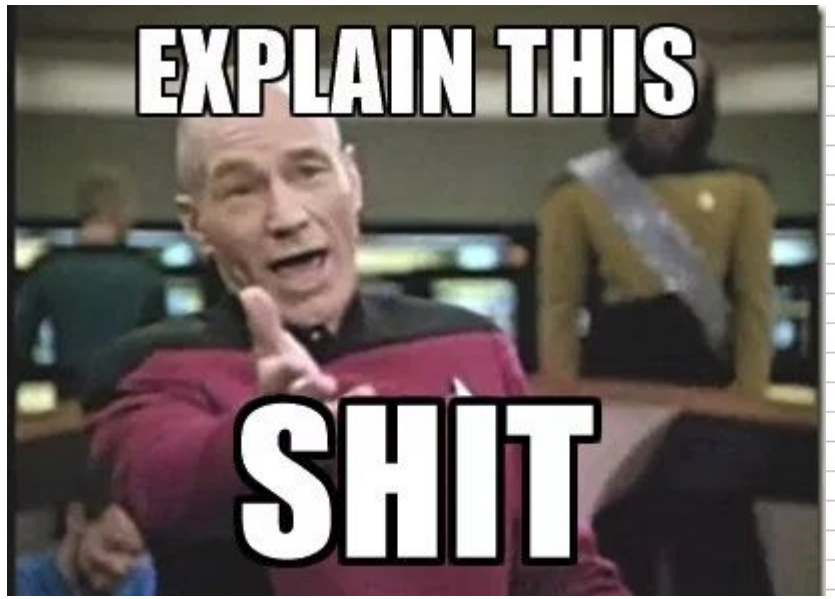
Как получить нагрузку на пустом месте

Нечто невероятное где-то живёт,
чтобы мы его узнали

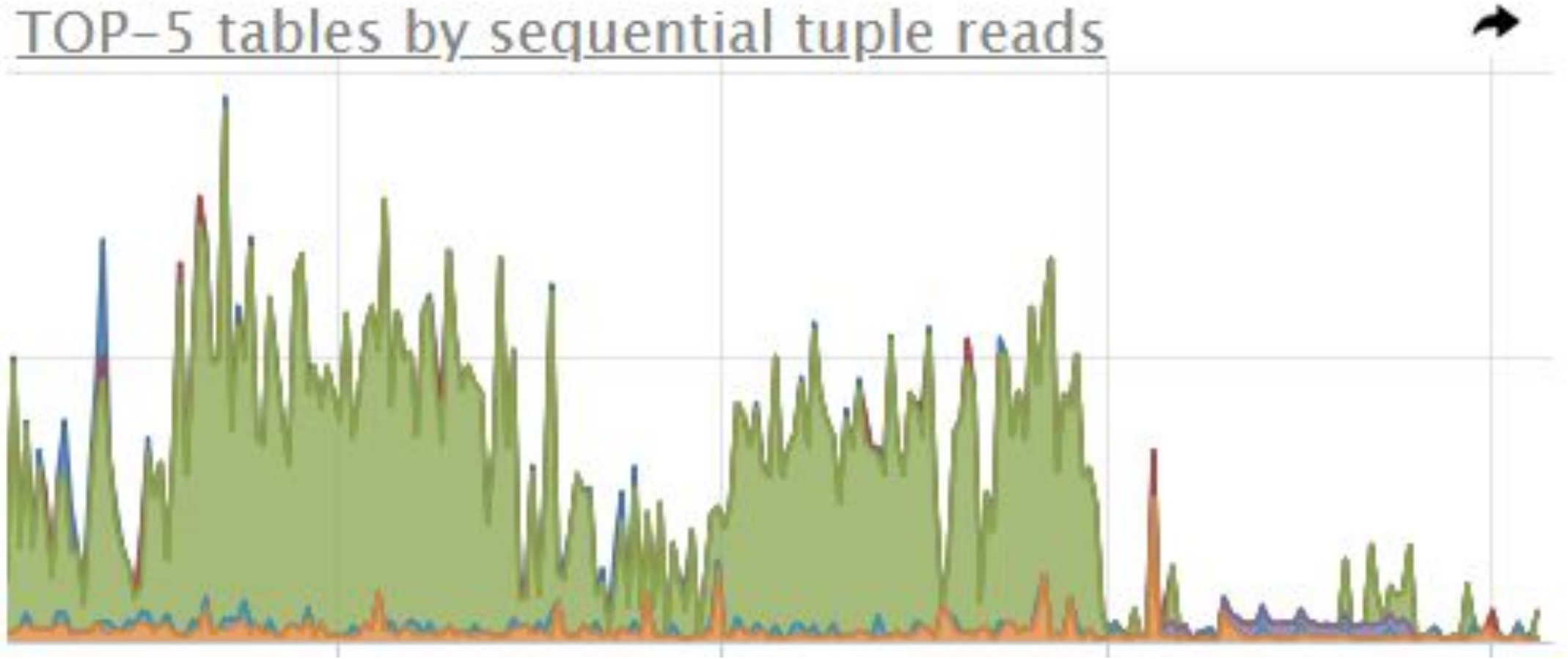
```
Subquery Scan on t  (cost=74013.47..74014.39 rows=1 width=558)
  Filter: ((t.category = 'Important'::text) OR ((t.rn > 0) AND (t.rn <= 10)))
  ->  WindowAgg  (cost=74013.47..74013.99 rows=23 width=538)
        ->  Sort  (cost=74013.47..74013.53 rows=23 width=526)
              Sort Key: sq.category, sq.createdate DESC
              ->  WindowAgg  (cost=74012.55..74012.95 rows=23 width=526)
                    ->  Sort  (cost=74012.55..74012.60 rows=23 width=518)
                          Sort Key: sq.category
                          ->  Subquery Scan on sq  (cost=73711.10..74012.03 rows=23 width=518)
                                ->  Nested Loop  (cost=73711.10..74011.80 rows=23 width=722)
                                      CTE rusers
                                        ->  Index Scan using userid on users u 2  (cost=0.42..13.44 rows=1000 width=16)
                                              Index Cond: (id = 'fbcdeabc-69de-40a9-868d-4179c6cb3c74'::uuid)
                                      CTE rcompanies
                                        ->  Index Scan using companyid on companies u 3  (cost=0.42..13.44 rows=1000 width=16)
                                              Index Cond: (id = '1086371a-6827-4714-b239-0d60118fa28d'::uuid)
                                      ->  Group  (cost=73683.66..73683.83 rows=35 width=16)
                                            Group Key: events 1.id
                                            ->  Sort  (cost=73683.66..73683.75 rows=35 width=16)
                                                  Sort Key: events 1.id
                                                  ->  Nested Loop  (cost=14103.82..73682.76 rows=35 width=16)
                                                        ->  HashAggregate  (cost=14103.26..14173.26 rows=7000 width=20)
                                                              Group Key: e.id, (1)
                                                              ->  Append  (cost=0.56..14068.26 rows=7000 width=20)
                                                                    ->  Nested Loop  (cost=0.56..4654.00 rows=1000 width=20)
                                                                          ->  CTE Scan on rusers u  (cost=0.00..20.00 rows=1000 width=16)
                                                                          ->  Index Only Scan using events pkey on events e  (cost=0.56..4.62 rows=1 width=16)
                                                                                Index Cond: (id = u.id)
                                                                    ->  Nested Loop  (cost=0.56..4654.00 rows=1000 width=20)
                                                                          ->  CTE Scan on rcompanies c  (cost=0.00..20.00 rows=1000 width=16)
                                                                          ->  Index Only Scan using events pkey on events e 1  (cost=0.56..4.62 rows=1 width=16)
                                                                                Index Cond: (id = c.id)
                                                                    ->  Hash Join  (cost=4666.50..4690.26 rows=5000 width=20)
                                                                          Hash Cond: (c 1.id = e 2.id)
                                                                          ->  CTE Scan on rcompanies c 1  (cost=0.00..20.00 rows=1000 width=16)
                                                                          ->  Hash  (cost=4654.00..4654.00 rows=1000 width=32)
                                                                                ->  Nested Loop  (cost=0.56..4654.00 rows=1000 width=32)
                                                                                      ->  CTE Scan on rusers u 1  (cost=0.00..20.00 rows=1000 width=16)
                                                                                      ->  Index Only Scan using events pkey on events e 2  (cost=0.56..4.62 rows=1 width=16)
                                                                                            Index Cond: (id = u 1.id)
                                                        ->  Index Scan using events pkey on events events 1  (cost=0.56..8.48 rows=1 width=20)
                                                              Index Cond: (id = e.id)
                                                              Filter: ((1) = targettype)
                                      ->  Index Scan using events pkey on events  (cost=0.56..8.58 rows=1 width=518)
                                            Index Cond: (id = events 1.id)
                                            Filter: (NOT exist(hideforusers, '4d8db63c-1d6a-11e5-9232-fbb76448b377'::text))
```



SeqScan

```
SELECT t.col
FROM (
        SELECT svals(hstore)::json col
        FROM   table1
) t
WHERE  t.col->>'Token' = :token
```

SeqScan



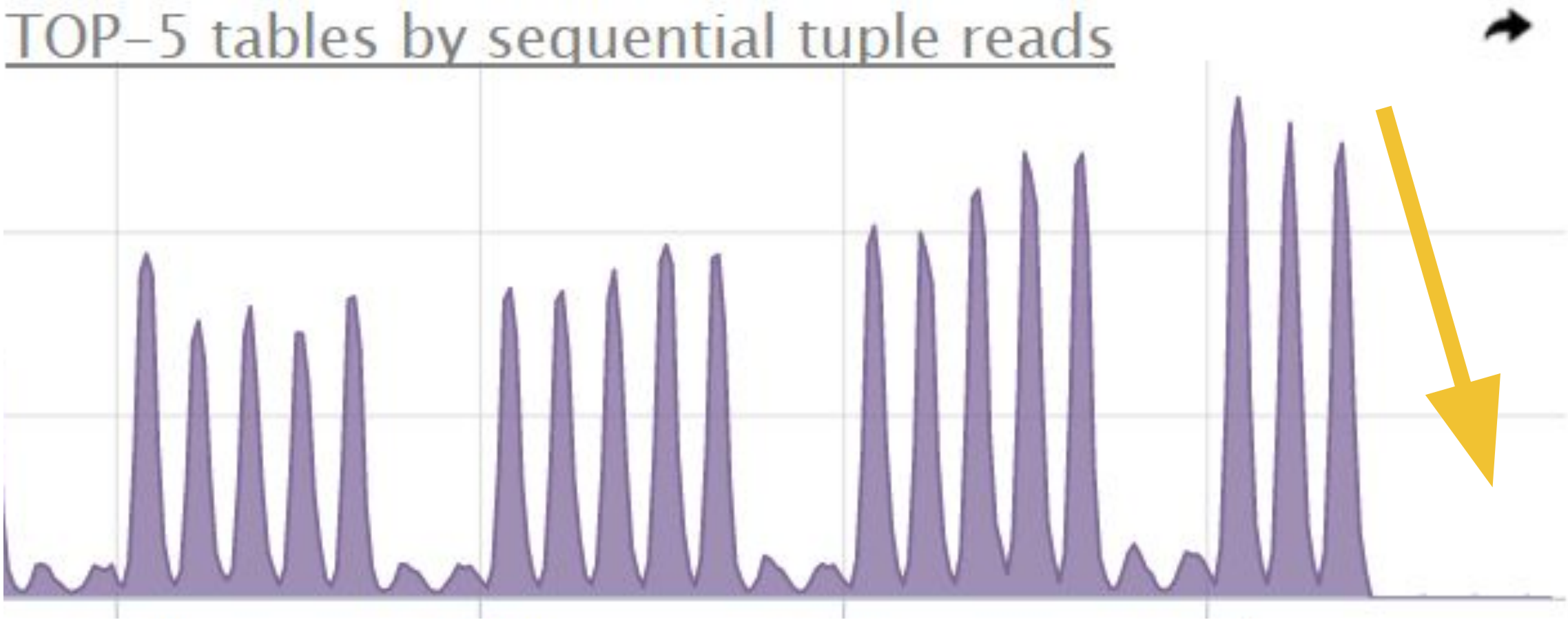
CTE

```
WITH src AS (
   SELECT *
   FROM documents
)
SELECT *
FROM src
WHERE docid=:id
```

CTE

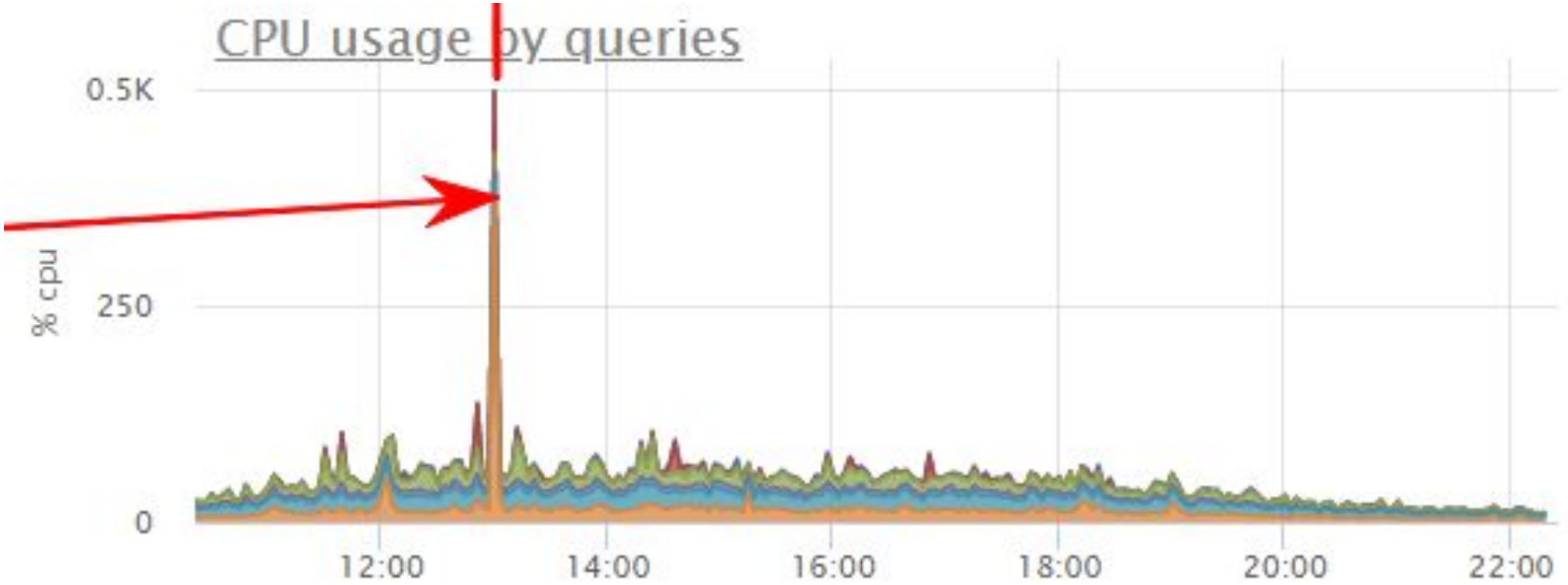
```
WITH src AS (
   SELECT *
   FROM documents
    WHERE docid=:id
)
SELECT *
FROM src
```

CTE + BTree



Безусловные обновления

```
UPDATE systemnews
SET    users = users - array[:uid]
```

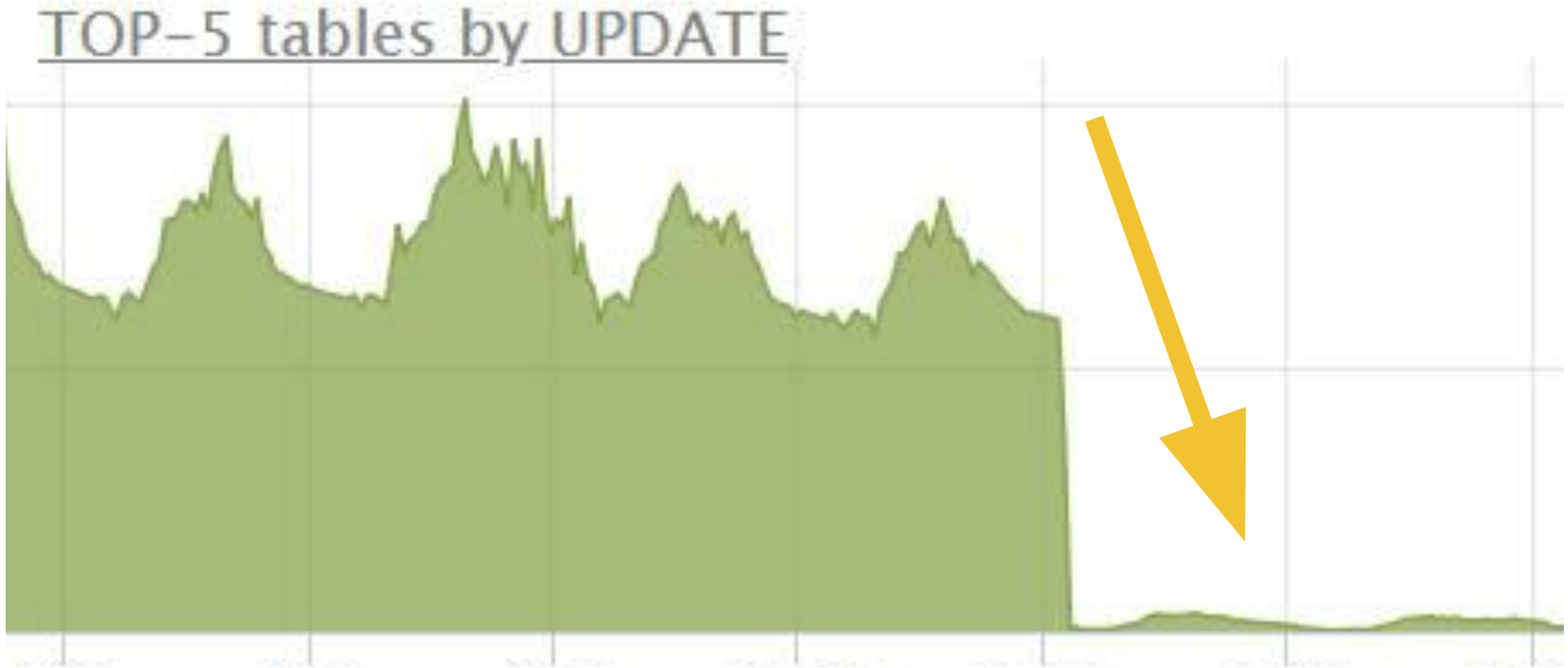


Обновлять не обновляемое

```
UPDATE table1
SET    content = :newcontent
WHERE  id = :id;
```

Обновлять не обновляемое

```
UPDATE table1
SET    content = :newcontent
WHERE  id = :id and content != :newcontent;
```

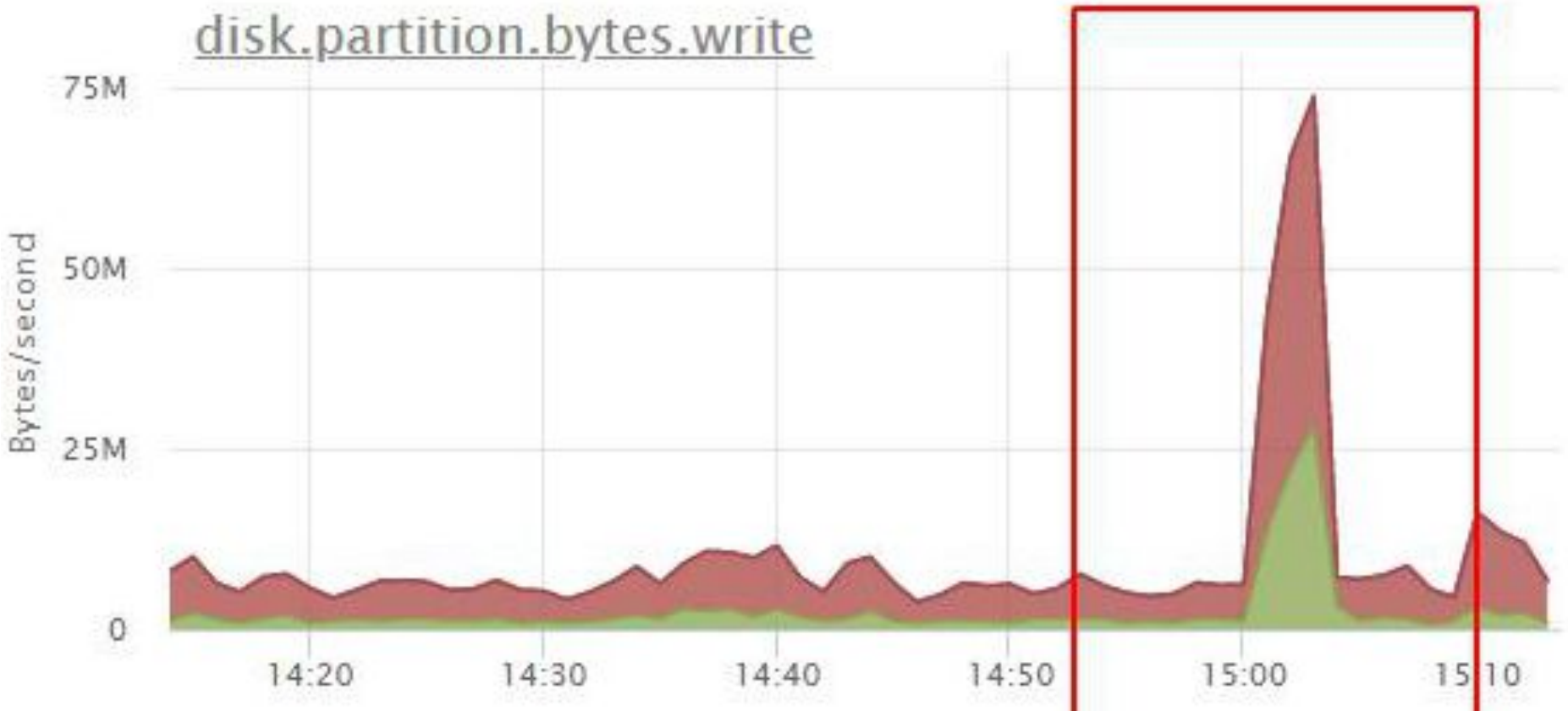


Любовь к hstore

```
UPDATE table1
SET    users = users || hstore(:uid, null)
WHERE  id = :id;
```

Любовь к hstore

```
UPDATE table1
SET    users = users || hstore(:uid, null)
WHERE  id = :id;
```



Слишком большой IN

```
SELECT *
FROM   table1
WHERE  companyid = :cid
       AND groupname IN (0..>1000 ids);
```

Слишком большой IN

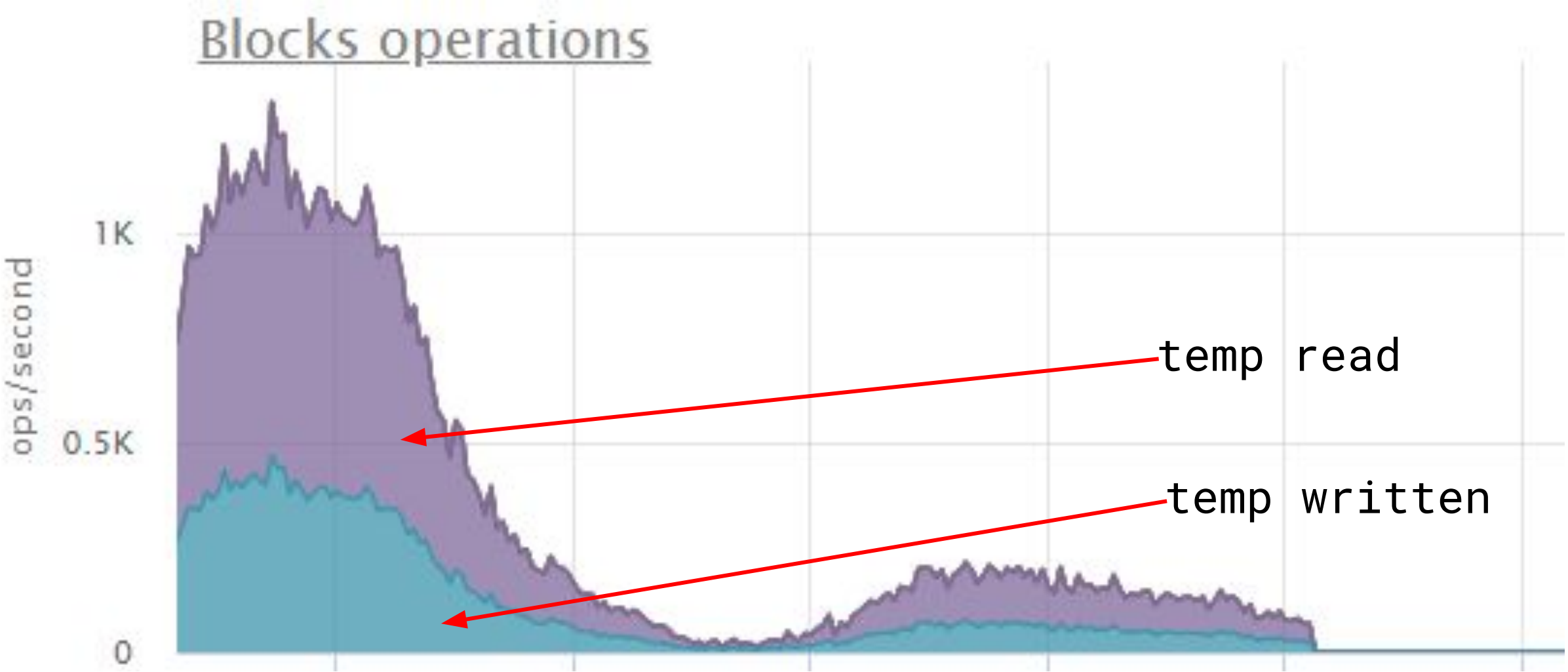
```
SELECT *
FROM   table1
WHERE  companyid = :cid
       AND groupname IN (0..>1000 ids);

CREATE INDEX ON table1
   (companyid, groupname)
WHERE groupname is not null;
```

Неправильная структура

```
CREATE TABLE events (
    id              UUID NOT NULL,
    companies       HSTORE,
    users           HSTORE,
    read            HSTORE,
    hideforusers    HSTORE,
    eventdata       JSON,
    created         TIMESTAMP,
    CONSTRAINT events_pkey PRIMARY KEY (id)
)
```

Неправильный запрос

```
SELECT    *
FROM      events
WHERE     (
             users ? :uid
               OR
             companies ? :cid
          )
      AND
        NOT hideforusers ? :uid
ORDER BY created DESC
LIMIT 20
```


Blocks operations
ops/second
1K
0.5K
0
temp read
temp written

Правильная структура

```
CREATE TABLE events2 (
   userid      UUID,
   companyid   UUID,
   id          UUID,
   CONSTRAINT  pk_eventsv2
      PRIMARY KEY (userid, companyid, id)
)
```

Запросы без таймаутов

```
set statement_timeout to 100000;COMMIT;
SELECT    *
FROM      table
WHERE     ...
ORDER by  ...
```

Проверка на существование

```
public bool IsEntityExists(string val) {

    string sql = @"SELECT count(*)
                   FROM entities
                   WHERE col = :val";

    int count = _dbMapper.Execute<int>(sql, val);

    return count > 0;
}
```

Проверка на существование

```
public bool IsEntityExists(string val) {

    string sql = @"SELECT EXISTS(SELECT 1
                        FROM entities
                        WHERE col = :val)";

    return _dbMapper.Execute<bool>(sql, val);
}
```

Херовый запрос

```
SELECT count (*) AS totalcount
FROM table
WHERE companyid = '0'
    AND STATUS <> '1' AND STATUS <> '2'
    AND (type <> '3'
        OR ((category = '5' AND kind <> '6')
                OR (category = '7' AND kind = '7')))
          AND (type <> '8' OR category = '9')
          AND (type <> '9' OR category = '5')
          AND date (date) >= '2017-06-24'
          AND date (date) <= '2017-07-24';
```

BRIN - Block Range Index

```
select count(*)
 from userlogin
 where
      date >= '2017-12-01' and
      date < '2018-01-01'
```

```
1   Aggregate (cost=362241.00..362241.01 rows=1 width=8)
2   -> Seq Scan on userlogin (cost=0.00..362114.72 rows=50512 width=0)
(1771 MB)
3   Filter: date >= '2017-12-01' AND date < '2018-01-01'
```

BRIN - Block Range Index

```
create index idx_userlogin_brin on userlogin
using brin (date)
with (pages_per_range=8);
```

BRIN - 72 KB

BTREE - 216 Mb

BRIN - Block Range Index

```
1   Aggregate (cost=268.02..268.03 rows=1 width=8)(actual time=7.009..7.009)
2   -> Bitmap Heap Scan on userlogin (cost=264.00..268.02 rows=1 width=0)
3        Recheck Cond: (date >= '2017-12-01' AND date < '2018-01-01')
4        Rows Removed by Index Recheck: 1311
5        Heap Blocks: lossy=26
6   ->      Bitmap Index Scan on idx_userlogin_brin (cost=0.00..264.00)
7                Index Cond: (date >= '2017-12-01' AND date < '2018-01-01')
8   Planning time: 0.212 ms
9   Execution time: 7.049 ms
```

Хранить меньше, если можно

```
{
  "id": 123,
  "name": "first",
  "status": "executed",
  "country": null,
  "city": ""
}
```

```
{
  "name": "five"
}
```

Change type

Колонка timestamp для хранения даты, т.е. время всегда было по нулям

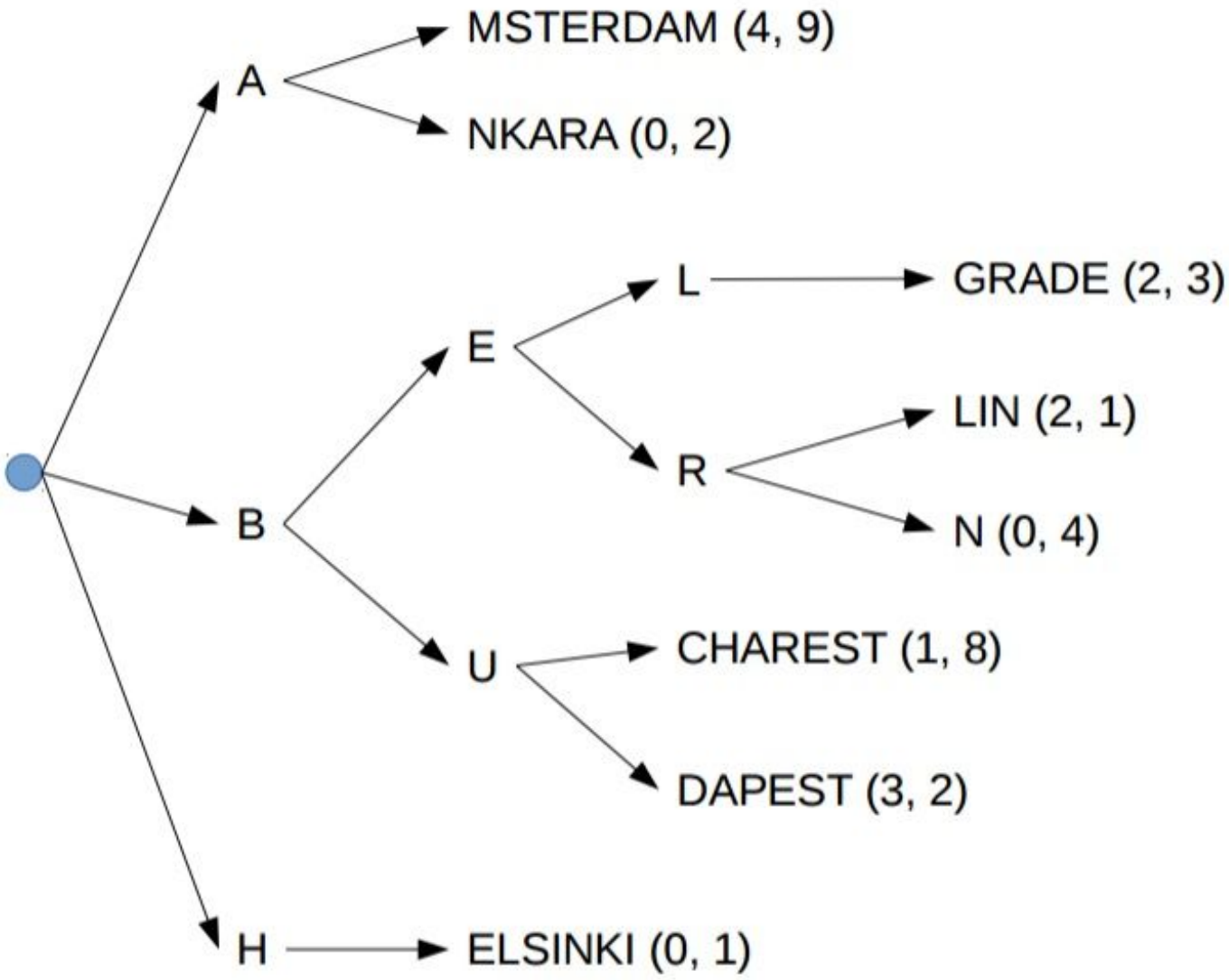
```
alter TABLE table1
   alter column date type date;
```

Экономия на убирании пустых байт - более 1.3 Гб в таблице + ежедневные бекапы

SP-GiST

```
select *
from t1
where str like ‘:str%’;
```

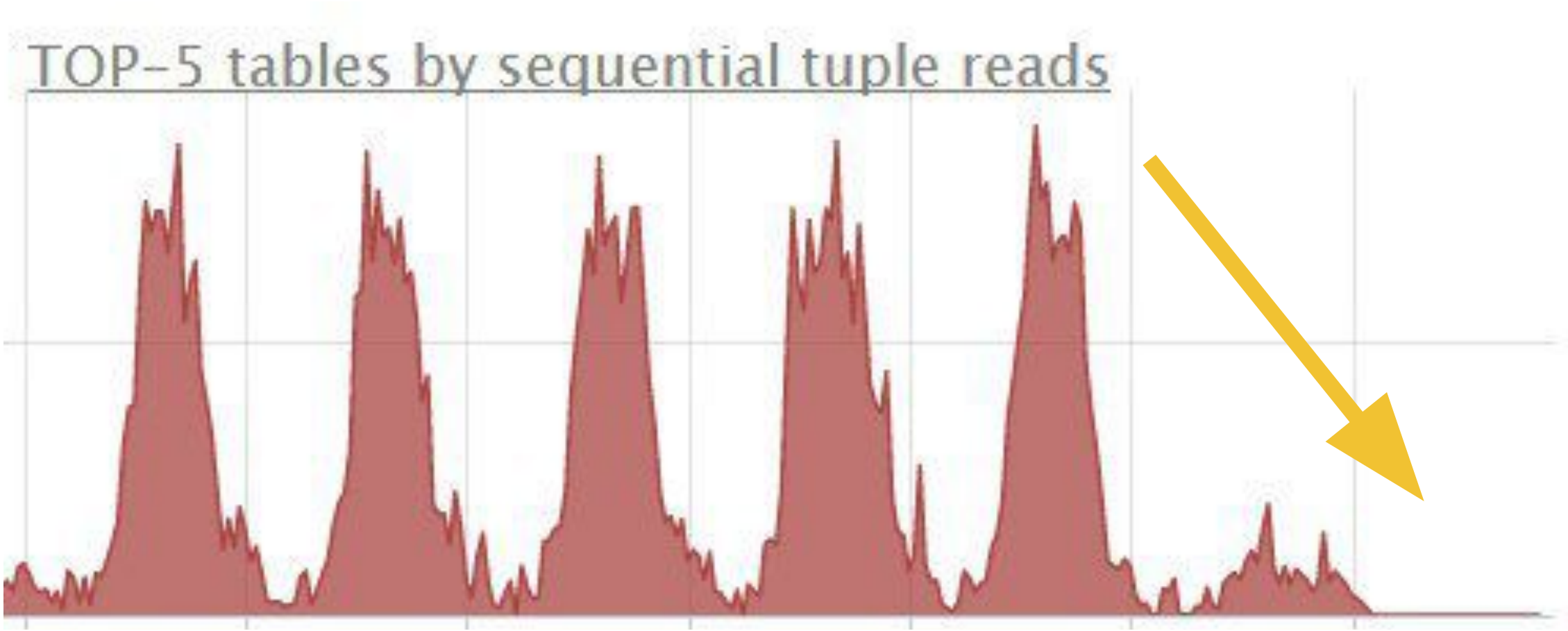
```
create index idx_t1_str_sp
   on t1
      using spgist (str);
```



GIN

```
select *
from t1
where
   name_tsv @@
   (plainto_tsquery(:name)::text || ':*')::tsquery;

create index idx_t1_name
   on t1
      using gin(name_tsv);
```

Польза GIN



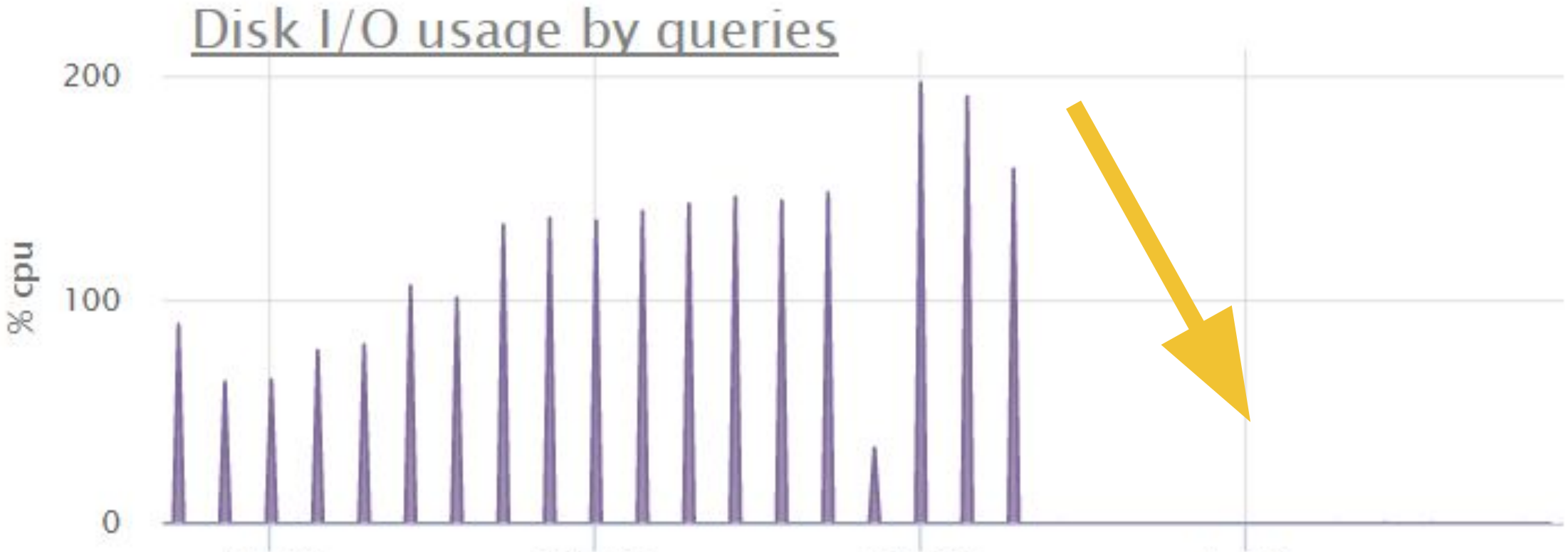
Partial index

```
SELECT true
FROM table1
WHERE companyid=:companyId
    AND f1='1'
    AND date_part('year', moddate)
      = date_part('year', CURRENT_DATE)
    AND status='2'
LIMIT 1;
```

Partial index

```
create index
on table1 (companyid, date_part('year', moddate))
where
   f1='1' AND status='2'
```

Примеры оптимизаций - Partial index



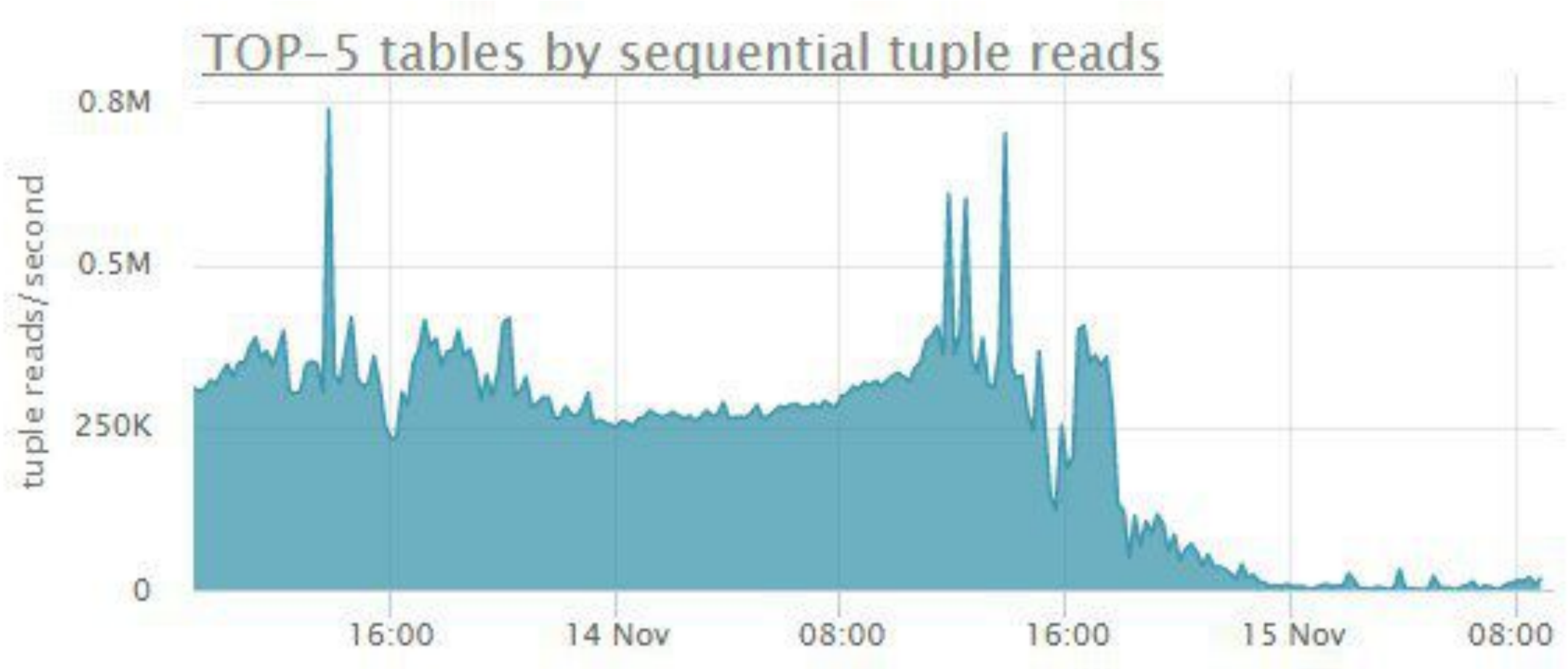
Сложная сортировка

```
select * from tasks ORDER BY
    (performerid IS NULL AND status = '1') desc,
    (status='2'),
    (status='3' OR status='4'),
    pauseenddate,
    expectedenddate desc,
    enddate desc,
    GREATEST(case when enddate is null then lastupdate else enddate
end, created) desc,
    id desc
) limit 20;
```

Индекс для сортировки

```
create index on tasks (
    (performerid IS NULL AND status = '1') desc,
    (status='2'),
    (status='3' OR status='4'),
    pauseenddate,
    expectedenddate desc,
    enddate desc,
    GREATEST(case when enddate is null then lastupdate else enddate
end, created) desc,
    id desc
);
```

Примеры оптимизаций - индекс для сортировки



Union sort

```
select * from (
    select id, d from table1 where c = 1
    union
    select id, d, from table2 where c = 2
) t
where date(d) >= :d
order by d desc
limit 20
```

Вредные советы

- Хинты планера - https://habrahabr.ru/post/169751/

```
/*+ HashJoin(t1 t2) */
EXPLAIN ANALYZE
SELECT * FROM test1 t1 JOIN test2 t2
   ON t1.id_2 = t2.id
   WHERE t2.value BETWEEN 0.5 AND 0.51
   ORDER BY t1.value1 LIMIT 100;
```

Вредные советы

- Костыльный выбор индекса - https://habrahabr.ru/post/343686/

```
CREATE FUNCTION "selindex" ("name" TEXT)
    RETURNS BOOLEAN
AS $$ BEGIN RETURN TRUE; END;
$$
LANGUAGE plpgsql
IMMUTABLE
```

```
CREATE INDEX "table_index1"
ON "table1"("a","b","c");
    WHERE "selindex"('table_index1')
```

Postgres auto_explain

```
load 'auto_explain';
SET auto_explain.log_min_duration = 10;
SET auto_explain.log_analyze = true;
```

```
select count(*)
from somet
where number < 1000
```

```
2018-05-05 23:16:47 +05 [25072]:[5-1]LOG:  statement:
        select count(*) from somet where number < 1000
2018-05-05 23:16:48 +05 [25102]:[1-1]LOG:  duration: 1079.020 ms  plan:
        Query Text:
        select count(*) from somet where number < 1000
        Partial Aggregate  (cost=138185.47..138185.48 rows=1 width=8)
          ->  Parallel Seq Scan on somet  (cost=0.00..138174.77 rows=4278 width=0)
                Filter: (number < 1000)
2018-05-05 23:16:48 +05 [25101]:[1-1]LOG:  duration: 1081.468 ms  plan:
        Query Text:
        select count(*) from somet where number < 1000
        Partial Aggregate  (cost=138185.47..138185.48 rows=1 width=8)
          ->  Parallel Seq Scan on somet  (cost=0.00..138174.77 rows=4278 width=0)
                Filter: (number < 1000)
2018-05-05 23:16:48 +05 [25072]:[6-1]LOG:  duration: 1095.373 ms  plan:
        Query Text:
        select count(*) from somet where number < 1000
        Finalize Aggregate  (cost=139185.68..139185.69 rows=1 width=8)
          ->  Gather  (cost=139185.47..139185.68 rows=2 width=8)
                Workers Planned: 2
                ->  Partial Aggregate  (cost=138185.47..138185.48 rows=1 width=8)
                      ->  Parallel Seq Scan on somet  (cost=0.00..138174.77 rows=42
                            Filter: (number < 1000)
```

https://goo.gl/tSCu7J

Просмотр логов - pgbadger

Slowest individual queries

Rank	Duration	Query
1	1s70ms	SELECT count (*) FROM somet WHERE number < 1000; [Date: 2018-05-05 23:24:07] Explain plan Finalize Aggregate (cost=139185.68..139185.69 rows=1 width=8) -> Gather (cost=139185.47..139185.68 rows=2 width=8) Workers Planned: 2 -> Partial Aggregate (cost=138185.47..138185.48 rows=1 width=8) -> Parallel Seq Scan on somet (cost=0.00..138174.77 rows=4278 width=0) Filter: (number < 1000)

http://dalibo.github.io/pgbadger/ https://habr.com/post/354948/

Рекомендуемая литература

- https://postgrespro.ru/docs/postgrespro/10/using-explain
- https://habrahabr.ru/company/postgrespro/blog/326096/
  - 9 статей про индексы
    - не забывай читать комменты
- Книга «Высоконагруженные приложения. Программирование, масштабирование, поддержка»

alexander.pavlov@live.ru

https://goo.gl/MEynUq