

Как тестирование показывает слабые стороны СУБД

Чувашов И.Н.

Чувашов Иван



SQL разработчик – 16 лет

Архитектор баз данных – 10 лет

DBA PostgreSQL – 8 лет

Сертификат «Администратор PostgreSQL. Эксперт»
(DBA1-10, DBA2-10, DBA3-10, QPT-10).

На данный момент работаю в компании SoftSwiss

Преподаватель курсов «Основы GreenPlum»,

«Оптимизация SQL запросов» и «Резервное

копирование, высокая доступность и траблшутинг в

PostgreSQL»

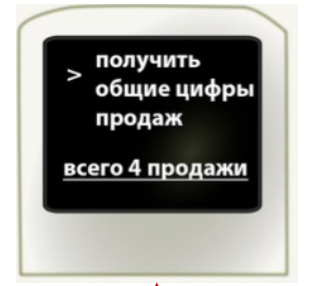
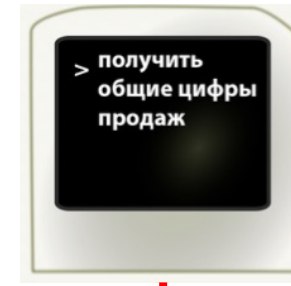
Цель

Из своего личного опыта, рассказать как в базе данных ищут слабые места производительности и какими инструментами можно для этого использовать

Трёхуровневая архитектура

Слой клиента

Самый верхний уровень приложения с интерфейсом пользователя

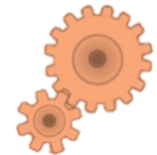


Слой логики

Этот слой координирует программу, обрабатывает команды, выполняет логические решения и вычисления, выполняет расчеты.



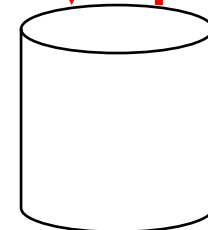
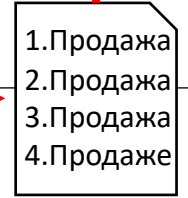
Получить список продаж за прошлый год



Объединить все продажи вместе

Слой данных

Здесь хранится информация и извлекается из базы данных и файловой системы.



База данных

Трёхуровневая архитектура

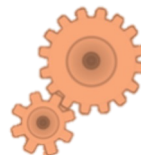
Слой клиента

Самый верхний уровень приложения с интерфейсом пользователя.

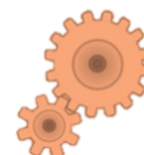


Слой логики

Этот слой координирует программу, обрабатывает команды, выполняет логические решения и вычисления, выполняет расчеты.



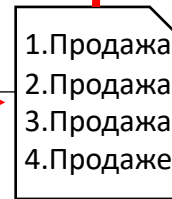
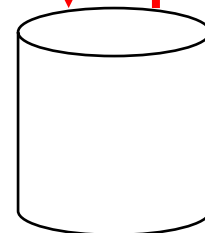
Получить список продаж за прошлый год ...



Объединить все продажи вместе

Слой данных

Здесь хранится информация и извлекается из базы данных и файловой системы.



База данных

Трёхуровневая архитектура

Слой клиента

Самый верхний уровень приложения с интерфейсом пользователя.



Слой логики

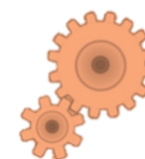
Этот слой координирует программу, обрабатывает команды, выводит логику

логику
вы

База Данных ОДНА!



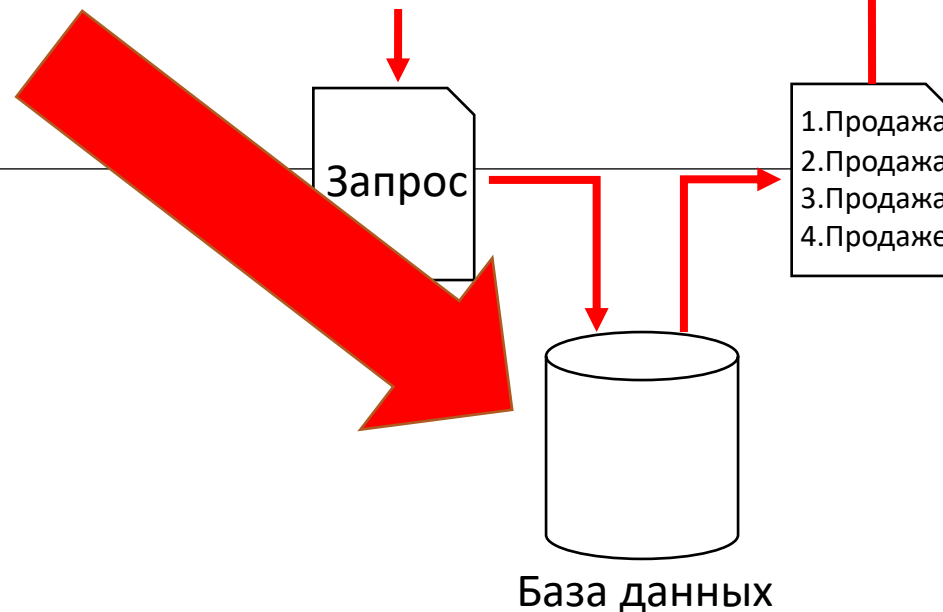
Получить
список продаж
... прошлый год ...



Объединить
все продажи
вместе

Слой данных

Здесь хранится информация и извлекается из базы данных и файловой системы.



Методы оптимизации

Расширения аппаратных ресурсов

~20% - 30%

Оптимизация конфигурации СУБД

~10% - 20%

Ускорение запросов

~10% - 20%

Изменение логики работы с данными

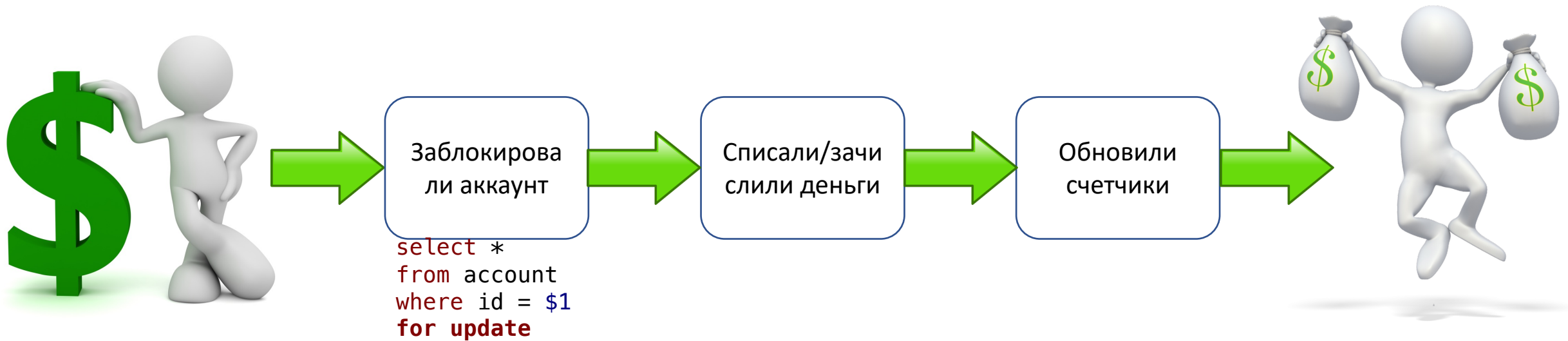
~30% - 60%

Практический случай с lock

Вводная

Есть платежная система

Обработка транзакции списания денег



Какие тут есть проблемы?

Практический случай с lock

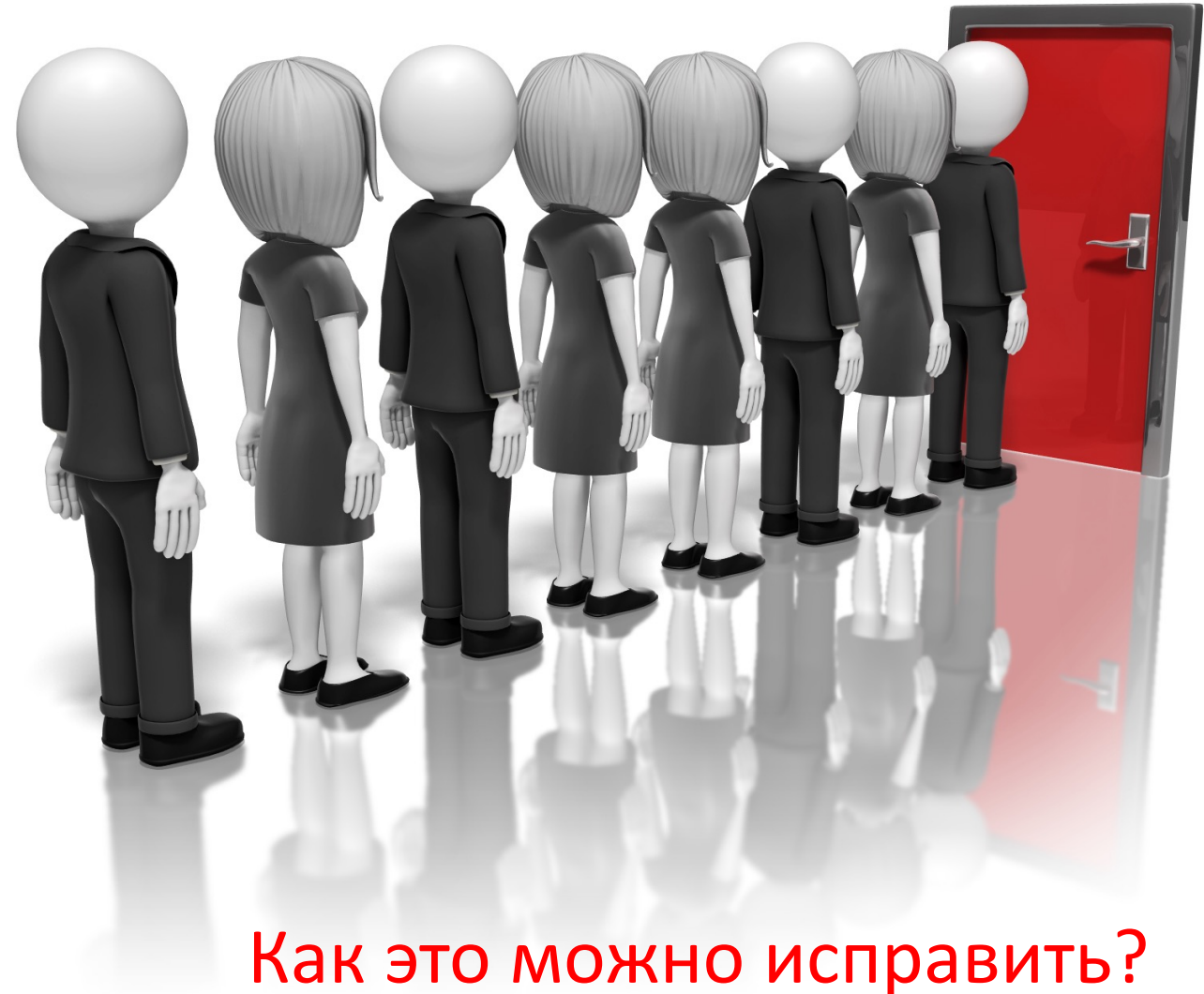
Проблема

Все хорошо работает в
однопользовательском режиме

Все плохо, когда с одного
аккаунта совершается множество
одновременных переводов

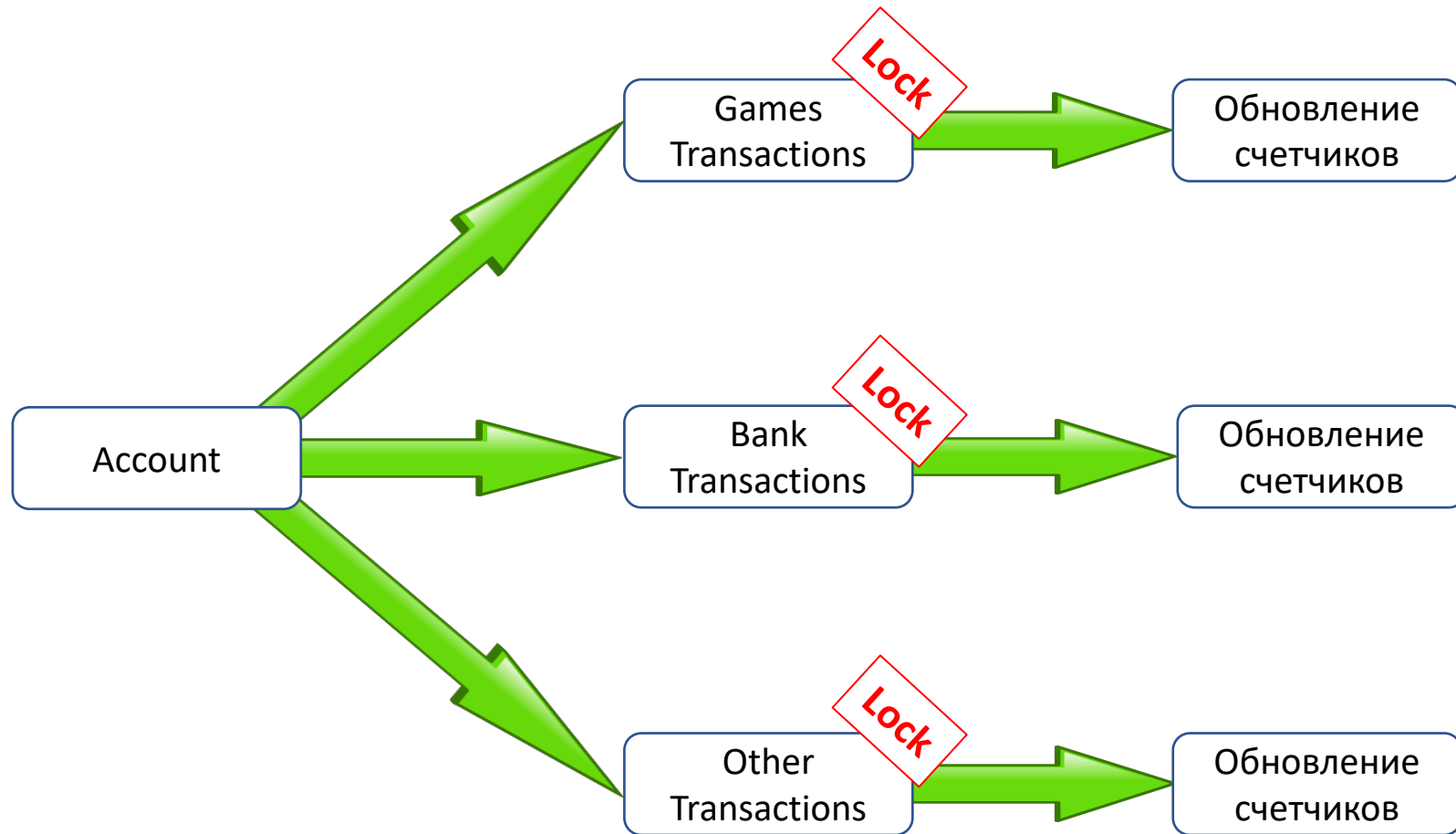
Все сильно портит команда

```
select * from account where  
id = $1 for update
```



Как это можно исправить?

Практический случай с lock



Решение проблемы
На время

Медленные запросы

```
create table users
(
    id bigint primary key,
    login varchar(200) not null,
    first_name varchar(200) not null,
    last_name varchar(200) not null,
    create_date timestamp not null default now()
);
```

CREATE TABLE

```
insert into users
select id, random() * id, md5(sin(id)::text), md5(cos(id)::text)
from generate_series(1, 10000000) id;
INSERT 0 10000000
```

```
analyze users;
ANALYZE
```

Медленные запросы

```
\timing
Timing is on.
select count(distinct id) from
users;
   count
-----
10000000
(1 row)

Time: 4535.931 ms (00:04.536)
```

Медленные запросы

```
explain select count(distinct id) from users;  
QUERY PLAN
```

```
Aggregate (cost=176230.12..176230.14 rows=1 width=8)  
-> Seq Scan on users (cost=0.00..173771.10 rows=983610 width=8)  
(2 rows)  
Time: 0.435 ms
```



Это не время работы запроса, а время планирования запроса

Медленные запросы

```
explain analyze select count(distinct id) from users;
```

```
QUERY PLAN
```

```
Aggregate (cost=284689.43..284689.45 rows=1 width=8) (actual  
time=4228.384..4228.386 rows=1 loops=1)  
  -> Index Only Scan using users_pkey on users (cost=0.43..259689.43  
rows=10000000 width=8) (actual time=1.006..1224.245 rows=10000000 loops=1)  
    Heap Fetches: 61  
    Planning Time: 0.079 ms  
    Execution Time: 4229.148 ms  
(5 rows)
```

Медленные запросы

```
explain (analyze, buffers) select count(distinct id) from users;
```

```
QUERY PLAN
```

```
Aggregate (cost=284689.43..284689.45 rows=1 width=8) (actual time=4408.248..4408.262  
rows=1 loops=1)
```

```
  Buffers: shared read=27332, temp read=30325 written=30382
```

```
    -> Index Only Scan using users_pkey on users (cost=0.43..259689.43 rows=10000000  
width=8) (actual time=0.084..1253.347 rows=10000000 loops=1)
```

```
      Heap Fetches: 61
```

```
      Buffers: shared read=27332
```

```
Planning Time: 0.081 ms
```

```
Execution Time: 4408.414 ms
```

```
(7 rows)
```

Медленные запросы

```
explain (analyze, buffers) select count(id) from users;
```

```
QUERY PLAN
```

```
Finalize Aggregate (cost=212772.98..212772.99 rows=1 width=8) (actual time=1093.250..1093.496 rows=1 loops=1)
  Buffers: shared hit=12 read=27332
  -> Gather (cost=212772.77..212772.98 rows=2 width=8) (actual time=1089.207..1093.486 rows=3 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    Buffers: shared hit=12 read=27332
    -> Partial Aggregate (cost=211772.77..211772.78 rows=1 width=8) (actual time=1061.481..1061.484 rows=1
loops=3)
      Buffers: shared hit=12 read=27332
      -> Parallel Index Only Scan using users_pkey on users (cost=0.43..201356.10 rows=4166667 width=0)
(actual time=0.050..759.141 rows=3333333 loops=3)
        Heap Fetches: 61
        Buffers: shared hit=12 read=27332

Planning:
  Buffers: shared read=3
Planning Time: 1.685 ms
Execution Time: 1093.589 ms
(15 rows)
```


Медленные запросы

```
explain (analyze, buffers) select count(1) from users;
```

```
QUERY PLAN
```

```
Finalize Aggregate (cost=212772.98..212772.99 rows=1 width=8) (actual time=1093.250..1093.496 rows=1 loops=1)
  Buffers: shared hit=12 read=27332
  -> Gather (cost=212772.77..212772.98 rows=2 width=8) (actual time=1089.207..1093.486 rows=3 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    Buffers: shared hit=12 read=27332
    -> Partial Aggregate (cost=211772.77..211772.78 rows=1 width=8) (actual time=1061.481..1061.484 rows=1
loops=3)
      Buffers: shared hit=12 read=27332
      -> Parallel Index Only Scan using users_pkey on users (cost=0.43..201356.10 rows=4166667 width=0)
(actual time=0.050..759.141 rows=3333333 loops=3)
        Heap Fetches: 61
        Buffers: shared hit=12 read=27332

Planning:
  Buffers: shared read=3
Planning Time: 1.685 ms
Execution Time: 1093.589 ms
(15 rows)
```

Медленные запросы

```
echo "select count(1) from users;" | pgbench -d test -t 50 -P 1 -f -
```

```
latency average = 602.805 ms  
latency stddev = 34.097 ms  
tps = 1.658885 (without initial connection time)
```

```
echo "select count(*) from users;" | pgbench -d test -t 50 -P 1 -f -
```

```
latency average = 593.256 ms  
latency stddev = 30.989 ms  
tps = 1.685587 (without initial connection time)
```

Хотите проверить?



А что делать если мне лень

искать ЭТИ запросы

оптимизировать ЭТИ запросы

и вообще я не люблю SQL

...

Но ускорить БД мне нужно!!!

Индексы!!!



Знакомьтесь, расширение `pg_qualstats`

`pg_qualstats` — это расширение PostgreSQL, хранящее статистику по предикатам, найденным в операторах `WHERE` и предложениях `JOIN`.

Расширение pg_qualstats

Рассмотрим на примере

```
yum install pg_qualstats_12
```

```
echo "shared_preload_libraries = 'pg_stat_statements, pg_qualstats'" >>  
/var/lib/pgsql/12/data/postgresql.conf
```

```
service postgresql-12 restart
```

```
psql -c "CREATE EXTENSION pg_qualstats"
```

Расширение pg_qualstats

```
pgbench -l dtgv -s 100 -i
```

```
dropping old tables...
```

```
creating tables...
```

```
generating data...
```

```
100000 of 10000000 tuples (1%) done (elapsed 0.09 s, remaining 9.10 s)
```

```
...
```

```
10000000 of 10000000 tuples (100%) done (elapsed 12.25 s, remaining 0.00 s)
```

```
vacuuming...
```

```
done.
```

Про ключи

-i создания шаблонных таблиц для проведения тестирования

-l

dt пересоздание таблиц

g вставка данных

v сбор статистики

-s множить число генерируемых строк на заданный коэффициент

Расширение pg_qualstats

```
psql -c "\dt+"
```

```
          List of relations
Schema |          Name          | Type  | Owner  | Size  | Description
-----+-----+-----+-----+-----+-----
public | pgbench_accounts      | table | postgres | 1281 MB |
public | pgbench_branches      | table | postgres | 128 kB  |
public | pgbench_history       | table | postgres | 192 kB  |
public | pgbench_tellers       | table | postgres | 160 kB  |
(4 rows)
```

Расширение pg_qualstats

```
psql -c "\d+ pgbench_accounts"
```

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
aid	integer		not null		plain		
bid	integer				plain		
abalance	integer				plain		
filler	character(84)				extended		

Access method: heap
Options: fillfactor=100

Расширение pg_qualstats

```
pgbench -c 10 -j 10 -T 600
```

...

client 4 sending SELECT abalance FROM pgbench_accounts WHERE aid = 2318194;

...

client 6 sending UPDATE pgbench_accounts SET abalance = abalance + 499 WHERE aid = 863112;

...

client 8 sending UPDATE pgbench_tellers SET tbalance = tbalance + -4160 WHERE tid = 300;

...

client 8 sending UPDATE pgbench_branches SET bbalance = bbalance + -4160 WHERE bid = 6;

...

client 8 sending INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (300, 6, 4197917, -4160, CURRENT_TIMESTAMP);

...

ждем 10 минут

Расширение pg_qualstats

psql

```
SELECT v
FROM json_array_elements(pg_qualstats_index_advisor(min_filter =>
50)->'indexes') v
ORDER BY v::text COLLATE "C"
```

```
"CREATE INDEX ON public.pgbench_accounts USING btree (aid)"
"CREATE INDEX ON public.pgbench_branches USING btree (bid)"
"CREATE INDEX ON public.pgbench_tellers USING btree (tid)"
```

Хотите проверить?



Рекомендации

База данных выполняет только те запросы, которые отправило ему приложение.

Для поиска «тяжелых» или частных запросов лучше всего использовать расширение **pg_stat_statements**

Если не видите проблемы с запросами, то установите расширение **pg_qualstats**

Если не умеете считать логи PostgreSQL, то воспользуйтесь **pgbadger**