого количества, и на больших объемах такой способ скорее всего проиграет соединению хешированием.

Вычислительная сложность

Посмотрим на стоимость соединения слиянием:

```
=> EXPLAIN SELECT *
FROM tickets t
     JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no
ORDER BY t.ticket_no;
```

QUERY PLAN

```
-----
Merge Join  (cost=0.99..822518.73 rows=8391030 width=136)
  Merge Cond: (t.ticket_no = tf.ticket_no)
    -> Index Scan using tickets_pkey on tickets t  (cost=0.43..139110.04 rows=2949837
width=104)
    -> Index Scan using ticket_flights_pkey on ticket_flights tf  (cost=0.56..571146.22
rows=8391030 width=32)
(4 rows)
```

Начальная стоимость включает:

- сумму начальных стоимостей дочерних узлов (включает стоимость сортировки, если она необходима);
- стоимость получения первой пары строк, соответствующих друг другу.

Полная стоимость добавляет к начальной:

- сумму стоимостей получения обоих наборов данных;
- стоимость сравнения строк.

Общий вывод: стоимость соединения слиянием пропорциональна $N + M$ (где N и M — число соединяемых строк), если не требуется отдельная сортировка. Сортировка набора из K строк добавляет к оценке как минимум $K \times \log(K)$.

В отличие от соединения хешированием, слияние без сортировки хорошо подходит для случая, когда надо быстро получить первые строки.

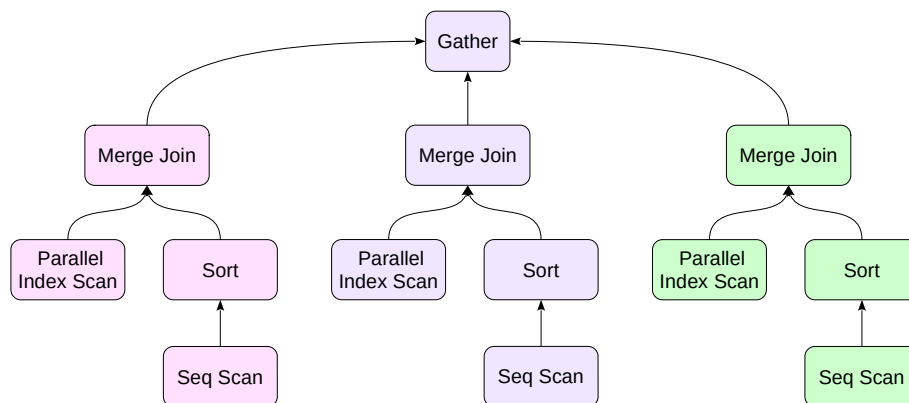
```
=> EXPLAIN SELECT *
FROM tickets t
     JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no
ORDER BY t.ticket_no
LIMIT 1000;
```

QUERY PLAN

```
-----
Limit  (cost=0.99..99.01 rows=1000 width=136)
  -> Merge Join  (cost=0.99..822518.73 rows=8391030 width=136)
    Merge Cond: (t.ticket_no = tf.ticket_no)
      -> Index Scan using tickets_pkey on tickets t  (cost=0.43..139110.04
rows=2949837 width=104)
      -> Index Scan using ticket_flights_pkey on ticket_flights tf
(cost=0.56..571146.22 rows=8391030 width=32)
(5 rows)
```

Обратите внимание и на то, как уменьшилась общая стоимость.

Внешний набор строк сканируется параллельно,
внутренний — последовательно каждым процессом

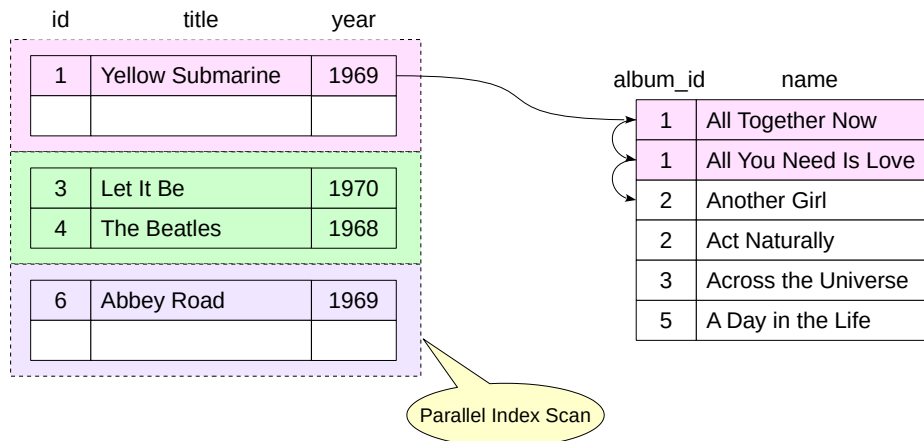


Соединение слиянием может использоваться в параллельных планах.

Так же, как и при соединении вложенным циклом, сканирование одного набора строк выполняется рабочими процессами параллельно, но другой набор строк каждый рабочий процесс читает полностью самостоятельно. Поэтому при соединении больших объемов строк в параллельных планах гораздо чаще используется соединение хешированием, имеющее эффективный параллельный алгоритм.

В параллельных планах

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```

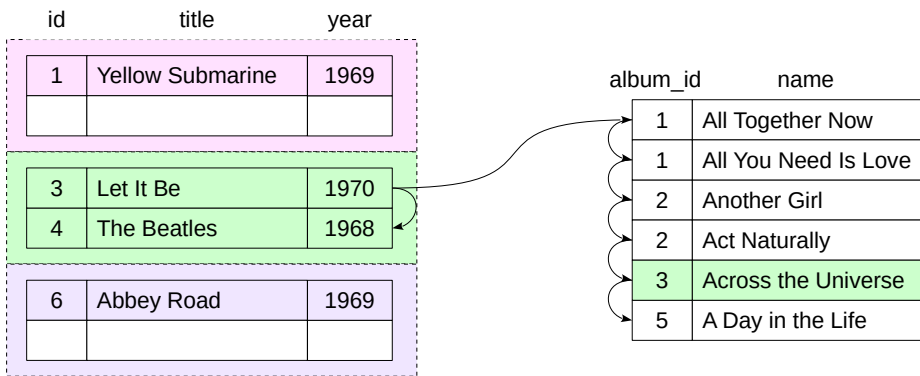


Внутренний набор данных будет просканирован каждым из рабочих процессов от начала и до того момента, как станет понятно, что больше нет соответствий.

На этом слайде показаны строки, перебираемые первым процессом.

В параллельных планах

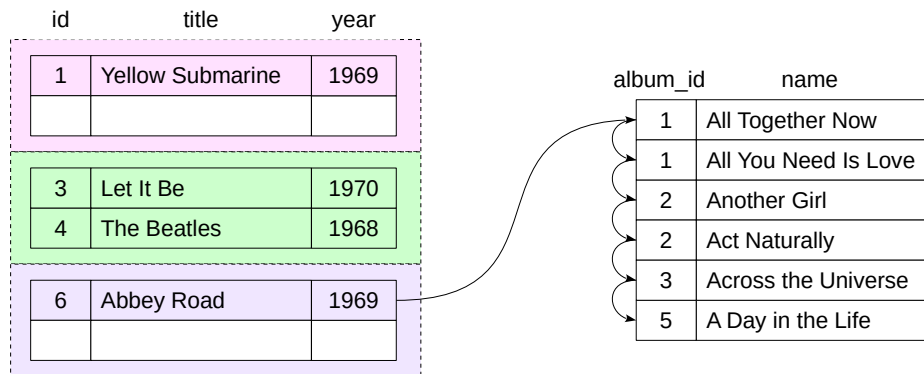
```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```



Второй процесс просканирует весь набор и найдет одно соответствие.

В параллельных планах

```
SELECT a.title, s.name  
FROM albums a JOIN songs s ON a.id = s.album_id;
```



А третий процесс не обнаружит ни одного соответствия.

Конечно, все три процесса сканируют внутренний набор одновременно, а не по очереди.

В параллельных планах

Нам потребуется индекс:

```
=> CREATE INDEX ON tickets(book_ref);
```

CREATE INDEX

Вот пример параллельного плана, в котором используется соединение слиянием:

```
=> EXPLAIN (costs off)
SELECT count(*)
FROM bookings b
JOIN (
  SELECT book_ref FROM tickets GROUP BY book_ref
) t ON b.book_ref = t.book_ref;
```

QUERY PLAN

```
-----
Finalize Aggregate
-> Gather
    Workers Planned: 2
    -> Partial Aggregate
        -> Merge Join
            Merge Cond: (b.book_ref = tickets.book_ref)
            -> Parallel Index Only Scan using bookings_pkey on bookings b
            -> Group
                Group Key: tickets.book_ref
                -> Index Only Scan using tickets_book_ref_idx on tickets

(10 rows)
```

Здесь индекс на таблице бронирований bookings_pkey сканируется параллельно, а результат группировки из подзапроса t — каждым процессом полностью.

Соединение слиянием может потребовать подготовки

- надо отсортировать наборы строк
- или получить их заранее отсортированными

Эффективно для больших выборок

- хорошо, если наборы уже отсортированы
- хорошо, если нужен отсортированный результат

Не зависит от порядка соединения

Поддерживает только эквисоединения

- другие не реализованы, но принципиальных ограничений нет

Чтобы начать соединение слиянием, оба набора строк должны быть отсортированы. Хорошо, если удастся получить данные уже в нужном порядке; если нет — требуется выполнить сортировку.

Само слияние выполняется очень эффективно даже для больших наборов строк. В качестве приятного бонуса результирующая выборка тоже упорядочена, поэтому такой способ соединения привлекателен, если вышестоящим узлам плана также требуется сортировка (например, запрос с фразой ORDER BY или еще одна сортировка слиянием).

Итак, в распоряжении планировщика есть три способа соединения: вложенный цикл, хеширование и слияние (не считая различных модификаций). Для каждого способа есть ситуации, в которых он оказывается более эффективным, чем остальные. Это позволяет планировщику выбрать именно тот способ, который — как предполагается — лучше подойдет в каждом конкретном случае. А точность предположений напрямую зависит от имеющейся статистики.

1. Проверьте план выполнения запроса, показывающего все места в салонах в порядке кодов самолетов:

```
SELECT * FROM aircrafts a
  JOIN seats s ON a.aircraft_code = s.aircraft_code
ORDER BY a.aircraft_code;
```

Но оформите его в виде курсора.
Уменьшите значение параметра *cursor_tuple_fraction* в десять раз. Как при этом изменился план выполнения?
2. Самолет можно заменить другим, если его вместимость отличается не более, чем на 20%. Получите таблицу замен между моделями Боинг и Аэробус, выполнив полное соединение. Как выполняется запрос?

22

2. Запрос, который нужно выполнить:

```
WITH cap AS (
  SELECT a.model, count(*)::numeric capacity
  FROM aircrafts a
  JOIN seats s ON a.aircraft_code = s.aircraft_code
  GROUP BY a.model
), a AS (
  SELECT * FROM cap WHERE model LIKE 'Аэробус%'
), b AS (
  SELECT * FROM cap WHERE model LIKE 'Боинг%'
)
SELECT a.model AS airbus, b.model AS boeing
FROM a FULL JOIN b
  ON b.capacity::numeric/a.capacity BETWEEN 0.8 AND 1.2
ORDER BY 1,2;
```

1. Параметр cursor_tuple_fraction

План выполнения курсора:

```
=> EXPLAIN DECLARE c CURSOR FOR SELECT *
FROM aircrafts a
JOIN seats s ON a.aircraft_code = s.aircraft_code
ORDER BY a.aircraft_code;
```

QUERY PLAN

```
-----
Merge Join (cost=1.51..420.71 rows=1339 width=55)
  Merge Cond: (s.aircraft_code = ml.aircraft_code)
    -> Index Scan using seats_pkey on seats s (cost=0.28..64.60 rows=1339 width=15)
    -> Sort (cost=1.23..1.26 rows=9 width=72)
        Sort Key: ml.aircraft_code
        -> Seq Scan on aircrafts_data ml (cost=0.00..1.09 rows=9 width=72)
(6 rows)
```

Текущее значение cursor_tuple_fraction:

```
=> SHOW cursor_tuple_fraction;
```

```
cursor_tuple_fraction
-----
0.1
(1 row)
```

Уменьшим его:

```
=> SET cursor_tuple_fraction = 0.01;
```

SET

```
=> EXPLAIN DECLARE c CURSOR FOR SELECT *
FROM aircrafts a
JOIN seats s ON a.aircraft_code = s.aircraft_code
ORDER BY a.aircraft_code;
```

QUERY PLAN

```
-----
Merge Join (cost=0.41..431.73 rows=1339 width=55)
  Merge Cond: (ml.aircraft_code = s.aircraft_code)
    -> Index Scan using aircrafts_pkey on aircrafts_data ml (cost=0.14..12.27 rows=9 width=72)
    -> Index Scan using seats_pkey on seats s (cost=0.28..64.60 rows=1339 width=15)
(4 rows)
```

Теперь планировщик выбирает другой план: его начальная стоимость ниже (хотя общая стоимость, наоборот, выше).

2. Полное соединение

Попробуем выполнить запрос.

```
=> WITH cap AS (
  SELECT a.model, count(*)::numeric capacity
  FROM aircrafts a
  JOIN seats s ON a.aircraft_code = s.aircraft_code
  GROUP BY a.model
), a AS (
  SELECT * FROM cap WHERE model LIKE 'Аэробус%'
), b AS (
  SELECT * FROM cap WHERE model LIKE 'Боинг%'
)
SELECT a.model AS airbus, b.model AS boeing
FROM a FULL JOIN b
ON b.capacity::numeric/a.capacity BETWEEN 0.8 AND 1.2
ORDER BY 1,2;
```

ERROR: FULL JOIN is only supported with merge-joinable or hash-joinable join conditions

Получаем ошибку: полное соединение реализовано только для условия равенства, поскольку только вложенный цикл поддерживает соединение по произвольному условию, но зато не поддерживает полное соединение.

Обойти ограничение можно, объединив результаты левого внешнего соединения и антисоединения:

```
=> WITH cap AS (  
    SELECT a.model, count(*)::numeric capacity  
    FROM aircrafts a  
    JOIN seats s ON a.aircraft_code = s.aircraft_code  
    GROUP BY a.model  
) , a AS (  
    SELECT * FROM cap WHERE model LIKE 'Аэробус%'  
) , b AS (  
    SELECT * FROM cap WHERE model LIKE 'Боинг%'  
)  
SELECT a.model AS airbus, b.model AS boeing  
FROM a LEFT JOIN b  
    ON b.capacity::numeric/a.capacity BETWEEN 0.8 AND 1.2  
UNION ALL  
SELECT NULL, b.model  
FROM b  
WHERE NOT EXISTS (  
    SELECT 1  
    FROM a  
    WHERE b.capacity::numeric/a.capacity BETWEEN 0.8 AND 1.2  
)  
ORDER BY 1,2;
```

airbus	boeing
Аэробус A319-100	Боинг 737-300
Аэробус A320-200	Боинг 737-300
Аэробус A321-200	Боинг 767-300
	Боинг 777-300

(5 rows)

План запроса показывает, что соединения выполняются методом вложенного цикла:

```
=> EXPLAIN (costs off)  
WITH cap AS (  
    SELECT a.model, count(*)::numeric capacity  
    FROM aircrafts a  
    JOIN seats s ON a.aircraft_code = s.aircraft_code  
    GROUP BY a.model  
) , a AS (  
    SELECT * FROM cap WHERE model LIKE 'Аэробус%'  
) , b AS (  
    SELECT * FROM cap WHERE model LIKE 'Боинг%'  
)  
SELECT a.model AS airbus, b.model AS boeing  
FROM a LEFT JOIN b  
    ON b.capacity::numeric/a.capacity BETWEEN 0.8 AND 1.2  
UNION ALL  
SELECT NULL, b.model  
FROM b  
WHERE NOT EXISTS (  
    SELECT 1  
    FROM a  
    WHERE b.capacity::numeric/a.capacity BETWEEN 0.8 AND 1.2  
)  
ORDER BY 1,2;
```


QUERY PLAN

```

-----
Sort
  Sort Key: a.model, b.model
  CTE cap
    -> HashAggregate
      Group Key: (ml.model ->> lang())
      -> Hash Join
        Hash Cond: (s.aircraft_code = ml.aircraft_code)
        -> Seq Scan on seats s
        -> Hash
          -> Seq Scan on aircrafts_data ml

  CTE a
    -> CTE Scan on cap
      Filter: (model ~~ 'Аэробус% '::text)

  CTE b
    -> CTE Scan on cap cap_1
      Filter: (model ~~ 'Боинг% '::text)

  -> Append
    -> Nested Loop Left Join
      Join Filter: (((b.capacity / a.capacity) >= 0.8) AND ((b.capacity /
a.capacity) <= 1.2))
      -> CTE Scan on a
      -> CTE Scan on b
    -> Nested Loop Anti Join
      Join Filter: (((b_1.capacity / a_1.capacity) >= 0.8) AND ((b_1.capacity /
a_1.capacity) <= 1.2))
      -> CTE Scan on b b_1
      -> CTE Scan on a a_1

(25 rows)

```