

++ 2017 PostgreSQL | Илья Космодемьянский - Производительность запросов в PostgreSQL

https://www.youtube.com/watch?v=c-ySk8COI1c&ab_channel=HighLoadChannel

Важное

1. Чтобы включить pg_stat_statements на вашем сервере, измените следующую строку в postgresql.conf и перезапустите PostgreSQL: shared_preload_libraries = 'pg_stat_statements' Затем выполните следующую команду для создания представления, необходимого для доступа к данным: **CREATE EXTENSION pg_stat_statements;**
SELECT * FROM pg_stat_statements ORDER BY total_time DESC;
2. Оптимизация может потреблять большое количество ресурсов. Допустим нужно сделать join больше чем по трем таблицам. Тогда сначала выполнится джоин с первой таблицей связи, потом resultSet сджоинится со второй. Если в запросе 512 джоинов, то постгрес будет вычислять какой порядок будет оптимальным. Чтобы перебрать все варианты ему потребуется сделать 512! сравнений, среди которых выбрать оптимальный способ.
3. Индекс при каждом добавлении/изменении - добавляется/балансируется. С большой вероятностью удаление лишних индексов приведет к ускорению производительности
4. Count работает очень медленно. При выполнении count postgres проходит таблицу целиком, а также проверяет актуальна ли эта версия данных или ее уже обновили.
5. Where in(1...10000) работает очень медленно, проблема решается join (values(1...10000))
6. Как быстрее обновлять записи? Postgres под капотом выполняет delete и insert


В докладе не будет конкретных примеров оптимизации запросов.

Почти все говорят смотрите Explain. Но не понятно на что обращать внимание. В докладе рассказывается про это.

- Что значит оптимизировать запросы?
- Когда начинать оптимизировать запросы?
- Какие запросы оптимизировать?
- Как оптимизировать запросы?
- Какие запросы бесполезно оптимизировать?

Обычно все проблемы сводятся к трем.

1. В базе данных лежит что-то не нужно.
2. Не лежит нужного.
3. База данных используется не правильно


Что значит оптимизировать запросы?

- Задача редко ставится в такой форме
- «Всё плохо»- более типичная постановка
- Медленные запросы просто видны наружу
- На ненастроенной базе запросы оптимизировать бесполезно


Что значит не настроена база данных:

1. Не включен автовакуум. Без него очень большая фрагментация таблицы. Таблица на 100к записей по размеру может быть как таблица на 100м записей.
2. 100500 подключений к базе без балансера. На одно подключение должно быть ядро
3. Правильные настройки памяти
4. Правильные настройки диска

Только после этого можно приступить к оптимизации запросов.

Алгоритм


- Проверить настройки
- Отобрать запросы для оптимизации
- Оптимизировать запросы
- Повторить



Какие запросы оптимизировать?

- Все подряд - бесполезно
- Отобрать top
- pg_stat_statements

Были разработаны утилиты позволяющие генерировать отчеты в виде. Запрос, сколько времени отнял, и какая нагрузка на него пришлась. Если запрос отнимает много времени, но не является одним из основных, то его стоит оптимизировать.





Изучаем top

```

total time: 06:01:33 (ID: 1.15%)
total queries: 896,507,876 (unique: 377)
report for all databases, version 0.9.2
=====
pos:1 total time: 01:27:08 (24.1%, CPU: 24.4%, ID: 0.0%) calls: 1,038,397 (0.12%) avg_time: 5.03ms (ID: 0.0%)
user: someuser db: somedb rows: 1,754,836 query:
select q.id, q.type, q.srvvertime, q.clienttime from foo q left outer join bar qq ON q.buzzid = qq.buzzid where qq.yaid
=====
pos:2 total time: 00:28:56 (8.0%, CPU: 8.1%, ID: 0.0%) calls: 120,782,681 (13.47%) avg_time: 0.01ms (ID: 0.0%)
user: someuser db: somedb rows: 557,454,570 query:
select seqid, id, name, notes from bar where pvid = $1 order by id asc


```

https://github.com/PostgreSQL-Consulting/pg-utils/tree/master/sql/global_reports




Что такое "запрос работает медленно?"

- Запрос работает 123,0 ms
- Запрос работает 7300,0 ms
- Запрос работает 3 min
- Запрос работает 0.12 ms




Более правильный подход

- Как часто исполняется запрос
- Какое время отклика мы считаем приемлемым
- Каков характер нагрузки на базу?
- Какой процент ресурсов базы занимает данный запрос

С помощью `pg_stat_statements` можно получить топ запросов в определенный промежуток времени, это позволит найти тяжелые запросы, например, для аналитики и вынести их во время когда нагрузка на баз меньше. Запрос для фронта должен иметь очень быстрое время отдачи, допустим, если время которое срабатывает запрос 1с, то пользователь точно получит данные не раньше секунды.

Оптимизация может потреблять большое количество ресурсов. Допустим нужно сделать `join` больше чем по трем таблицам. Тогда сначала выполнится джоин с первой таблицей связи, потом `resultSet` сджоинится со второй.

Если в запросе 512 джоинов, то постгрес будет вычислять какой порядок будет оптимальным. Чтобы перебрать все варианты ему потребуется сделать 512! сравнений, среди которых выбрать оптимальный способ.




На чем расходуется время при выполнении запроса

- Передача запроса от клиента (НЕ СМЕШНО!)
- Парсинг
- Оптимизация
- Исполнение
- Возврат результатов

После получения проблемных запросов нужно выполнить команду `Explain`

`Explain` проанализирует запрос

`Explain Analyze` выполнит и покажет план запроса



EXPLAIN

```

PREPARE query(int, int) AS SELECT sum(bar) FROM test WHERE id > $1 AND id < $2 GROUP BY foo;
EXPLAIN ANALYZE EXECUTE query(100, 200);

```

QUERY PLAN

```

HashAggregate  (cost=9.54..9.54 rows=1 width=8) (actual time=0.156..0.161 rows=1 loops=1)
  Group Key: foo
    -> Index Scan using test_pkey on test  (cost=0.29..9.29 rows=50 width=8) (actual time=0.039..0.091 rows=99 loops=1)
        Index Cond: ((id > $1) AND (id < $2))
Planning time: 0.197 ms
Execution time: 0.225 ms
(6 rows)

```


pg@pgiect01:~/dellstore2-normal-1.0> psql dellstore2 -c "explain (analyze on, buffers on) select count(*) from customers"

QUERY PLAN

```


Aggregate  (cost=738.00..738.01 rows=1 width=0) (actual time=8.357..8.357 rows=1 loops=1)
  Buffers: shared hit=488
    -> Seq Scan on customers  (cost=0.00..688.00 rows=20000 width=0) (actual time=0.026..5.728 rows=20000 loops=1)
        Buffers: shared hit=488
Total runtime: 8.607 ms
(5 rows)

```



В результате запроса cost - время затрачиваемое на получение одного блока размером размером 8кб в sequentialScan. Эта величина зависит от машины и ее можно считать условной. Cost 9.54 значит, что исполнить текущий запрос будет в 9.54 раза медленнее, чем достать один блок размером 8кб. Первая цифра - сколько будет потрачено времени до возврата первых результатов, вторая - время до возврата всего. Рядом с cost указано время исполнения. Если cost маленький, а время большое, нужно проверить что некорректно работает при сборе статистики, возможно отключен автовакуум.

Индексы



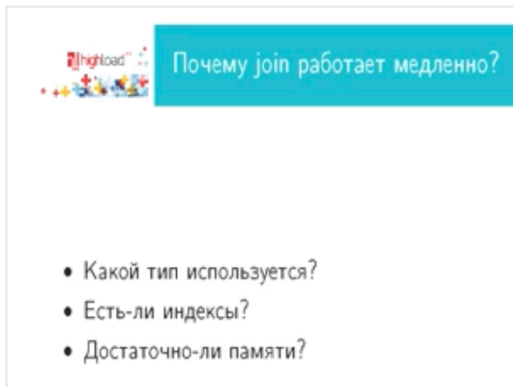
Почему не используется индекс?

- Собрана-ли статистика?
- Ускорит-ли индекс исполнение запроса?
- Правильно-ли написан запрос? (where counter + 1 = 46)

Индекс при каждом добавлении/изменении - добавляется/балансируется. С большой вероятностью удаление лишних индексов приведет к ускорению производительности. Если нужно получить большую часть таблицы, то индекс использоваться не будет. SequentialScan всегда быстрее, чем IndexScan. Для того, чтобы проверить как работает быстрее, можно написать

enableIndexScan. Или EnableSequentialScan.

Join



В Postgres Join работает гораздо быстрее, чем в MySQL.

Алгоритмы Join

1. NestedLoop – берем данные из одной таблицы и циклами их джоиним
2. HashIndex – одна маленькая табличка хэштрруется и по этому хэшу джоинится с другой таблицей
3. MergeJoin

Оптимизатор выбирает тип Join. Не всегда может быть выбран оптимальный вариант. Если поля по которым выполняется join посечены индексами, то выбирается тип NestedLoop, который работает быстрее.

Если одна табличка маленькая, а оптимизатор выбирает NestedLoop, правильнее было бы выбрать HashIndex, прохэшировать маленькую табличку и пройтись по ней. Но может получиться так, что для воркера установлено ограничение 30мб, а хэширование этой таблички занимает 50мб. Если увеличить лимит памяти воркерам, оптимизатор будет выбирать правильный вариант.

Orm любят большие In, но они работают медленно.

С слайда ниже можно заметить что where id<10000 выполняется 28 мс, а In(от 1 до 10000) 380мс



Длинный IN (очень любят ORM)

```

1. SELECT * FROM test WHERE id<10000
1.2ms

2. SELECT * FROM test WHERE id<10000 AND val IN (список от 1 до 10)
2.1ms

3. SELECT * FROM test WHERE id<10000 AND val IN (список от 1 до 100)
6ms

4. SELECT * FROM test WHERE id<10000 AND val IN (список от 1 до 1000)
38ms

5. SELECT * FROM test WHERE id<10000 AND val IN (список от 1 до 10000)
380ms (в 30 раз медленнее от данных таблиц)

```

PostgreSQL-Consulting.com



IN (1,...100)

```

explain analyze select * from test where id<10000 and val IN (1,...100);
QUERY PLAN
-----
Index Scan using test_pkey on test (cost=0.43..1666.85 rows=10
width=140) (actual time=0.448..5.602 rows=16 loops=1)
Index Cond: (id < 10000)
Filter: (val = ANY ('{1,...100}'::integer[]))
Rows Removed by Filter: 9984

```

PostgreSQL-Consulting.com

Запрос можно переписать таким образом



Длинный IN (hash join)

```

explain select count(*) from test JOIN (VALUES (1),...,(10)) AS v(val) USING (val) where id<10000;
QUERY PLAN
-----
Aggregate (cost=497.65..497.66 rows=1 width=0)
-> Hash Join (cost=0.69..497.65 rows=1 width=0)
Hash Cond: (test.val = "VALUES".column1)
-> Index Scan using test_pkey on test (cost=0.43..461.22
rows=9645 width=4)
Index Cond: (id < 10000)
-> Hash (cost=0.12..0.12 rows=10 width=4)
-> Values Scan on "VALUES" (cost=0.00..

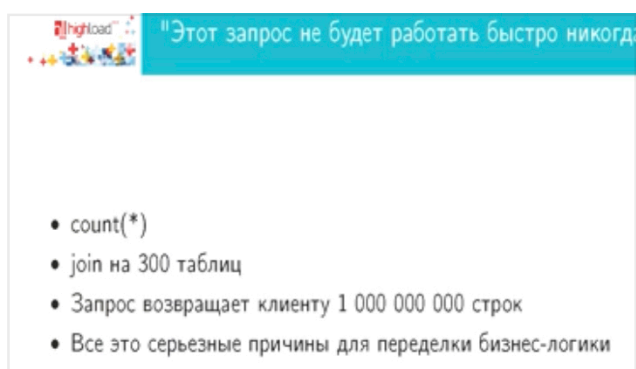
```

PostgreSQL-Consulting.com

Можно заметить, что заменив In на join получается очень хороший результат выполнения.



Часть запросов никогда не будет работать быстро



Count работает очень медленно. При выполнении count postgres проходит таблицу целиком, а также проверяет актуальна ли эта версия данных или ее уже обновили.

Как быстрее обновлять записи?

Postgres под капотом выполняет delete и insert