Оптимизация запросов

Сканирование по битовой карте

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

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Сканирование по битовой карте

Сравнение эффективности разных методов доступа

Битовая карта

Построение битовой карты (Bitmap Index Scan)

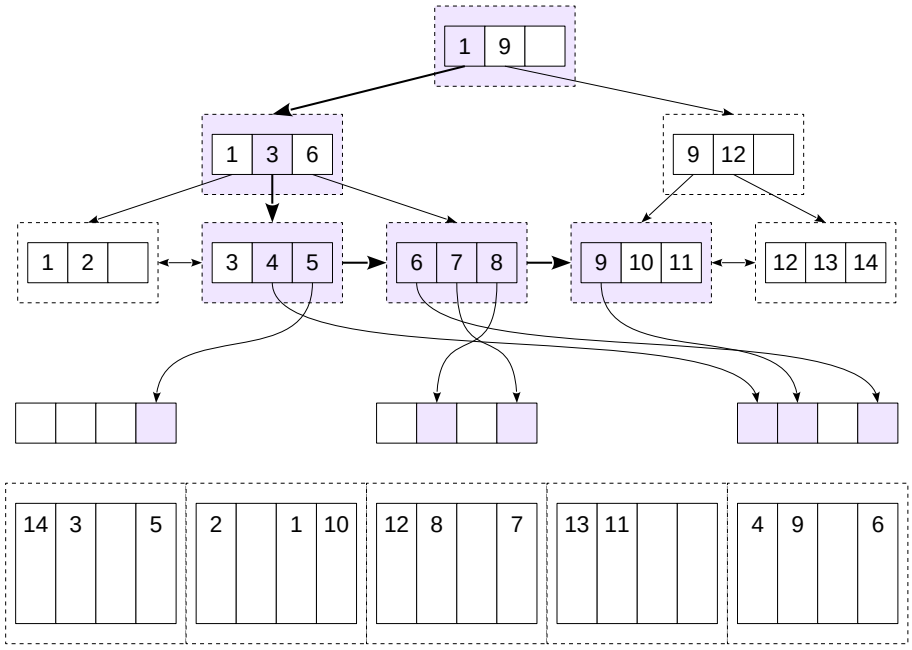
Сканирование по битовой карте (Bitmap Heap Scan)

Точные и неточные фрагменты

Параллельное сканирование (Parallel Bitmap Heap Scan)

Объединение битовых карт

Bitmap Index Scan



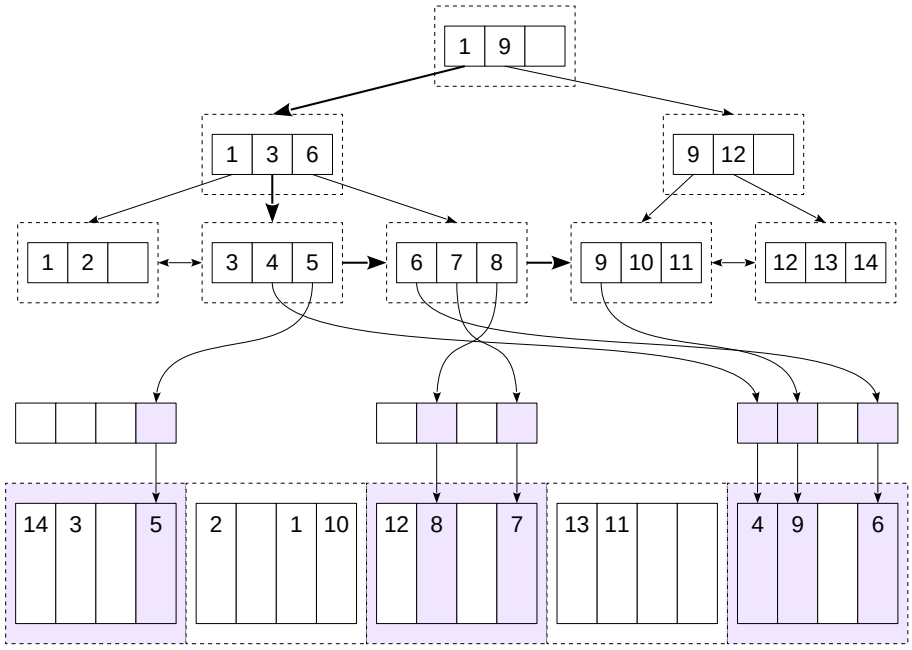
Многократный просмотр одних и тех же табличных страниц крайне неэффективен. Даже в лучшем случае, если нужная страница находится в буферном кеше, ее нужно найти и заблокировать (см. курс DBA2: тема «Буферный кеш» модуля «Журналирование»), а в худшем приходится иметь дело со случайными чтениями с диска.

Чтобы не тратить ресурсы на повторный просмотр табличных страниц, применяется еще один способ доступа — сканирование по битовой карте. Он похож на обычный индексный доступ, но происходит в два этапа.

Сначала сканируется индекс (Bitmap Index Scan) и в локальной памяти процесса строится битовая карта. Битовая карта состоит из фрагментов. Фрагменты соответствуют табличным страницам, а каждый бит фрагмента соответствует версии строки в этой странице. При построениии битовой карты в ней отмечаются те версии строк, которые удовлетворяют условию и должны быть прочитаны.

За счет разделения битовой карты на фрагменты карта, в которой отмечено немного версий, будет занимать мало места.

Bitmap Heap Scan



Когда индекс просканирован и битовая карта готова, начинается сканирование таблицы (Bitmap Heap Scan). При этом:

- используется собственный механизм предвыборки, при котором асинхронно читаются *effective_io_concurrency* страниц (по умолчанию значение равно единице);

- на одной странице может проверяться несколько версий строк, но каждая страница просматривается ровно один раз.

Сканирование по битовой карте

Будем рассматривать таблицу бронирований bookings.

Создадим на ней два дополнительных индекса:

```
=> CREATE INDEX ON bookings(book_date);
```

```
CREATE INDEX
```

```
=> CREATE INDEX ON bookings(total_amount);
```

```
CREATE INDEX
```

Посмотрим, какой метод доступа будет выбран для поиска диапазона.

```
=> EXPLAIN SELECT * FROM bookings WHERE total_amount < 10000;
```

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Bitmap Heap Scan on bookings  (cost=1145.71..15356.08 rows=61069 width=21)
   Recheck Cond: (total_amount < '10000'::numeric)
   ->  Bitmap Index Scan on bookings_total_amount_idx  (cost=0.00..1130.45 rows=61069 width=0)
         Index Cond: (total_amount < '10000'::numeric)
(4 rows)
```

Выбран метод доступа Bitmap Scan. Он состоит из двух узлов:

- Bitmap Index Scan читает индекс и строит битовую карту;
- Bitmap Heap Scan читает табличные страницы, используя построенную карту.

Обратите внимание, что карта должна быть построена полностью, прежде чем ее можно будет использовать.

Объединение битовых карт

Кроме того, что битовая карта позволяет избежать повторных чтений табличных страниц, с ее помощью можно объединять нескольких условий.

```
=> EXPLAIN (costs off)
  SELECT * FROM bookings
  WHERE total_amount < 10000 OR total_amount > 100000;
```

```
                                     QUERY PLAN
-------------------------------------------------------------------------------------
 Bitmap Heap Scan on bookings
   Recheck Cond: ((total_amount < '10000'::numeric) OR (total_amount > '100000'::numeric))
   ->  BitmapOr
         ->  Bitmap Index Scan on bookings_total_amount_idx
               Index Cond: (total_amount < '10000'::numeric)
         ->  Bitmap Index Scan on bookings_total_amount_idx
               Index Cond: (total_amount > '100000'::numeric)
(7 rows)
```

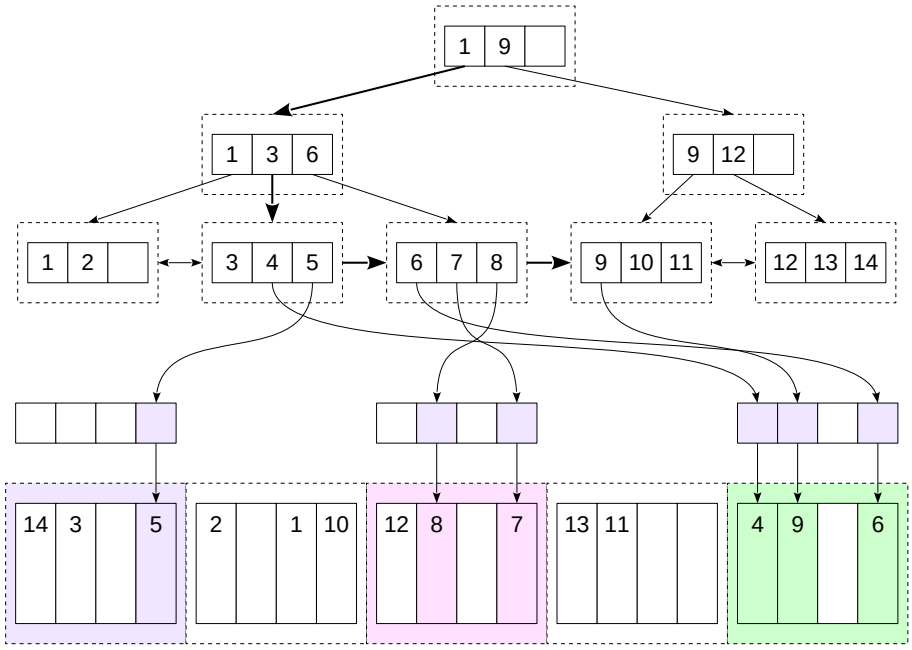
Здесь сначала были построены две битовые карты — по одной на каждое условие, а затем объединены побитовой операцией «или».

Таким же образом могут быть использованы и разные индексы.

```
=> EXPLAIN (costs off)
  SELECT * FROM bookings
  WHERE total_amount < 10000
     OR book_date = bookings.now() - INTERVAL '1 day';
```

```
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on bookings
   Recheck Cond: ((total_amount < '10000'::numeric) OR (book_date = ('2017-08-15 18:00:00+03'::timestamp with time zone - '1 day'::interval)))
   ->  BitmapOr
         ->  Bitmap Index Scan on bookings_total_amount_idx
               Index Cond: (total_amount < '10000'::numeric)
         ->  Bitmap Index Scan on bookings_book_date_idx
               Index Cond: (book_date = ('2017-08-15 18:00:00+03'::timestamp with time zone - '1 day'::interval))
(7 rows)
```

Parallel Bitmap Heap Scan



Сканирование по битовой карте может выполняться параллельно.

Первый этап — сканирование индекса — всегда выполняется последовательно ведущим процессом.

А второй этап — сканирование таблицы — выполняется рабочими процессами параллельно. Это происходит аналогично параллельному последовательному сканированию.

Параллельное сканирование по битовой карте

Сканирование по битовой карте может работать в параллельном режиме. Сколько бронирований сделано за последний месяц на сумму до 20 тысяч ₽?

```
=> SELECT bookings.now() - INTERVAL '1 months' AS d \gset
```

```
=> EXPLAIN (costs off)
  SELECT count(*) FROM bookings
  WHERE total_amount < 20000 AND book_date > :'d';
```

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Parallel Bitmap Heap Scan on bookings
                     Recheck Cond: (book_date > '2017-07-15 18:00:00+03'::timestamp with time zone)
                     Filter: (total_amount < '20000'::numeric)
                     ->  Bitmap Index Scan on bookings_book_date_idx
                           Index Cond: (book_date > '2017-07-15 18:00:00+03'::timestamp with time zone)
(9 rows)
```

Узел Bitmap Index Scan выполняется ведущим процессом, который строит битовую карту. Параллельно выполняется только сканирование таблицы по уже готовой битовой карте — узел Parallel Bitmap Heap Scan.

Неточные фрагменты

Битовая карта без потери точности

пока размер карты не превышает *work_mem*,
информация хранится с точностью до версии строки

Битовая карта с потерей точности

если память закончилась, происходит огрубление
части уже построенной карты до отдельных страниц

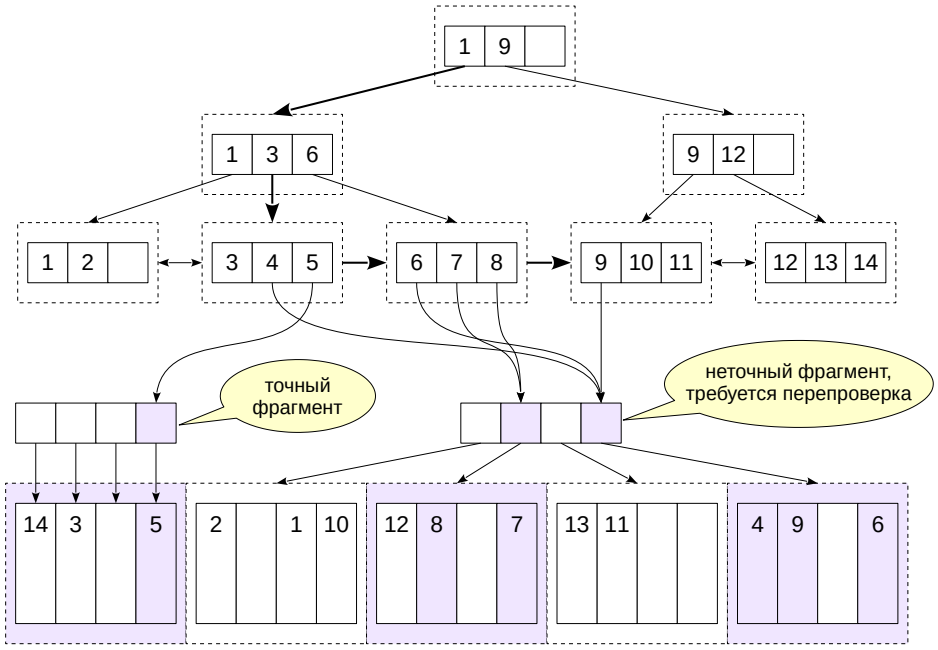
требуется примерно 1 МБ памяти на 64 ГБ данных;
ограничение памяти может быть превышено

Битовая карта хранится в локальной памяти обслуживающего процесса и под ее хранение выделяется *work_mem* байт. Временные файлы никогда не используются.

Если карта перестает помещаться в *work_mem,* часть ее фрагментов «огрубляется» — каждый бит начинает соответствовать целой странице, а не отдельной версии строки (lossy bitmap). Стоимость обработки таких фрагментов увеличивается. Освободившееся место используется для того, чтобы продолжить строить карту.

В принципе, при сильно ограниченном *work_mem* и большой выборке, битовая карта может не поместиться в памяти, даже если в ней совсем не останется информации на уровне версий строк. В таком случае ограничение *work_mem* нарушается — под карту будет дополнительно выделено столько памяти, сколько необходимо.

Неточные фрагменты



На рисунке показана битовая карта, состоящая из двух фрагментов. Первый фрагмент — точный, каждый его бит соответствует одной версии строки. Второй фрагмент — неточный, в нем каждый бит соответствует целой странице.

Неточные фрагменты требуют перепроверки условий для всех версий строк в табличной странице, а это сказывается на производительности. Если две битовые карты объединяются, причем фрагмент хотя бы одной из карт неточен, то и результирующий фрагмент тоже вынужденно будет неточным. Поэтому размер *work_mem* играет большую роль для эффективности сканирования по битовой карте.

Неточные фрагменты

Узел Bitmap Heap Scan показывает условие перепроверки (Recheck Cond). Сама же перепроверка выполняется не всегда, а только при потере точности, когда битовая карта не помещается в память.

Повторим запрос, изменив условие для получения бронирований до 10 тысяч ₽:

```
=> SELECT bookings.now() - INTERVAL '1 months' AS d \gset

=> EXPLAIN (analyze, costs off, timing off)
  SELECT count(*) FROM bookings
  WHERE total_amount < 10000.00 AND book_date > :'d';
```

```
                                                                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate (actual rows=1 loops=1)
   ->  Bitmap Heap Scan on bookings (actual rows=5386 loops=1)
         Recheck Cond: ((total_amount < 10000.00) AND (book_date > '2017-07-15 18:00:00+03'::timestamp with time zone))
         Heap Blocks: exact=4413
         ->  BitmapAnd (actual rows=0 loops=1)
               ->  Bitmap Index Scan on bookings_total_amount_idx (actual rows=63944 loops=1)
                     Index Cond: (total_amount < 10000.00)
               ->  Bitmap Index Scan on bookings_book_date_idx (actual rows=178142 loops=1)
                     Index Cond: (book_date > '2017-07-15 18:00:00+03'::timestamp with time zone)
 Planning Time: 0.078 ms
 Execution Time: 22.026 ms
(11 rows)
```

Строка «Heap Blocks: exact» говорит о том, что все фрагменты битовой карты построены с точностью до строк — перепроверка не выполняется.

Уменьшим размер выделяемой памяти.

```
=> SET work_mem = '64kB';
```

```
SET
```

```
=> EXPLAIN (analyze, costs off, timing off)
  SELECT count(*) FROM bookings
  WHERE total_amount < 10000.00 AND book_date > :'d';
```

```
                                                                                                                                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Finalize Aggregate (actual rows=1 loops=1)
   ->  Gather (actual rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial Aggregate (actual rows=1 loops=3)
               ->  Parallel Bitmap Heap Scan on bookings (actual rows=1795 loops=3)
                     Recheck Cond: ((total_amount < 10000.00) AND (book_date > '2017-07-15 18:00:00+03'::timestamp with time zone))
                     Rows Removed by Index Recheck: 648741
                     Heap Blocks: exact=291 lossy=4009
                     ->  BitmapAnd (actual rows=0 loops=1)
                           ->  Bitmap Index Scan on bookings_total_amount_idx (actual rows=63944 loops=1)
                                 Index Cond: (total_amount < 10000.00)
                           ->  Bitmap Index Scan on bookings_book_date_idx (actual rows=178142 loops=1)
                                 Index Cond: (book_date > '2017-07-15 18:00:00+03'::timestamp with time zone)
 Planning Time: 0.087 ms
 Execution Time: 338.574 ms
(16 rows)
```

Здесь появились lossy-фрагменты битовой карты — с точностью до страниц. Также указано, сколько строк не прошло перепроверку условия (Rows Removed by Index Recheck).

Планировщик переключился на параллельный план, поскольку стоимость сканирования по битовой карте (теперь неточной) увеличилась.

Восстановим значение параметра.

```
=> RESET work_mem;
```

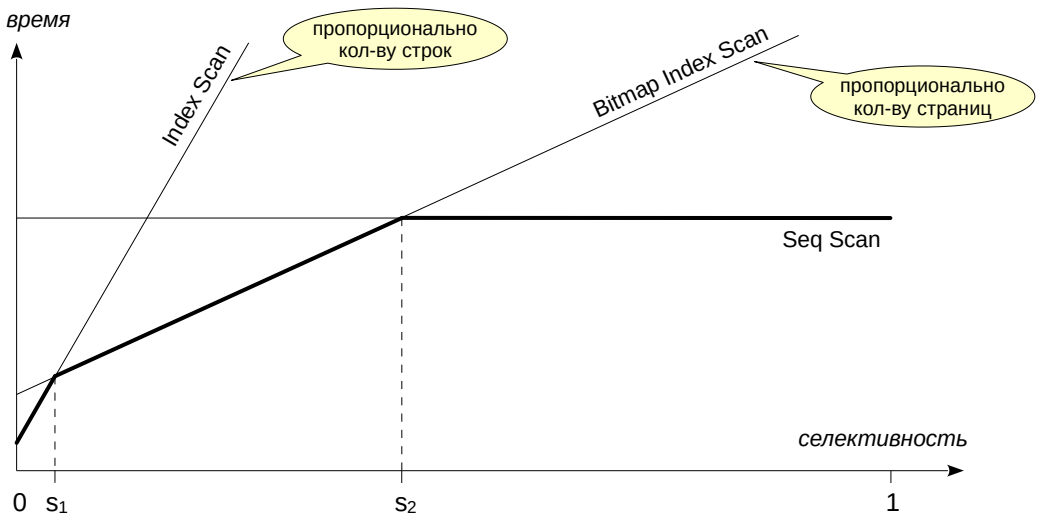
```
RESET
```

Сравнение эффективности

Сравнение различных методов доступа

Кластеризация

Сравнение эффективности



Индексное сканирование лучше всего работает при очень высокой селективности, когда по индексу выбирается одно или несколько значений.

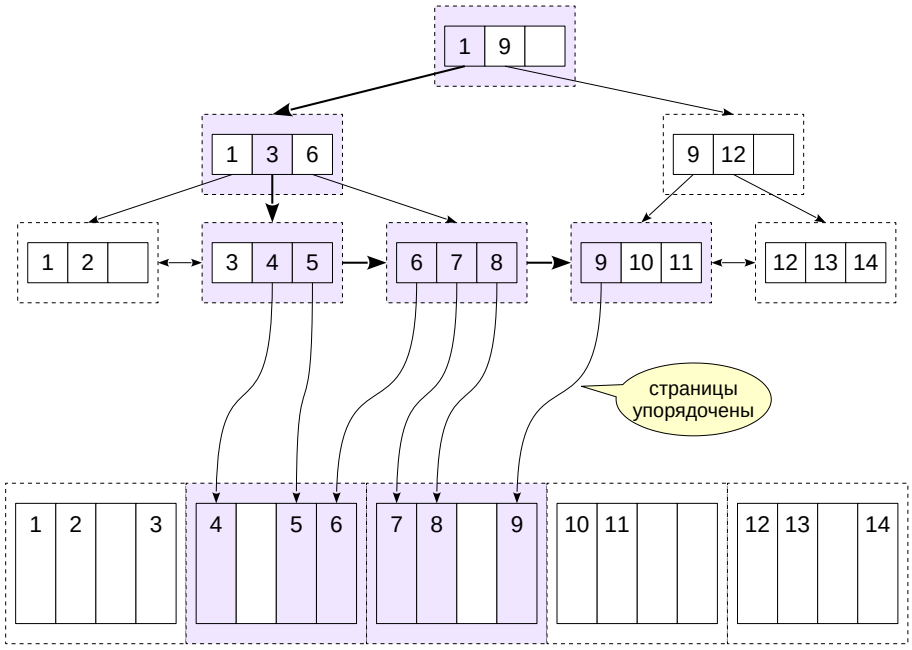
При средней селективности лучше всего показывает себя сканирование по битовой карте. Оно работает лучше индексного сканирования, поскольку обходится без повторных чтений одних и тех же страниц. Однако при сканировании по битовой карте возникают накладные расходы на построение карты, поэтому при высокой селективности оно проигрывает.

При низкой селективности лучше всего работает последовательное сканирование: если надо выбрать все или почти все табличные строки, обращение к индексным страницам только увеличивает накладные расходы. Этот эффект усиливается в случае вращающихся дисков, где стоимость произвольного чтения существенно выше стоимости чтения последовательно расположенных страниц.

Значение селективности, при котором становится выгодно переключиться на другой метод доступа, сильно зависит от конкретной таблицы и конкретного индекса. Планировщик учитывает множество параметров, чтобы выбрать наиболее подходящий способ.

Еще одно замечание: при индексном доступе результат возвращается отсортированным, что в ряде случаев увеличивает привлекательность такого доступа даже при низкой селективности.

Index Scan / Bitmap Scan



Если данные в таблице физически упорядочены, обычное индексное сканирование не будет читать табличную страницу повторно. В таком (нечастом на практике) случае метод сканирования по битовой карте теряет смысл, проигрывая обычному индексному сканированию.

Разумеется, планировщик это тоже учитывает (как именно – рассматривается в теме «Статистика»).

Кластеризация

Если строки таблицы упорядочены так же, как и индекс, битовая карта становится излишней. Продемонстрируем это с помощью команды CLUSTER.

Сейчас строки таблицы физически упорядочены по номеру бронирования:

```
=> SELECT * FROM bookings LIMIT 10;
```

```
 book_ref |       book_date        | total_amount
----------+------------------------+--------------
 000004   | 2016-08-13 15:40:00+03 |     55800.00
 00000F   | 2017-07-05 03:12:00+03 |    265700.00
 000010   | 2017-01-08 19:45:00+03 |     50900.00
 000012   | 2017-07-14 09:02:00+03 |     37900.00
 000026   | 2016-08-30 11:08:00+03 |     95600.00
 00002D   | 2017-05-20 18:45:00+03 |    114700.00
 000034   | 2016-08-08 05:46:00+03 |     49100.00
 00003F   | 2016-12-12 15:02:00+03 |    109800.00
 000048   | 2016-09-17 01:57:00+03 |     92400.00
 00004A   | 2016-10-13 21:57:00+03 |     29000.00
(10 rows)
```

Переупорядочим строки в соответствии с индексом по столбцу total_amount.

Пока идет процесс, обратите внимание:

- Команда CLUSTER устанавливает исключительную блокировку, поскольку полностью перестраивает таблицу (как VACUUM FULL);
- Строки упорядочиваются, но не поддерживаются в упорядоченном виде — в процессе работы кластеризация будет ухудшаться.

```
=> CLUSTER bookings USING bookings_total_amount_idx;
```

```
CLUSTER
```

```
=> ANALYZE bookings;
```

```
ANALYZE
```

Убедимся, что строки упорядочены по стоимости:

```
=> SELECT * FROM bookings LIMIT 10;
```

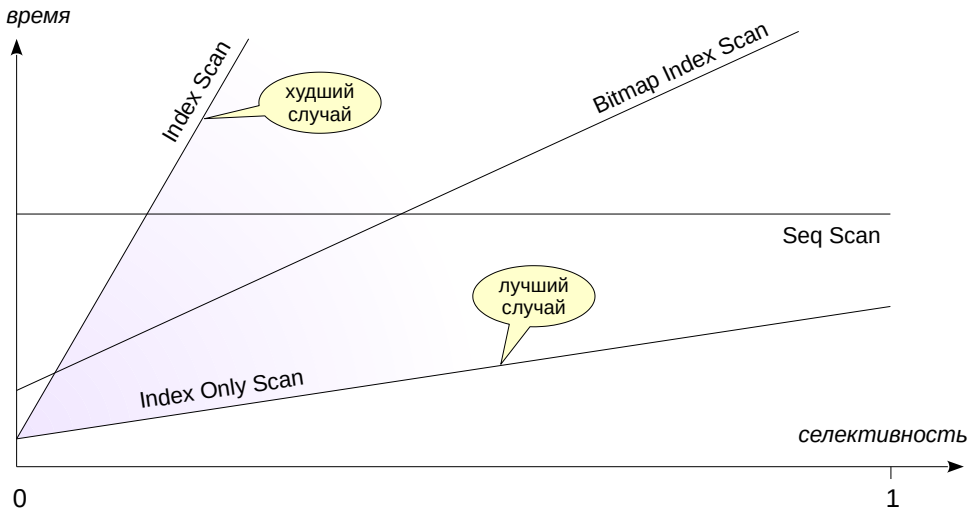
```
 book_ref |       book_date        | total_amount
----------+------------------------+--------------
 00F39E   | 2017-02-12 04:11:00+03 |      3400.00
 0103E1   | 2017-04-03 09:32:00+03 |      3400.00
 013695   | 2016-11-01 09:30:00+03 |      3400.00
 0158C0   | 2017-04-24 07:47:00+03 |      3400.00
 01AD57   | 2017-03-15 16:11:00+03 |      3400.00
 020E97   | 2017-01-02 12:25:00+03 |      3400.00
 021FD3   | 2017-05-27 10:20:00+03 |      3400.00
 0278C7   | 2017-02-04 17:42:00+03 |      3400.00
 029452   | 2016-12-10 13:51:00+03 |      3400.00
 0355DD   | 2016-09-19 13:22:00+03 |      3400.00
(10 rows)
```

```
=> EXPLAIN SELECT * FROM bookings WHERE total_amount < 10000;
```

```
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------
 Index Scan using bookings_total_amount_idx on bookings  (cost=0.43..2215.86 rows=63339 width=21)
   Index Cond: (total_amount < '10000'::numeric)
(2 rows)
```

До кластеризации использовалось сканирование по битовой карте, но теперь проще и выгодней сделать обычное индексное сканирование.

Сравнение эффективности



Эффективность сканирования только индекса сильно зависит от актуальности карты видимости (и от того, сколько страниц действительно содержат только актуальные версии строк).

В лучшем случае этот метод доступа может быть эффективней последовательного сканирования даже при низкой селективности (если индекс меньше, чем таблица, и особенно на SSD-дисках).

В худшем случае, когда требуется проверять видимость каждой строки, метод вырождается в обычное индексное сканирование.

Поэтому планировщику приходится учитывать состояние карты видимости: при неблагоприятном прогнозе исключительно индексное сканирование не применяется из-за опасности получить замедление вместо ускорения.

Итоги

Postgres PROFESSIONAL

Сканирование по битовой карте

- исключает повторное чтение табличных страниц
- позволяет объединять несколько индексов
- эффективно при средней селективности

Несколько методов доступа позволяют планировщику выбрать лучший, учитывая различные факторы:

- селективность условия
- необходимость обращения к таблице
- полноту карты видимости
- соответствие порядка данных в таблице и в индексе
- необходимость получить отсортированную выборку
- потребность быстрого получения первых результатов
- и другие

Практика

1. Создайте индекс по столбцу amount таблицы перелетов (ticket_flights).
2. Напишите запрос, находящий информацию о перелетах стоимостью более 180 000 руб. (менее 1 % строк).
   Какой метод доступа был выбран? Сколько времени выполняется запрос?

   Запретите выбранный метод доступа, снова выполните запрос и сравните время выполнения. Прав ли был оптимизатор?
3. Повторите пункт 2 для стоимости менее 44 000 руб. (чуть более 90 % строк).

2. Для запрета метода доступа установите один из параметров в значение off:

- *enable_seqscan*,

- *enable_indexscan*,

- *enable_bitmapscan*.

Например:

```
SET enable_seqscan = off;
```

3. Не забудьте восстановить значение измененного параметра, например:

```
RESET enable_seqscan;
```

1. Индекс

```
=> CREATE INDEX ON ticket_flights(amount);

CREATE INDEX
```

2. Выборка < 1%

```
=> EXPLAIN ANALYZE SELECT * FROM ticket_flights WHERE amount > 180000;

                                                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on ticket_flights  (cost=975.27..69519.65 rows=51979 width=32) (actual time=4.625..19.628 rows=55640 loops=1)
   Recheck Cond: (amount > '180000'::numeric)
   Heap Blocks: exact=1747
   ->  Bitmap Index Scan on ticket_flights_amount_idx  (cost=0.00..962.28 rows=51979 width=0) (actual time=4.471..4.471 rows=55640 loops=1)
         Index Cond: (amount > '180000'::numeric)
 Planning Time: 3.662 ms
 Execution Time: 20.959 ms
(7 rows)
```

Запретим сканирование по битовой карте.

```
=> SET enable_bitmapscan = off;

SET

=> EXPLAIN ANALYZE SELECT * FROM ticket_flights WHERE amount > 180000;

                                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..119838.46 rows=51979 width=32) (actual time=0.127..979.102 rows=55640 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on ticket_flights  (cost=0.00..113640.56 rows=21658 width=32) (actual time=145.205..934.561 rows=18547 loops=3)
         Filter: (amount > '180000'::numeric)
         Rows Removed by Filter: 2778737
 Planning Time: 0.071 ms
 Execution Time: 980.440 ms
(8 rows)
```

Для небольшой выборки сканирование по битовой карте оказывается эффективнее.

```
=> RESET enable_bitmapscan;

RESET
```

3. Выборка > 90%

```
=> EXPLAIN ANALYZE SELECT * FROM ticket_flights WHERE amount < 44000;

                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on ticket_flights  (cost=0.00..174831.15 rows=7581639 width=32) (actual time=7.463..1095.171 rows=7583228 loops=1)
   Filter: (amount < '44000'::numeric)
   Rows Removed by Filter: 808624
 Planning Time: 0.195 ms
 Execution Time: 1266.765 ms
(5 rows)
```

Запретим последовательное сканирование.

```
=> SET enable_seqscan = off;

SET

=> EXPLAIN ANALYZE SELECT * FROM ticket_flights WHERE amount < 44000;

                                                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on ticket_flights  (cost=142038.14..312123.84 rows=7581639 width=32) (actual time=553.875..1512.979 rows=7583228 loops=1)
   Recheck Cond: (amount < '44000'::numeric)
   Rows Removed by Index Recheck: 296939
   Heap Blocks: exact=36281 lossy=33034
   ->  Bitmap Index Scan on ticket_flights_amount_idx  (cost=0.00..140142.73 rows=7581639 width=0) (actual time=548.376..548.377 rows=7583228 loops=1)
         Index Cond: (amount < '44000'::numeric)
 Planning Time: 0.075 ms
 Execution Time: 1684.882 ms
(8 rows)
```

Для большой выборки полное сканирование выгоднее.

Следует также учесть, что повторно запрос обычно выполняется быстрее из-за кеширования.