PostgresPro Joker<?>

Как PostgreSQL может сделать больно, когда не ожидаешь?

Михаил Жилин

PostgresPro

Михаил Жилин

- Физтех
- 15 лет в нагрузочном тестировании и Java разработке
- Занимаюсь производительностью PostgreSQL и не только
- Контрибьютор в Open Source проекты

PostgresPro

О чём сегодня будет разговор?

PostgresPro

О чём сегодня будет разговор?

- PostgreSQL – это база

PostgresPro

О чём сегодня будет разговор?

- PostgreSQL – это база
- Тормоза

PostgresPro

О чём сегодня будет разговор?

- PostgreSQL – это база
- Тормоза
- Страдания

PostgresPro

О чём сегодня будет разговор?

- PostgreSQL – это база
- Тормоза
- Страдания
- Кошмары

PostgresPro

О чём сегодня будет разговор?

- Только реальные примеры

PostgresPro

О чём сегодня будет разговор?

- Только реальные примеры
- Все имена изменены

PostgresPro

О чём сегодня будет разговор?

- Только реальные примеры
- Все имена изменены
- Все совпадения случайны

PostgresPro

О чём сегодня будет разговор?

- Только реальные примеры
- Все имена изменены
- Все совпадения случайны
- Хотите верьте, хотите нет

PostgresPro

О чём сегодня не будет разговора?

- Как находить кривые запросы?

PostgresPro

О чём сегодня не будет разговора?

- Как находить кривые запросы?
- Как тюнить запросы?

PostgresPro

О чём сегодня не будет разговора?

- Как находить кривые запросы?
- Как тюнить запросы?
- То что уже было…

PostgresPro

О чём сегодня не будет разговора?

- Как находить кривые запросы?
- Как тюнить запросы?
- То что уже было…
  - **HL 2022: Аномальные случаи высокой нагрузки в PostgreSQL**

    https://highload.ru/moscow/2022/abstracts/9659
  - **PgConf.СПб 2023: Способы решения проблем PostgreSQL путём отладки и профилирования**

    https://pgconf.ru/talk/1589558

PostgresPro

PostgreSQL

PostgresPro

PostgreSQL

- Это база: таблички, индексы, вьюхи

PostgresPro

PostgreSQL

- Это база: таблички, индексы, вьюхи
- 1970-1980...

PostgresPro

PostgreSQL

- Егор Рогов, PgConf.СПб 2024

  «Сиквел и приквел: занимательная археология»

  https://pgconf.ru/talk/1866399

PostgresPro

PostgreSQL

- Это база: таблички, индексы, вьюхи
- 1970-1980...
- SEQUEL (SQL) vs QUEL (Ingres)

PostgresPro

PostgreSQL

QUEL(Ingres):

```
RANGE OF X IS CITY
  RETRIEVE INTO W(X.STATE)
  WHERE SET(X.CNAME BY X.STATE
  WHERE X.POPU<4M)
  = SET(X.CNAME BY X.STATE)
```

https://bit.ly/3U3so6o

PostgresPro

PostgreSQL

- Postgres = пост-Ingres
  - POSTQUEL (вариант SQL-3) → **libpq**

PostgresPro

PostgreSQL

- Postgres = пост-Ingres
  - POSTQUEL (вариант SQL-3) → **libpq**
- PostgreSQL = Postgres + SQL

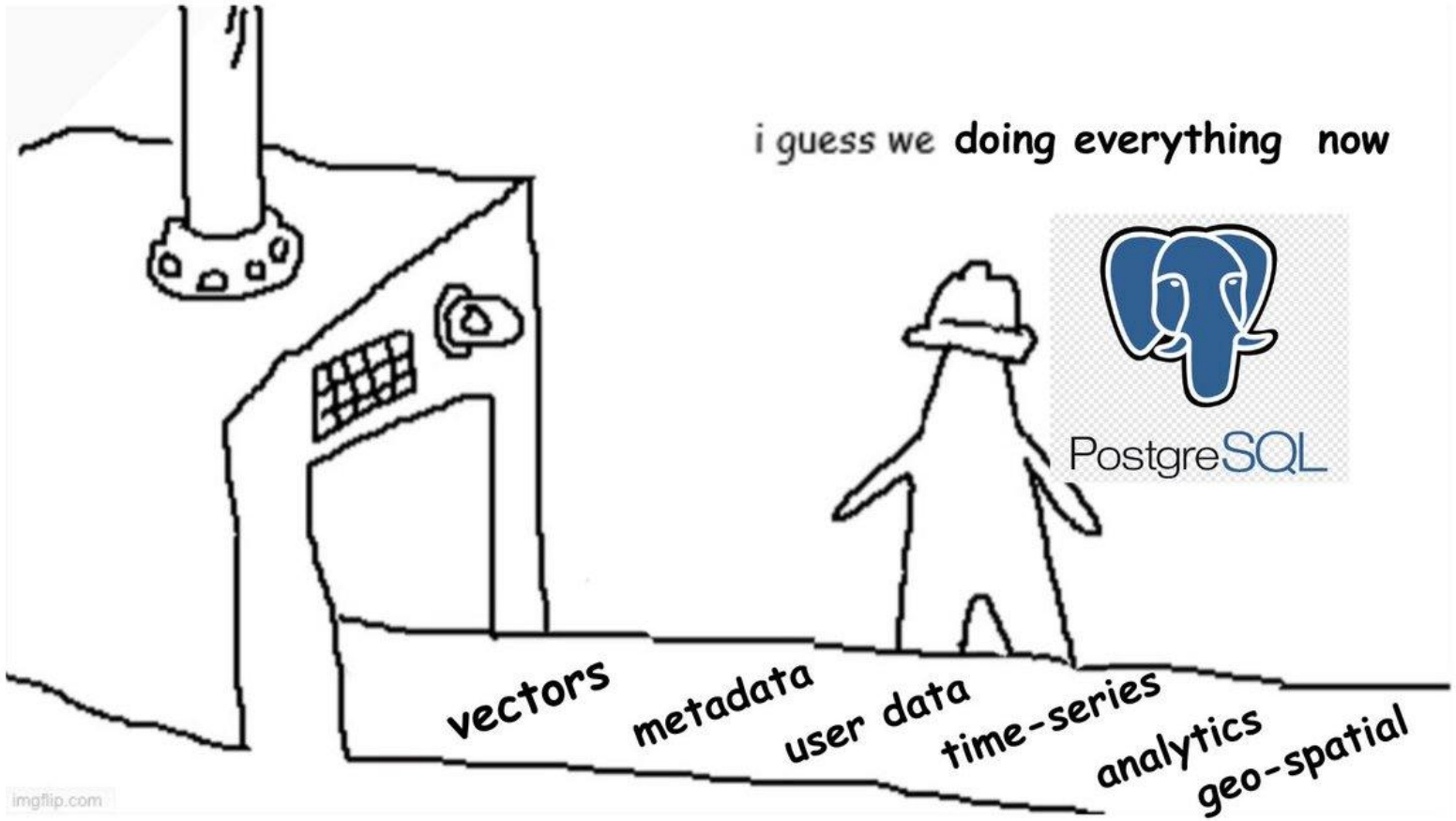
PostgresPro

PostgreSQL

- Postgres = пост-Ingres
  - POSTQUEL (вариант SQL-3) → **libpq**
- PostgreSQL = Postgres + SQL
- Сейчас
  - SQL 2023
  - PostGIS, TimescaleDB, Citus / Cloudberry, Apache AGE, pgvector
  - Partitioning (секционирование), jsonb

PostgresPro

PostgreSQL



PostgresPro

PostgreSQL

- Performance
  - 3 миллиона запросов в секунду на 2s Intel 8352У (2022 год)
  - 150+ Терабайтов данных

PostgresPro

PostgreSQL

- PG17 (26 сентября 2024)
  - Инкрементальные бекапы
  - JSON_TABLE (Никита Глухов)
  - Коррелированные подзапросы в Join-ы
- Feature Matrix

  https://bit.ly/3Ymzcyj

PostgresPro

PostgreSQL

- 30 лет разработки
- Мощный функционал
- Быстродействие

PostgresPro

PostgreSQL

- 30 лет разработки
- Мощный функционал
- Быстродействие
- **Ну что может пойти не так?**

PostgresPro

#1: Глобальные счётчики

PostgresPro

#1: Глобальные счётчики

- **Условный Wildberries**

  8.8 миллионов заказов в день = 100 в секунду

- **Общее количество товаров на складах**

  ~100 строк в таблице (id, cnt)

PostgresPro

#1: Глобальные счётчики

```
# create table global_counters (id int primary key, cnt
bigint);
```

PostgresPro

#1: Глобальные счётчики

```
# create table global_counters (id int primary key, cnt
bigint);

# insert into global_counters (id, cnt)
select id, 1000000 from generate_series (1, 100) as id;
```

PostgresPro

#1: Глобальные счётчики

```
# create table global_counters (id int primary key, cnt
bigint);

# insert into global_counters (id, cnt)
select id, 1000000 from generate_series (1, 100) as id;
```

SQL: “update global_counters set cnt = cnt + ? where id = ?”

PostgresPro

#1: Глобальные счётчики

```
\set v_id random(1,100)
\set v_delta random(-10,10)

update global_counters set cnt = cnt + :v_delta where id
= :v_id;
```

PostgresPro

#1: Глобальные счётчики

```
\set v_id random(1,100)
\set v_delta random(-10,10)

update global_counters set cnt = cnt + :v_delta where id
= :v_id;
```

```
# pgbench -n -T 3600 -c 10 -j 10 -P 1 -f test.sql
```

PostgresPro

#1: Глобальные счётчики

```
progress: 117.0 s, 5077.0 tps, lat 1.969 ms stddev 0.493, 0 failed
progress: 118.0 s, 4962.0 tps, lat 2.016 ms stddev 0.641, 0 failed
progress: 119.0 s, 5152.0 tps, lat 1.939 ms stddev 0.498, 0 failed
progress: 120.0 s, 5018.0 tps, lat 1.994 ms stddev 0.497, 0 failed
progress: 121.0 s, 5039.0 tps, lat 1.983 ms stddev 0.50, 0 failed
progress: 122.0 s, 5086.0 tps, lat 1.966 ms stddev 0.483, 0 failed
progress: 123.0 s, 5042.0 tps, lat 1.982 ms stddev 0.498, 0 failed
```

PostgresPro

#1: Глобальные счётчики

PostgresPro

#1: Глобальные счётчики

```
progress: 1663.0 s, 958.0 tps, lat 10.411 ms stddev 4.278, 0 failed
progress: 1664.0 s, 1043.0 tps, lat 9.619 ms stddev 2.373, 0 failed
progress: 1665.0 s, 1049.0 tps, lat 9.490 ms stddev 2.119, 0 failed
progress: 1666.0 s, 1067.0 tps, lat 9.393 ms stddev 2.109, 0 failed
progress: 1667.0 s, 1054.0 tps, lat 9.489 ms stddev 2.019, 0 failed
progress: 1668.0 s, 1077.0 tps, lat 9.286 ms stddev 1.827, 0 failed
progress: 1669.0 s, 1056.0 tps, lat 9.471 ms stddev 2.135, 0 failed
```

Было же 5000! Как так? Почему?

PostgresPro

#1: Глобальные счётчики

```
# create extension pg_query_state;
CREATE EXTENSION
```

#1: Глобальные счётчики

- Екатерина Соколова, PgConf.Spb 2024

  «Дело о пропавшей производительности в PostgreSQL: руководство по поимке и обезвреживанию проблемных запросов»

  https://pgconf.ru/talk/1907134

PostgresPro

#1: Глобальные счётчики

```
# select plan from pg_query_state(693654, buffers=>true);
```

PostgresPro

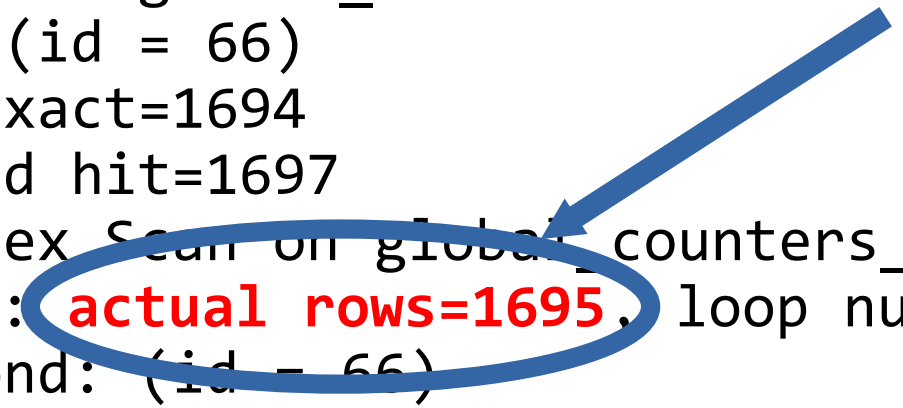
#1: Глобальные счётчики

```
# select plan from pg_query_state(693654, buffers=>true);

                                  plan
-----------------------------------------------------------------------
 Update on global_counters
   ->  Bitmap Heap Scan on global_counters
         Recheck Cond: (id = 66)
         Heap Blocks: exact=1694
         Buffers: shared hit=1697
         ->  Bitmap Index Scan on global_counters_pkey
          (Current loop: actual rows=1695, loop number=1)
               Index Cond: (id = 66)
               Buffers: shared hit=3
```

PostgresPro

#1: Глобальные счётчики

```
# select plan from pg_query_state(693654, buffers=>true);

                                  plan
-------------------------------------------------------------------------
 Update on global_counters
   ->  Bitmap Heap Scan on global_counters
         Recheck Cond: (id = 66)
         Heap Blocks: exact=1694
         Buffers: shared hit=1697
         ->  Bitmap Index Scan on global_counters_pkey
          (Current loop: actual rows=1695, loop number=1)
               Index Cond: (id = 66)
               Buffers: shared hit=3
```



PostgresPro

#1: Глобальные счётчики

```
# select plan from pg_query_state(6936

                                plan
-------------------------------------------------------------
 Update on global_counters
   ->  Bitmap Heap Scan on global_coun
         Recheck Cond: (id = 66)
         Heap Blocks: exact=1694
         Buffers: shared hit=1697
         ->  Bitmap Index Scan on global_counters_pkey
          (Current loop: actual rows=1695, loop number=1)
               Index Cond: (id = 66)
               Buffers: shared hit=3
```

PostgresPro

#1: Глобальные счётчики

- **PostgreSQL Multi-Version Concurrency Control (MVCC)**

PostgresPro

#1: Глобальные счётчики

- **PostgreSQL Multi-Version Concurrency Control (MVCC)**
- Каждый update - новая версия строки

PostgresPro

#1: Глобальные счётчики

- **PostgreSQL Multi-Version Concurrency Control (MVCC)**
- Каждый update - новая версия строки
- Индекс сохраняет ссылки на **все** блоки данных с копиями строки

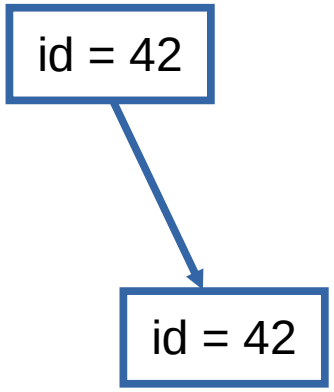
PostgresPro

#1: Глобальные счётчики

id = 42

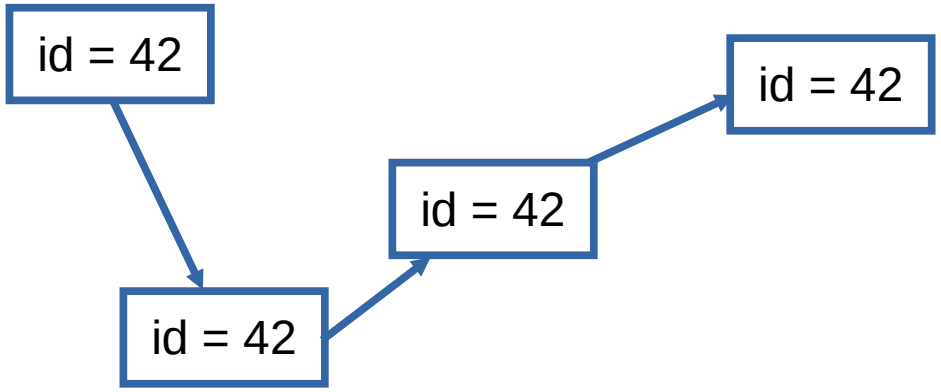
PostgresPro

#1: Глобальные счётчики



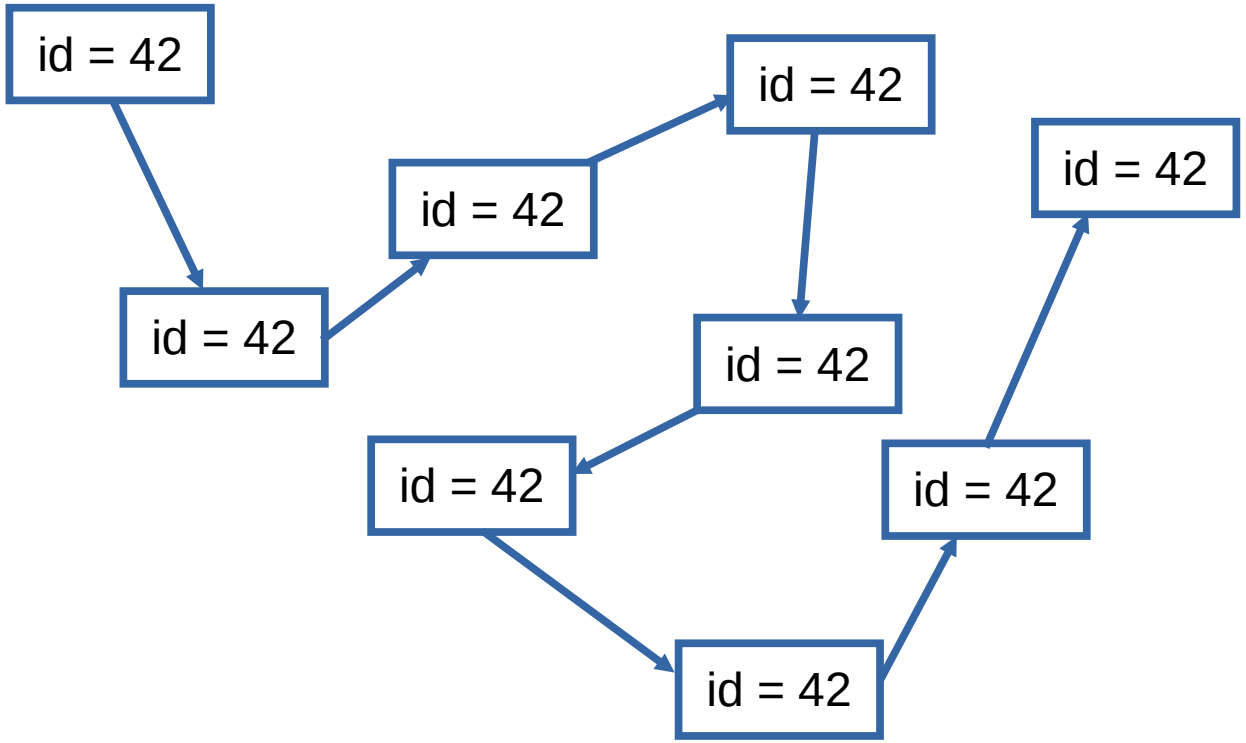
PostgresPro

#1: Глобальные счётчики



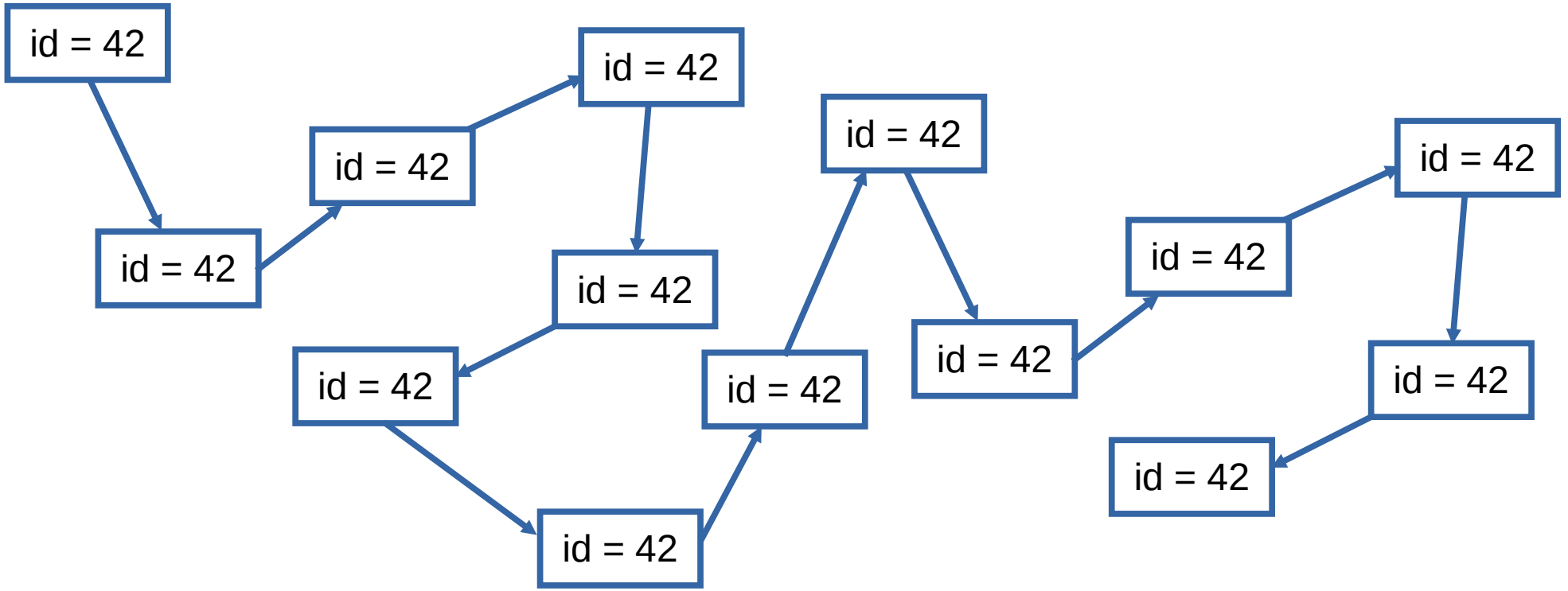
PostgresPro

#1: Глобальные счётчики



PostgresPro

#1: Глобальные счётчики

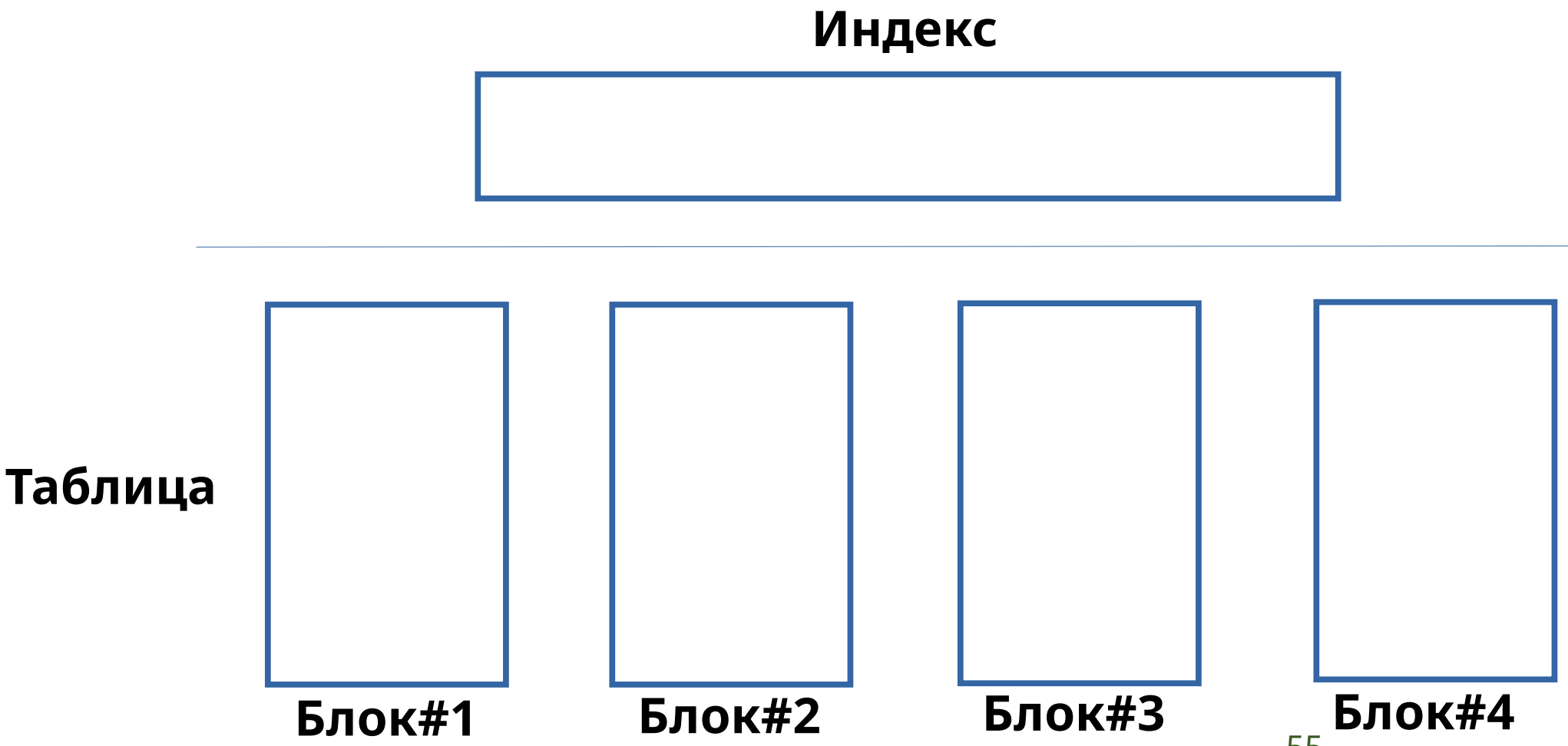


PostgresPro

#1: Глобальные счётчики

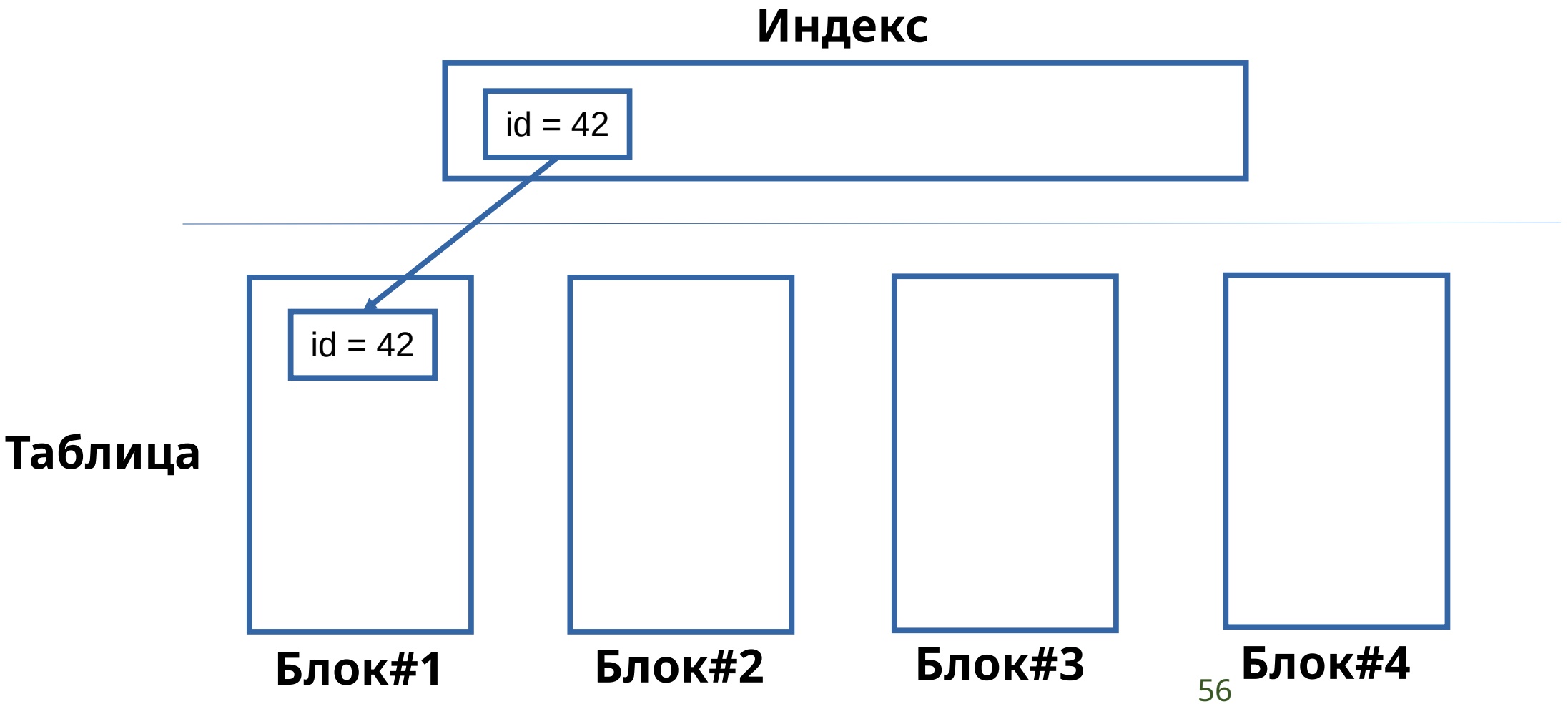
PostgresPro

Глобальные счётчики



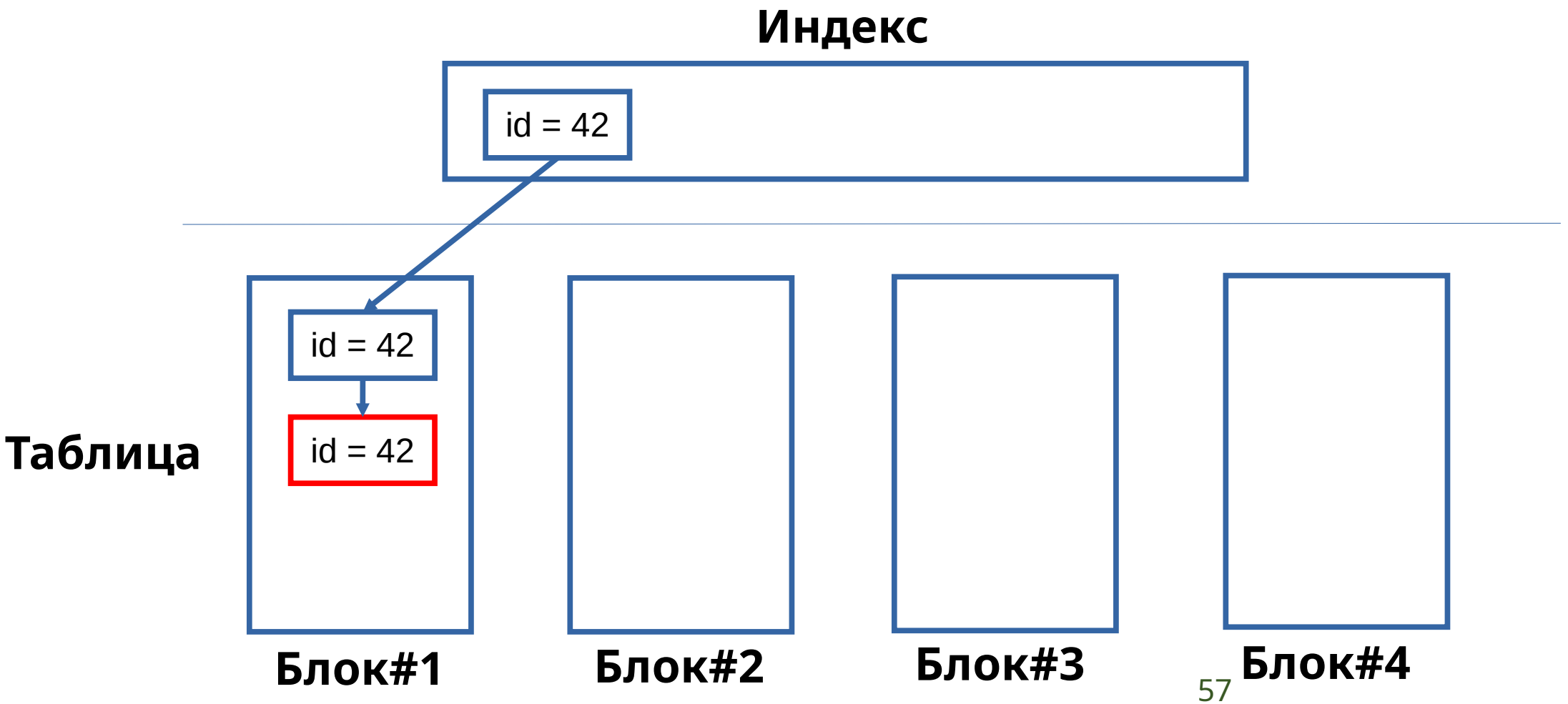
PostgresPro

Глобальные счётчики



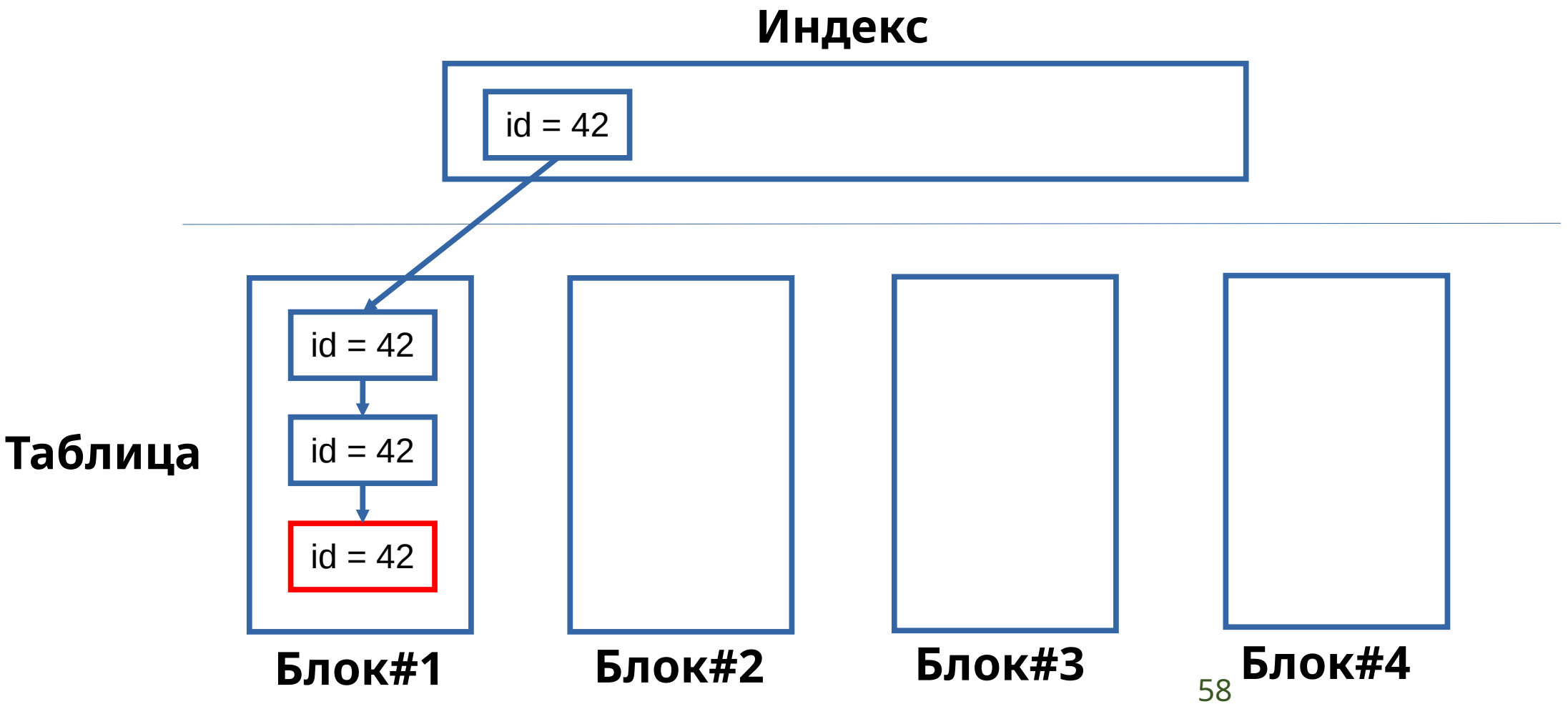
PostgresPro

Глобальные счётчики



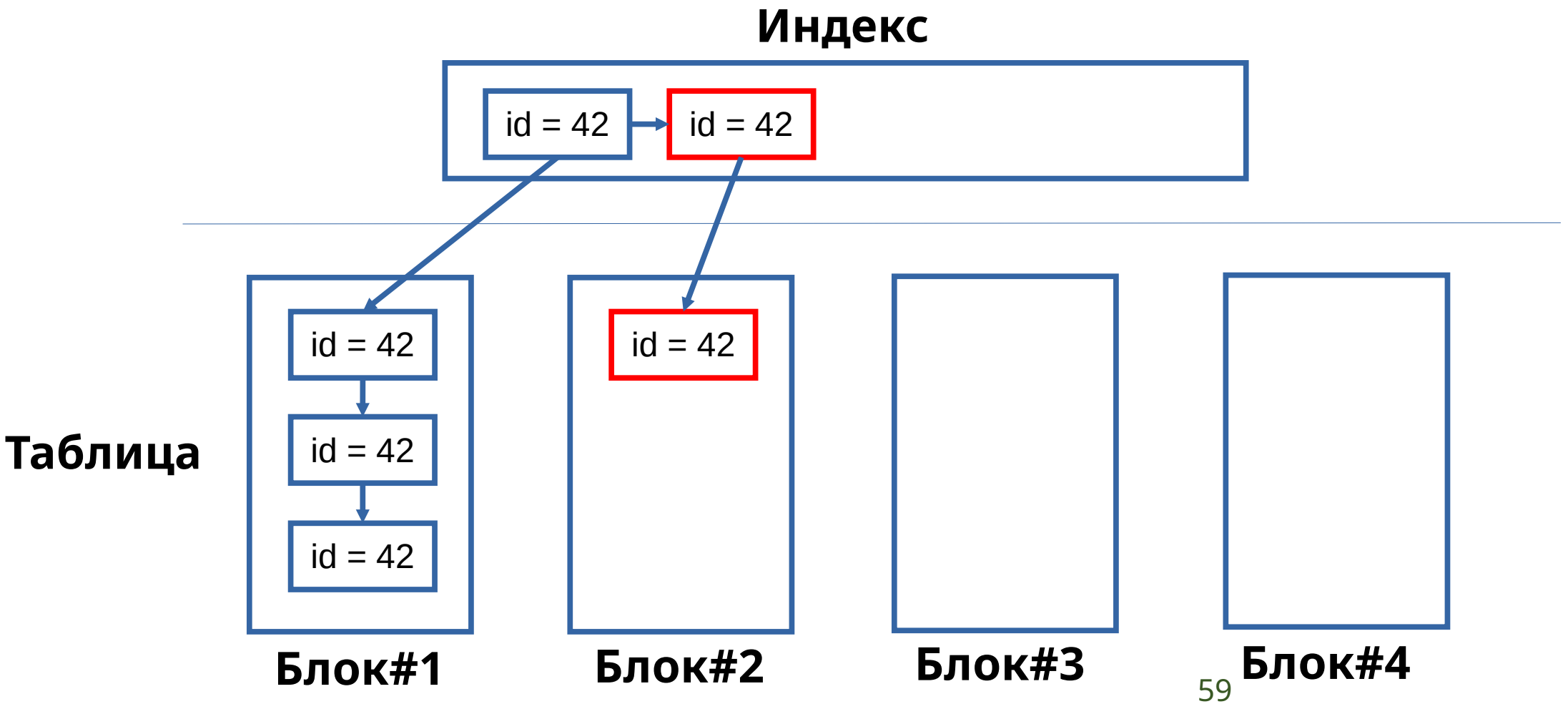
PostgresPro

Глобальные счётчики



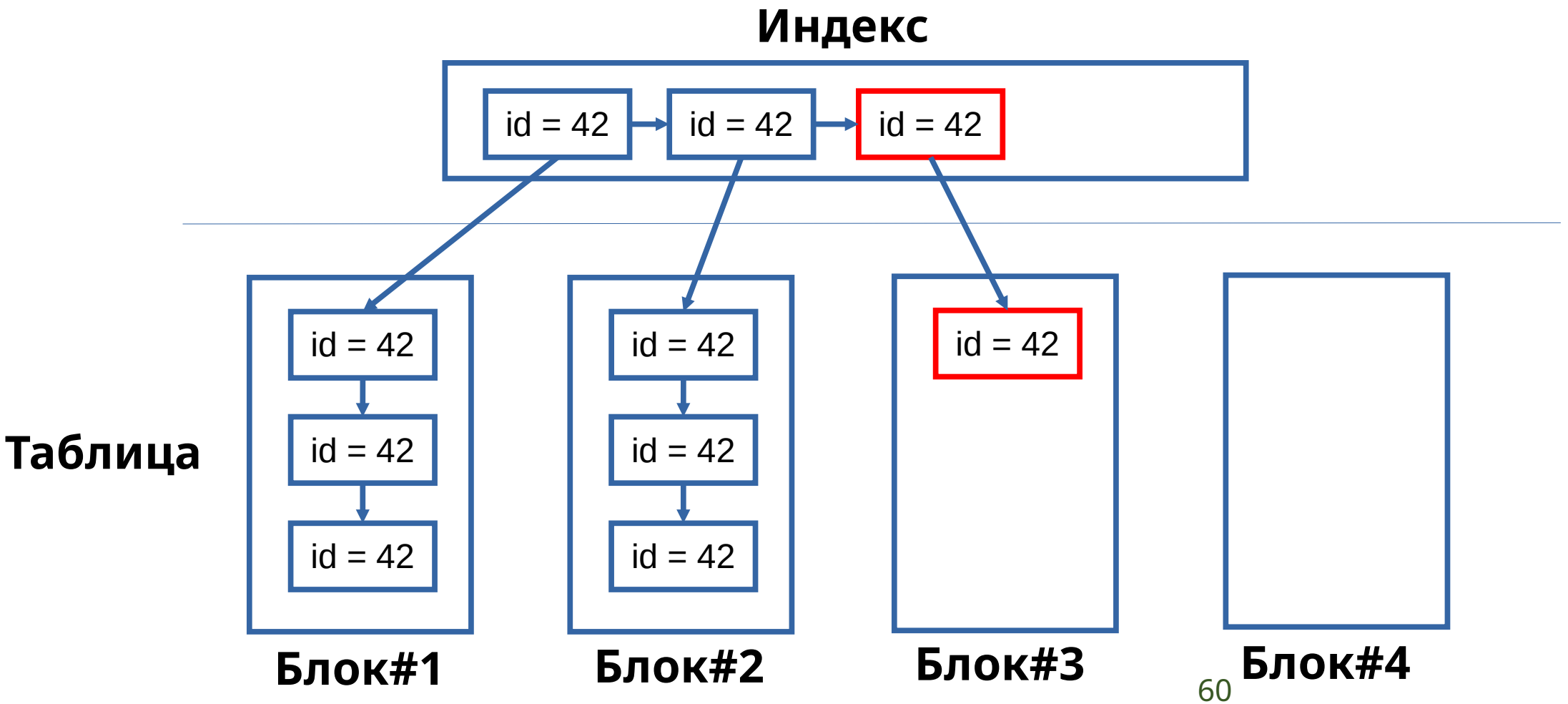
PostgresPro

Глобальные счётчики



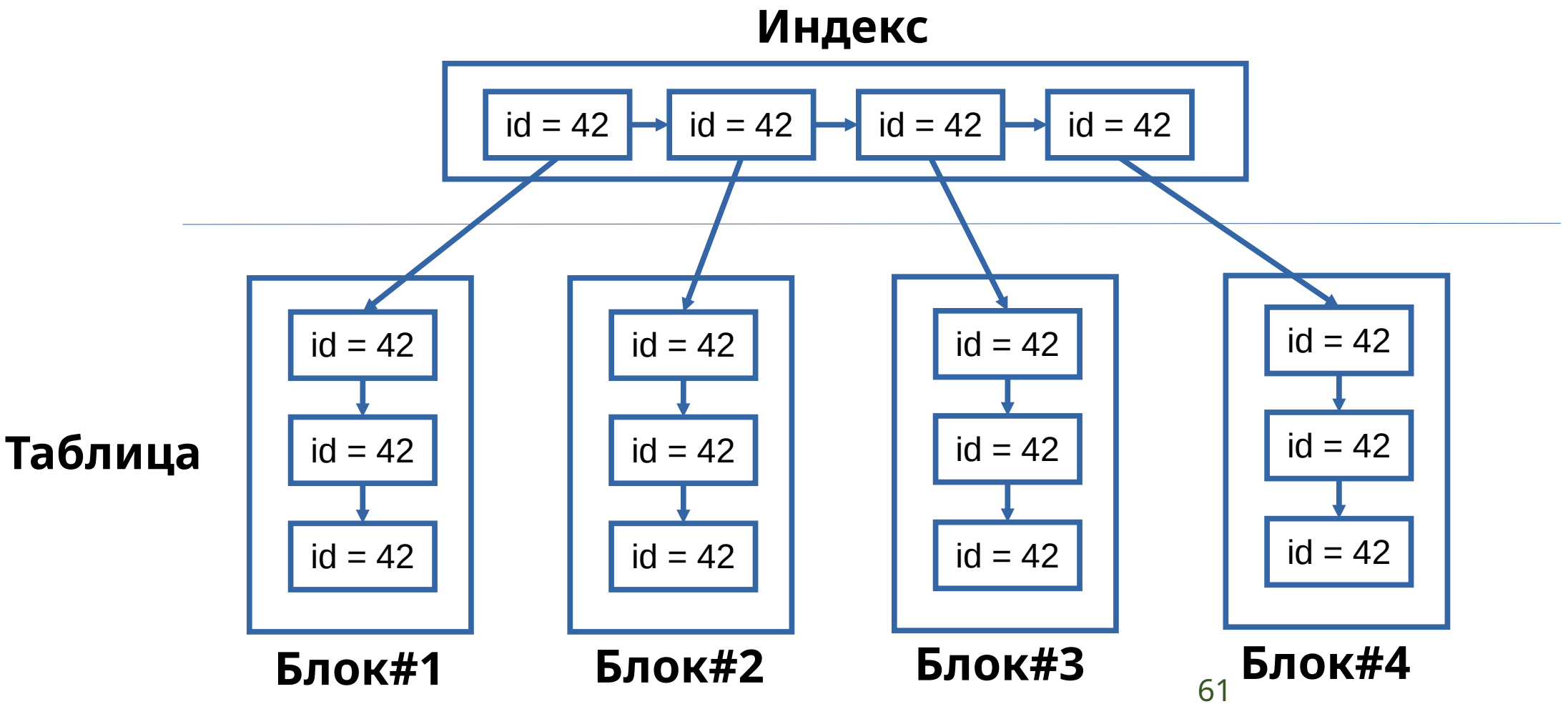
PostgresPro

Глобальные счётчики



PostgresPro

Глобальные счётчики



PostgresPro

#1: Глобальные счётчики

- Владимир Ситников, JPoint 2023

  «B-Tree индексы в базах данных на примере Spring Boot-приложений, PostgreSQL и JPA»

  https://bit.ly/4f6ICn6

PostgresPro

#1: Глобальные счётчики

```
# select plan from pg_query_state(693654, buffers=>true);

                                plan
-----------------------------------------------------------------------
 Update on global_counters
   ->  Bitmap Heap Scan on global_counters
         Recheck Cond: (id = 66)
         Heap Blocks: exact=1694
         Buffers: shared hit=1697
         ->  Bitmap Index Scan on global_counters_pkey
          (Current loop: actual rows=1695, loop number=1)
               Index Cond: (id = 66)
               Buffers: shared hit=3
```

PostgresPro

#1: Глобальные счётчики

- Старые версии удаляются процессом autovacuum

PostgresPro

#1: Глобальные счётчики

- Старые версии удаляются процессом autovacuum
- Причины ”неудаления” строк

PostgresPro

#1: Глобальные счётчики

- Старые версии удаляются процессом autovacuum
- Причины ”неудаления” строк
  - Давно висящие транзакции

PostgresPro

#1: Глобальные счётчики

- Старые версии удаляются процессом autovacuum
- Причины ”неудаления” строк
  - Давно висящие транзакции
  - Редкий запуск autovacuum

PostgresPro

#1: Глобальные счётчики

- Старые версии удаляются процессом autovacuum
- Причины ”неудаления” строк
  - Давно висящие транзакции
  - Редкий запуск autovacuum
  - Репликации

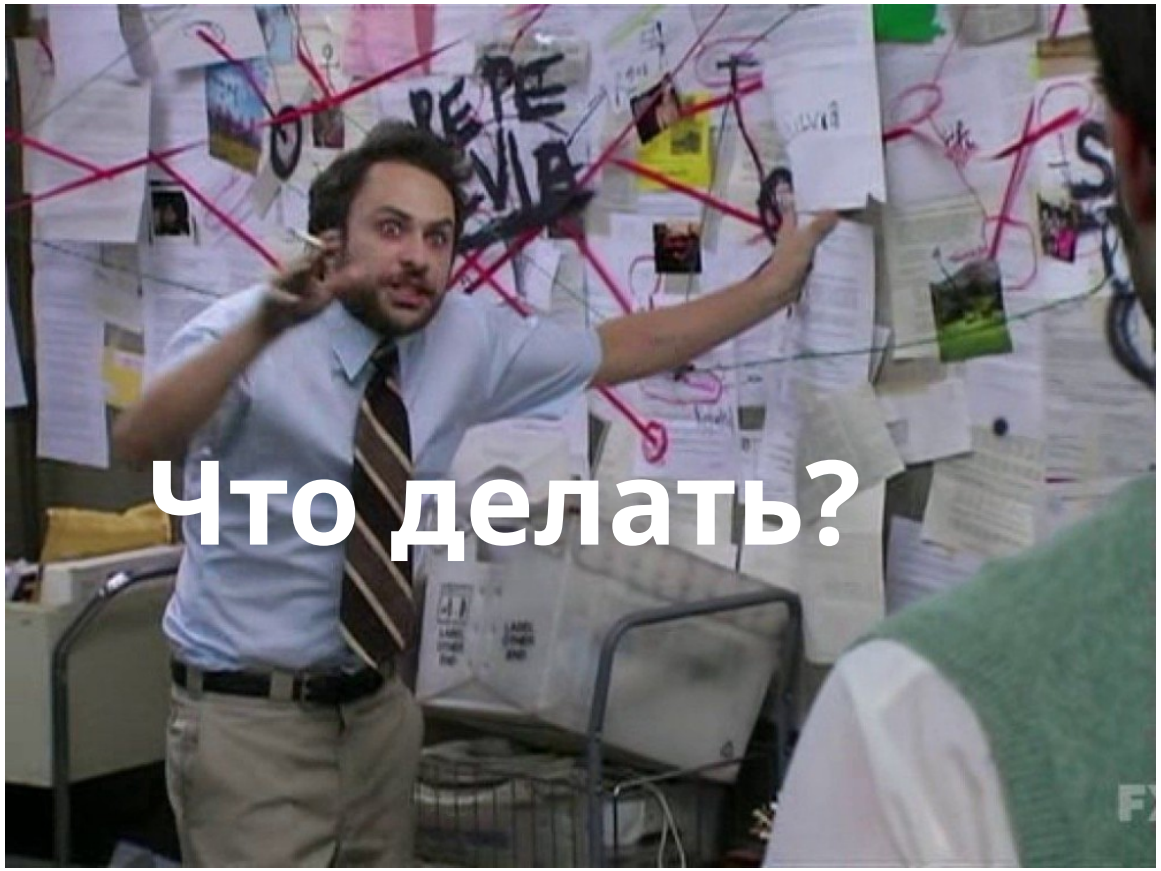
PostgresPro

#1: Глобальные счётчики

- Старые версии удаляются процессом autovacuum
- Причины ”неудаления” строк
  - Давно висящие транзакции
  - Редкий запуск autovacuum
  - Репликации
  - И 100500 других причин

PostgresPro

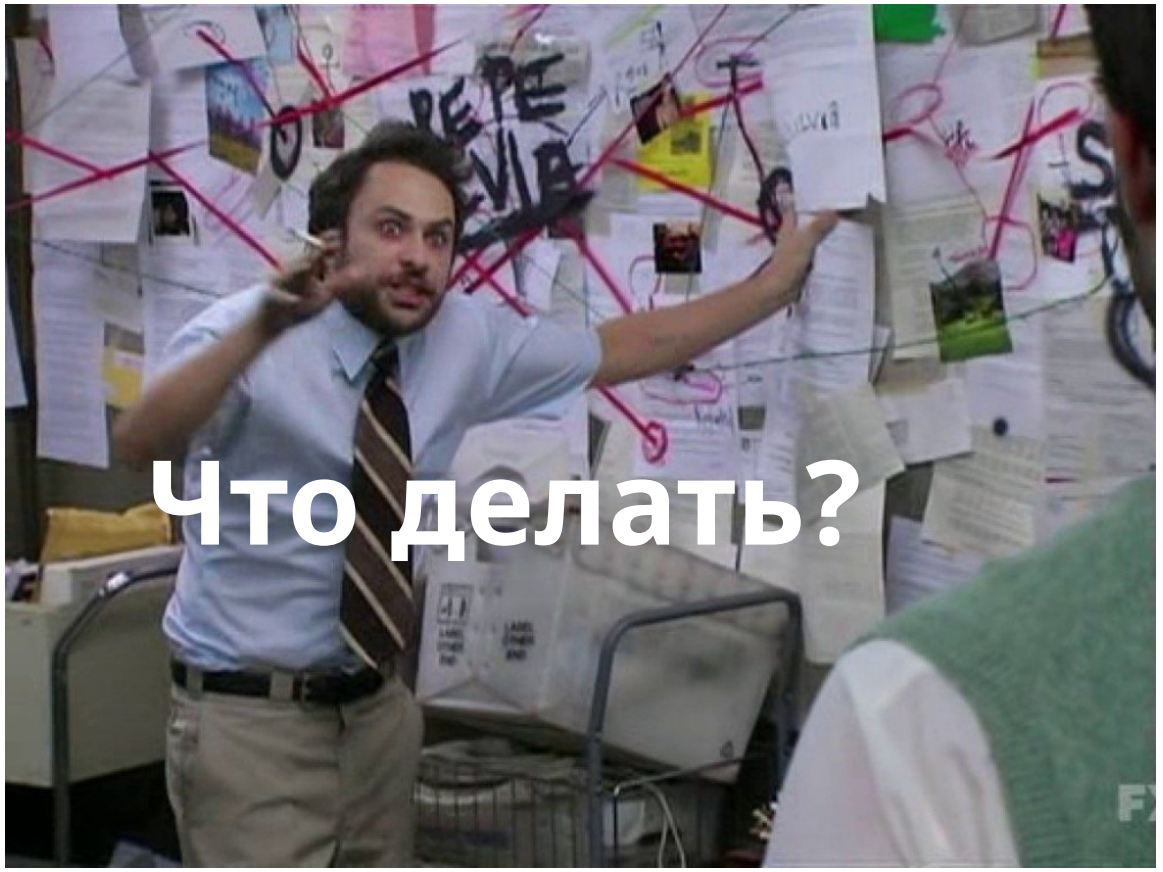
#1: Глобальные счётчики



PostgresPro

#1: Глобальные счётчики

- Отказаться от PostgreSQL

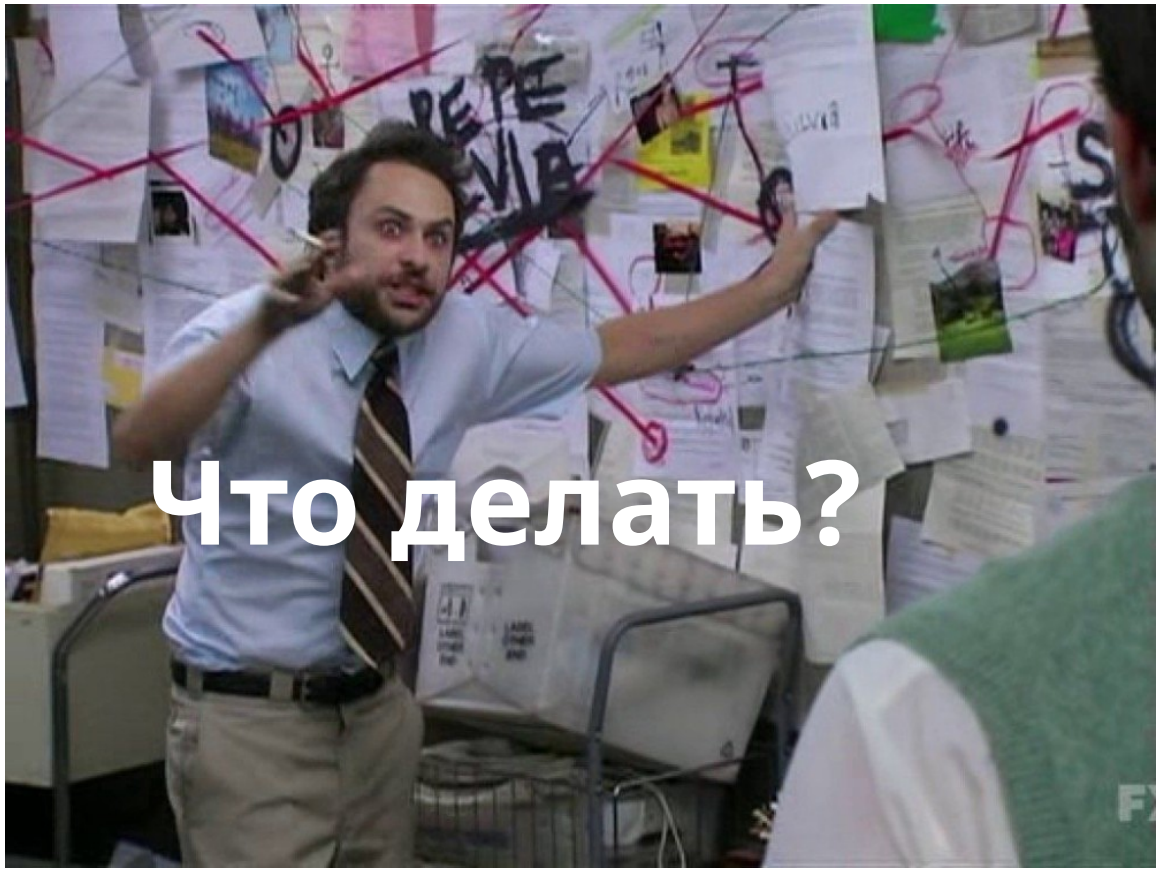


PostgresPro

#1: Глобальные счётчики

- ~~Отказаться от PostgreSQL~~
- Заиспользовать ZHEAP!

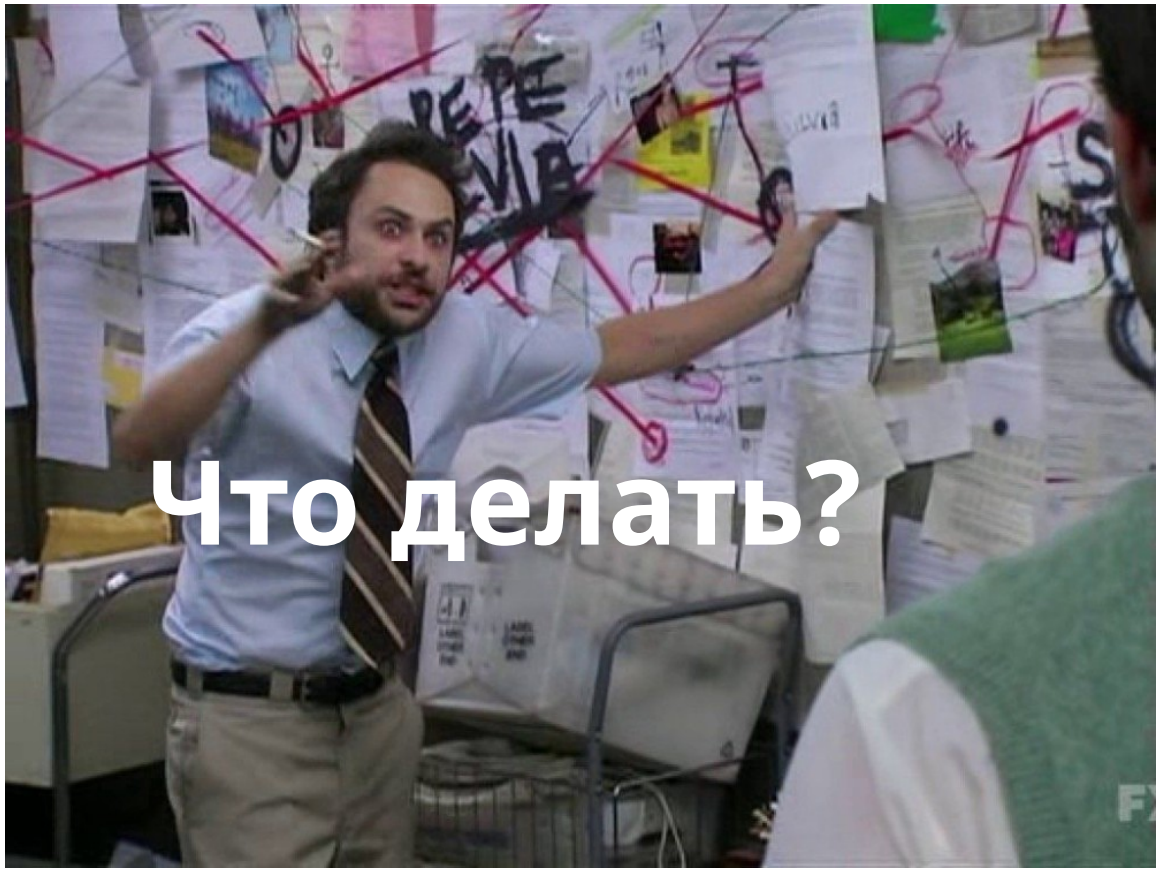
  “zheap is a way to keep table bloat **under control**”



PostgresPro

#1: Глобальные счётчики

- ~~Отказаться от PostgreSQL~~
- ~~Заиспользовать ZHEAP!~~
  ~~“zheap is a way to keep table bloat under control”~~
  https://github.com/EnterpriseDB/zheap



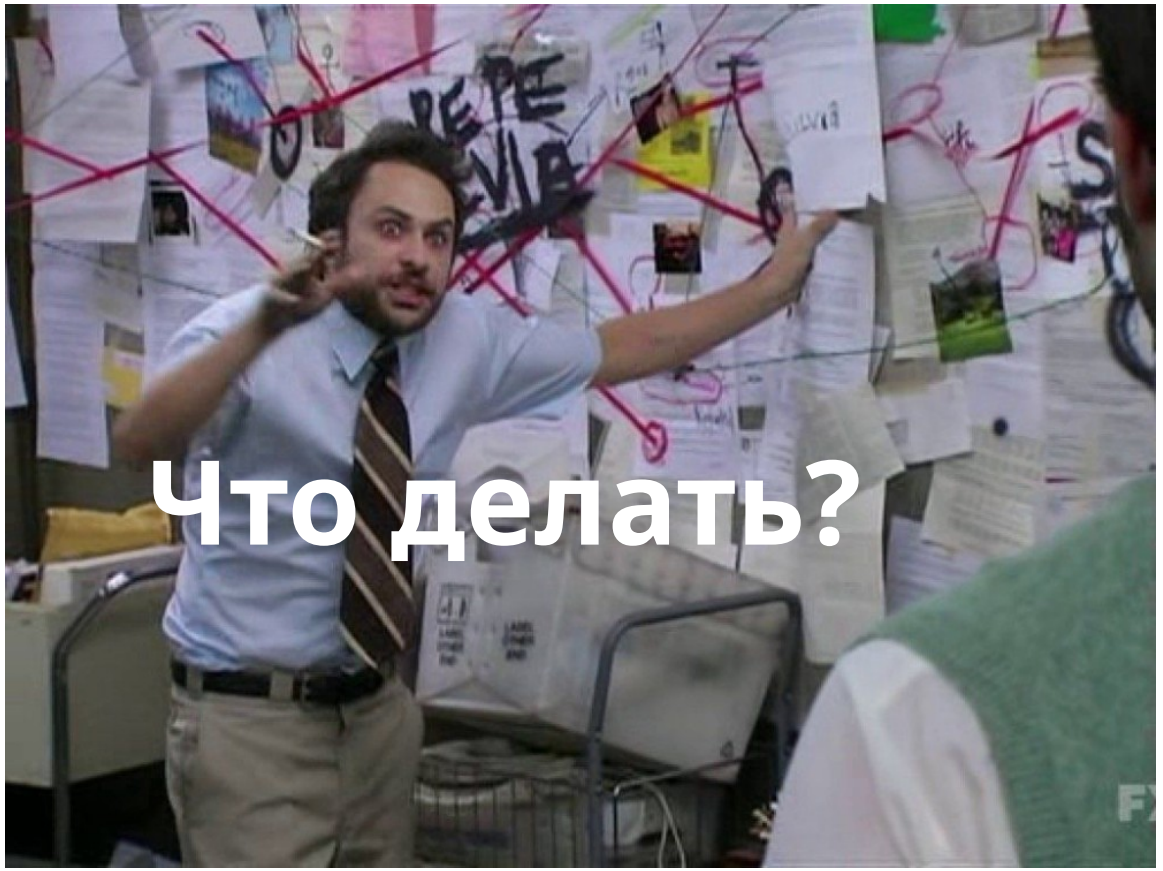
PostgresPro

#1: Глобальные счётчики

- ~~Отказаться от PostgreSQL~~
- ~~Заиспользовать ZHEAP!~~

  ~~“zheap is a way to keep table bloat under control”~~

  https://github.com/EnterpriseDB/zheap
- Подсказать, где искать нужную строку



PostgresPro

#1: Глобальные счётчики

```
# create table global_counters
  (id int primary key,
  cnt bigint,
  last_updated timestamp);
```

PostgresPro

#1: Глобальные счётчики

```
# create table global_counters
  (id int primary key,
  cnt bigint,
  last_updated timestamp);

# create index g1 on global_counters(id, last_updated);
```

PostgresPro

#1: Глобальные счётчики

```
# insert into global_counters (id, cnt, last_updated)
select id, 1000000, now() from generate_series (1, 100) as id;
```

PostgresPro

#1: Глобальные счётчики

```
# insert into global_counters (id, cnt, last_updated)
select id, 1000000, now() from generate_series (1, 100) as id;

SQL: update global_counters
        set cnt = cnt + ?, last_updated = now()
      where ctid = (
            select ctid
              from global_counters
             where id = ?
             order by last_updated desc
             limit 1);
```

PostgresPro

#1: Глобальные счётчики

```
# select plan from pg_query_state(694712, buffers=>true);
                                plan
-----------------------------------------------------------------
 Update on global_counters
   InitPlan 1 (returns $0)
     ->  Limit
           Buffers: shared hit=4
           ->  Index Scan Backward using g1 on global_counters
               (Current loop: actual rows=1, loop number=1)
                 Index Cond: (id < 15)
                 Buffers: shared hit=4
   ->  Tid Scan on global_counters
       (Current loop: actual rows=1, loop number=1)
         TID Cond: (ctid = $0)
```

PostgresPro

#1: Глобальные счётчики

- Выводы
  - PostgreSQL хранит версии строк
  - Это может ухудшать производительность запросов
  - Можно «подсказать» PostgreSQL как лучше искать данные

PostgresPro

#2: Dude, Where’s my index?

PostgresPro

#2: Dude, Where’s my index?

- Клиент: PostgreSQL не использует созданный индекс

PostgresPro

#2: Dude, Where’s my index?

- Клиент: PostgreSQL не использует созданный индекс
- Тормозит Production сервер

PostgresPro

#2: Dude, Where’s my index?

- Клиент: PostgreSQL не использует созданный индекс
- Тормозит Production сервер
- Пересоздавали индекс — не помогло

PostgresPro

#2: Dude, Where’s my index?

- Клиент: PostgreSQL не использует созданный индекс
- Тормозит Production сервер
- Пересоздавали индекс — не помогло
- Через 2 часа: ура, заработало

PostgresPro

#2: Dude, Where’s my index?

- Клиент: PostgreSQL не использует созданный индекс
- Тормозит Production сервер
- Пересоздавали индекс — не помогло
- Через 2 часа: ура, заработало
- **“Убили какую-то транзакцию в базе”**

PostgresPro

#2: Dude, Where’s my index?

- Клиент: PostgreSQL не использует созданный индекс
- Тормозит Production сервер
- Пересоздавали индекс — не помогло
- Через 2 часа: ура, заработало
- **“Убили какую-то транзакцию в базе”**

PostgresPro

#2: Dude, Where’s my index?

- Создали таблицу

PostgresPro

#2: Dude, Where’s my index?

- Создали таблицу
- Заполнили данными

PostgresPro

#2: Dude, Where’s my index?

- Создали таблицу
- Заполнили данными
- Добавили индексы

PostgresPro

#2: Dude, Where’s my index?

- Создали таблицу
- Заполнили данными
- Добавили индексы
- Запустили приложение

PostgresPro

#2: Dude, Where’s my index?

- Создали таблицу
- Заполнили данными
- Добавили индексы
- Запустили приложение
- На тесте всё работает, на продуктиве нет...

PostgresPro

#2: Dude, Where’s my index?

- Создали таблицу
- Заполнили данными
- Добавили индексы
- Запустили приложение
- На тесте всё работает, на продуктиве нет...
- **“Убили какую-то транзакцию в базе и всё взлетело”**

PostgresPro

#2: Dude, Where's my index?

```
postgres=# begin;
BEGIN
postgres=*# select txid_current();
 txid_current
--------------
   10008974883
(1 row)
```

Подключение #1

PostgresPro

#2: Dude, Where's my index?

```
postgres=# create table tt(id bigint, val text);
CREATE TABLE
```

Подключение #2

PostgresPro

#2: Dude, Where's my index?

```
postgres=# insert into tt(id,val)
           select i, md5(i::text)
             from generate_series(1,10000) as i;

INSERT 0 10000
```

Подключение #2

PostgresPro

#2: Dude, Where's my index?

```
postgres=# update tt set val = 'asdf' where id = 42;
UPDATE 1
postgres=# delete from tt where id = 42;
DELETE 1
```

Подключение #2

PostgresPro

#2: Dude, Where's my index?

```
postgres=# create index on tt(id);
CREATE INDEX
```

Подключение #2

PostgresPro

#2: Dude, Where's my index?

```
postgres=# explain select * from tt where id = 100;
```

Подключение #2

PostgresPro

#2: Dude, Where's my index?

```
postgres=# explain select * from tt where id = 100;
                          QUERY PLAN
---------------------------------------------------------
 Seq Scan on tt  (cost=0.00..218.99 rows=50 width=40)
   Filter: (id = 100)
(2 rows)
```

Подключение #2

PostgresPro

#2: Dude, Where's my index?

Полное сканирование таблицы, никаких индексов

```
postgres=# explain select * from tt where id = 100;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on tt  (cost=0.00..218.99 rows=50 width=40)
   Filter: (id = 100)
(2 rows)
```

Подключение #2

PostgresPro

#2: Dude, Where’s my index?

```
postgres=# commit;
COMMIT
```

Подключение #1

PostgresPro

#2: Dude, Where's my index?

УРА! Мой индекс

```
postgres=# explain select * from tt where id = 100;
                          QUERY PLAN
---------------------------------------------------------------
 Index Scan using tt_id_idx on tt  (cost=0.29..2.50 rows=1)
   Index Cond: (id = 100)
(2 rows)
```

Подключение #2

PostgresPro

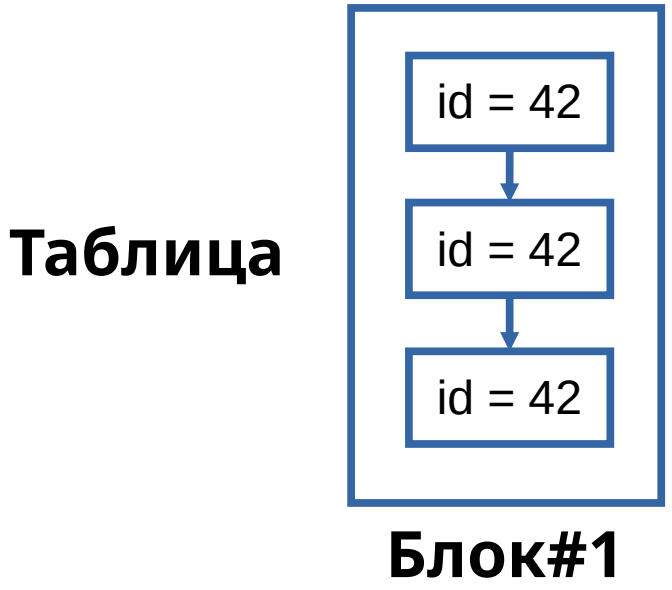
#2: Dude, Where's my index?

```
postgres=# update tt set val = 'asdf' where id = 42;
UPDATE 1
postgres=# delete from tt where id = 42;
DELETE 1
```

Подключение #2

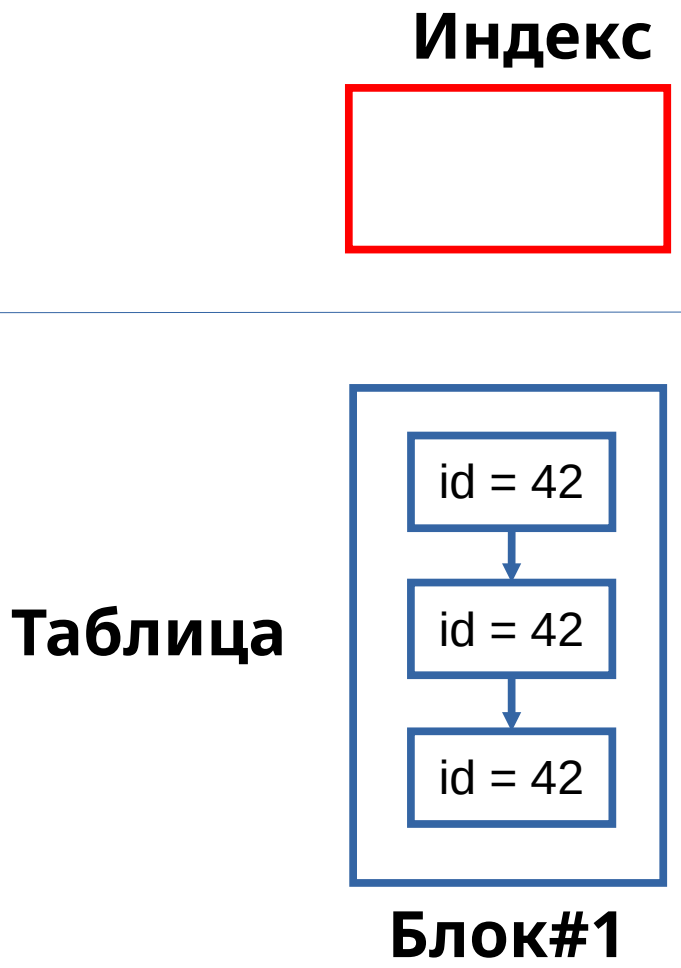
PostgresPro

#2: Dude, Where’s my index?



PostgresPro

#2: Dude, Where's my index?

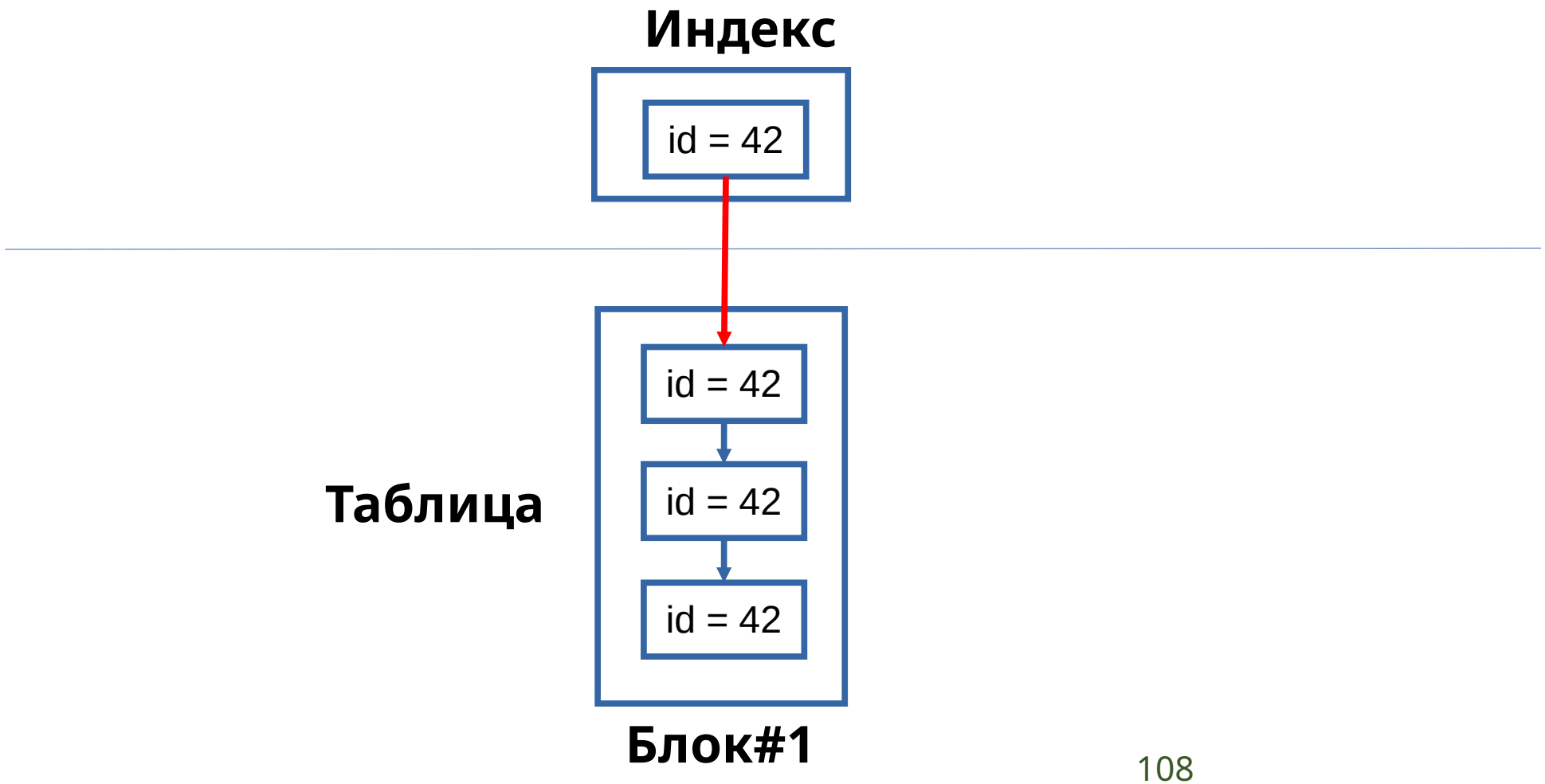


PostgresPro

#2: Dude, Where's my index?

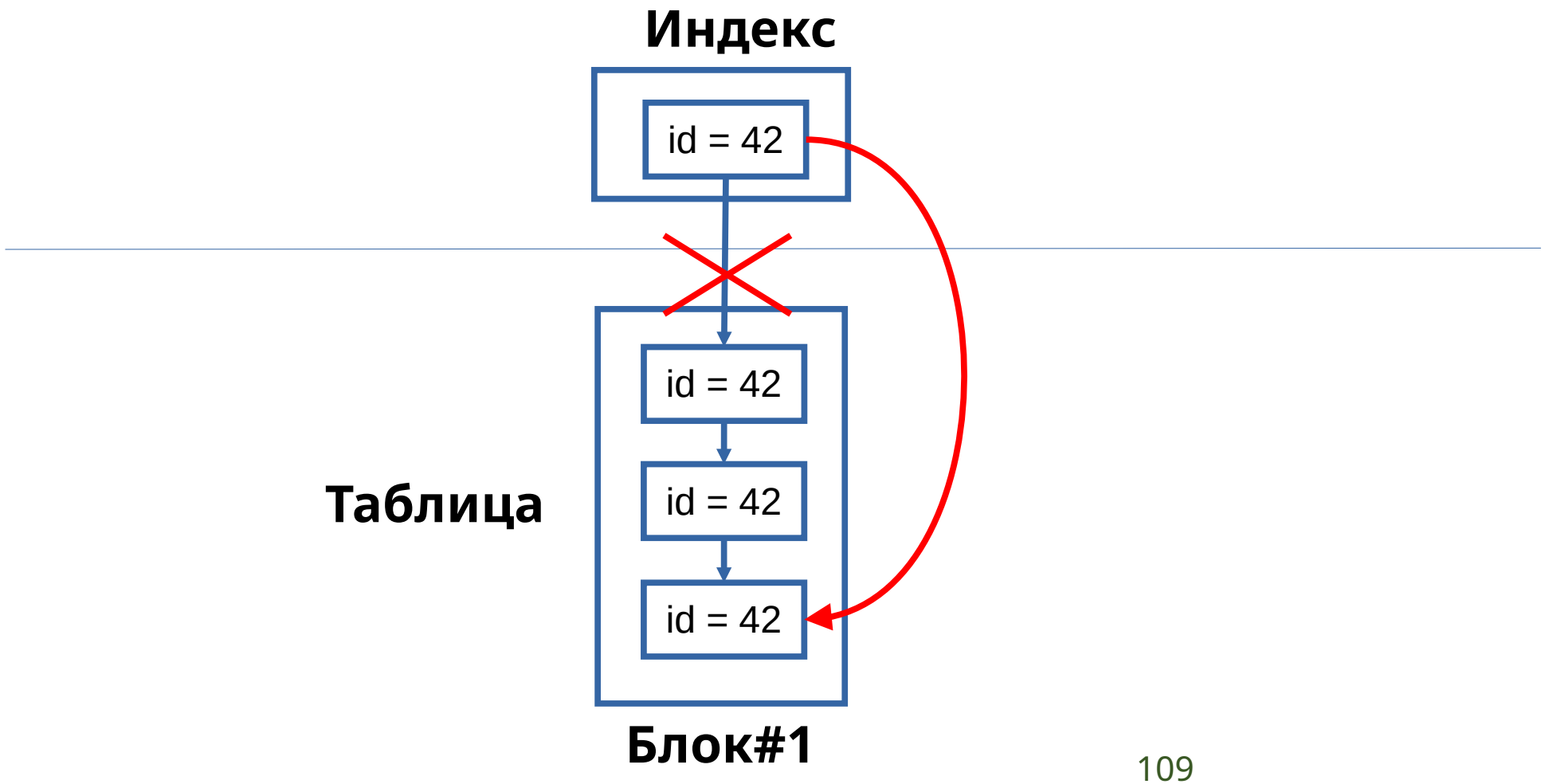
PostgresPro

#2: Dude, Where's my index?



PostgresPro

#2: Dude, Where's my index?



PostgresPro

#2: Dude, Where’s my index?

- PostgreSQL не знает содержимое таплов (вдруг там менялось значение ключа?)

PostgresPro

#2: Dude, Where's my index?

- PostgreSQL не знает содержимое таплов (вдруг там менялось значение ключа?)
- Выбираем pessimistic way (функция **heapam_index_build_range_scan**):

> However, if it was HOT-updated then we must only index the live tuple at the end of the HOT-chain. Since this breaks semantics for pre-existing snapshots, mark the index as unusable for them.

PostgresPro

#2: Dude, Where’s my index?

- Если есть цепочка изменений (aka HOT updates), то

PostgresPro

#2: Dude, Where’s my index?

- Если есть цепочка изменений (aka HOT updates), то
  - Индексируются последние версии строк

PostgresPro

#2: Dude, Where’s my index?

- Если есть цепочка изменений (aka HOT updates), то
  - Индексируются последние версии строк
  - Индекс не ссылается на устаревшие версии

PostgresPro

#2: Dude, Where’s my index?

- Если есть цепочка изменений (aka HOT updates), то
  - Индексируются последние версии строк
  - Индекс не ссылается на устаревшие версии
  - Нельзя использовать пока устаревшие версии кто-то использует

PostgresPro

#2: Dude, Where’s my index?

- Если есть цепочка изменений (aka HOT updates), то
  - Индексируются последние версии строк
  - Индекс не ссылается на устаревшие версии
  - Нельзя использовать пока устаревшие версии кто-то использует
  - Надо дождаться… После индекс будет использоваться

PostgresPro

#2: Dude, Where’s my index?

- А как понять что индекс “отложенный”?

PostgresPro

#2: Dude, Where’s my index?

- А как понять что индекс “отложенный”?
- `select indcheckxmin from pg_index where ...`

PostgresPro

#2: Dude, Where’s my index?

- А как понять что индекс “отложенный”?
- `select indcheckxmin from pg_index where ...`
- pg_index.indcheckxmin
  - “Если true, запросы не должны использовать этот индекс, пока поле xmin данной записи в pg_index не окажется ниже их горизонта событий TransactionXmin, так как таблица может содержать оборванные цепочки HOT с видимыми несовместимыми строками”

PostgresPro

#2: Dude, Where’s my index?

- Что может помочь?

PostgresPro

#2: Dude, Where’s my index?

- Что может помочь?
- vacuum перед созданием индекса
  - Может не сработать для самых свежих цепочек

PostgresPro

#2: Dude, Where’s my index?

- Что может помочь?
- vacuum перед созданием индекса
  - Может не сработать для самых свежих цепочек
- Подождать
  - CREATE INDEX CONCURRENTLY

PostgresPro

#2: Dude, Where’s my index?

- Что может помочь?
- vacuum перед созданием индекса
  - Может не сработать для самых свежих цепочек
- Подождать
  - CREATE INDEX CONCURRENTLY
- Завершить транзакции в сессиях, у которых xmin меньше xmin-а индекса (представление pg_stat_activity)
  - Нужны права на pg_terminate_backend

PostgresPro

#2: Dude, Where’s my index?

- Выводы
  - Индексы могут не сразу подхватиться
  - Такие индексы легко найти через pg_index.indcheckxmin
  - Совет всем разработчикам и тестировщикам: перед деплоем и тестом сделать «долговисящую» транзакцию — **Have Fun!**

PostgresPro

#3: (Auto)Vacuum

PostgresPro

#3: (Auto)Vacuum

- Илья и Федор Сазоновы
  JPoint 2024 «БД-укротитель»
  https://bit.ly/3zmCND6

PostgresPro

#3: (Auto)Vacuum

- Девушка из ~~Tinkoff~~ Т-Банка:

  Может ли блокировать запись автовакуумом в базу?

PostgresPro

#3: (Auto)Vacuum

- Девушка из ~~Tinkoff~~ Т-Банка:

  Может ли блокировать запись автовакуумом в базу?

- Я:

  Нет, конечно!

PostgresPro

#3: (Auto)Vacuum

- Девушка из ~~Tinkoff~~ Т-Банка:

  Может ли блокировать запись автовакуумом в базу?

- Я:

  Нет, конечно!

И был неправ!

PostgresPro

#3: (Auto)Vacuum

- Нагруженная система

PostgresPro

#3: (Auto)Vacuum

- Нагруженная система
- Таблица на гигабайты-терабайты

PostgresPro

#3: (Auto)Vacuum

- Нагруженная система
- Таблица на гигабайты-терабайты
- Терабайты RAM, сотни ядер

PostgresPro

#3: (Auto)Vacuum

- Нагруженная система
- Таблица на гигабайты-терабайты
- Терабайты RAM, сотни ядер
- Неожиданно замирает любой SQL на таблице

PostgresPro

#3: (Auto)Vacuum

- Нагруженная система
- Таблица на гигабайты-терабайты
- Терабайты RAM, сотни ядер
- Неожиданно замирает любой SQL на таблице
- И потом отмирает

PostgresPro

#3: (Auto)Vacuum

PostgresPro

#3: (Auto)Vacuum

- Нашли в логах autovacuum

PostgresPro

#3: (Auto)Vacuum

- Нашли в логах autovacuum
- vacuum_truncate
  - https://bit.ly/4eEzgis

PostgresPro

#3: (Auto)Vacuum

- Нашли в логах autovacuum
- vacuum_truncate
  - https://bit.ly/4eEzgis
- «Включает или отключает процедуру отсечения пустых страниц в конце таблицы в процессе очистки»

PostgresPro

#3: (Auto)Vacuum

- Нашли в логах autovacuum
- vacuum_truncate
  - https://bit.ly/4eEzgis
- «Включает или отключает процедуру отсечения пустых страниц в конце таблицы в процессе очистки»
- «Заметьте, что **для этого отсечения требуется блокировка таблицы на уровне ACCESS EXCLUSIVE**»

PostgresPro

#3: (Auto)Vacuum

- Чтобы обрезать это несчастные пустые блоки, надо пройти по всем блокам в кэше PostgreSQL

PostgresPro

#3: (Auto)Vacuum

- Чтобы обрезать это несчастные пустые блоки, надо пройти по всем блокам в кэше PostgreSQL
- Если кэш большой, сотни миллионов блоков, то это долго (минуты)!

PostgresPro

#3: (Auto)Vacuum

- Чтобы обрезать это несчастные пустые блоки, надо пройти по всем блокам в кэше PostgreSQL
- Если кэш большой, сотни миллионов блоков, то это долго (минуты)!
- Выставить опцию таблицы **vacuum_truncate=false**

PostgresPro

#3: (Auto)Vacuum

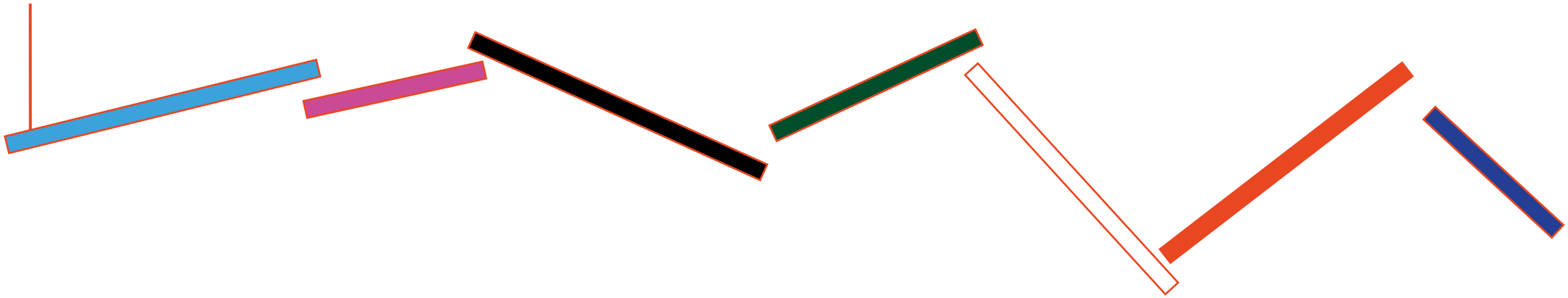
- Чтобы обрезать это несчастные пустые блоки, надо пройти по всем блокам в кэше PostgreSQL
- Если кэш большой, сотни миллионов блоков, то это долго (минуты)!
- Выставить опцию таблицы **vacuum_truncate=false**
- Идея обсудить в сообществе что так не стоит делать

PostgresPro

Итоги

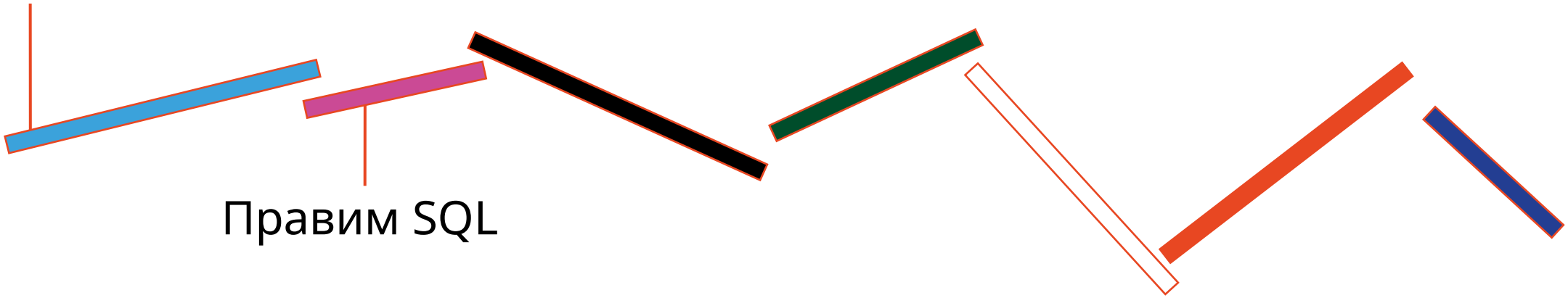
PostgresPro

Тормоза из-за **bloating**



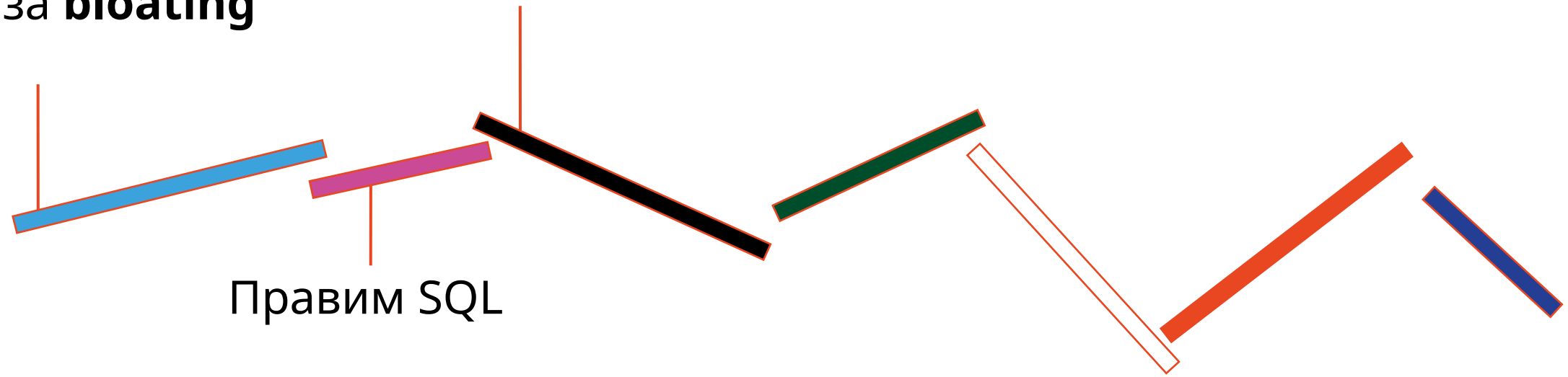
PostgresPro

Тормоза из-
за **bloating**



Индексы **не** используются

Тормоза из-за **bloating**

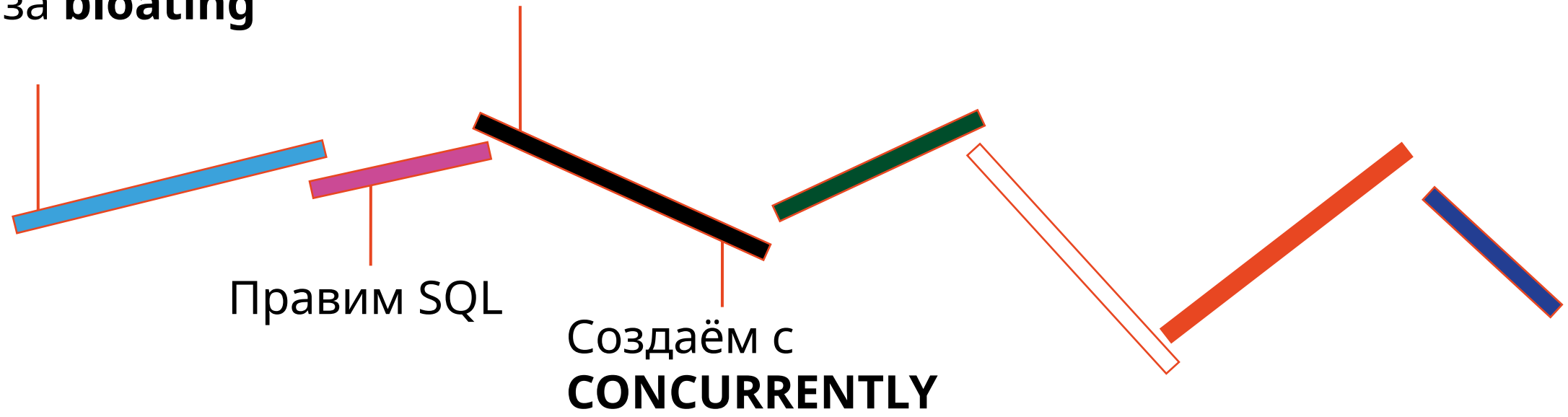


Правим SQL

PostgresPro

Индексы **не** используются

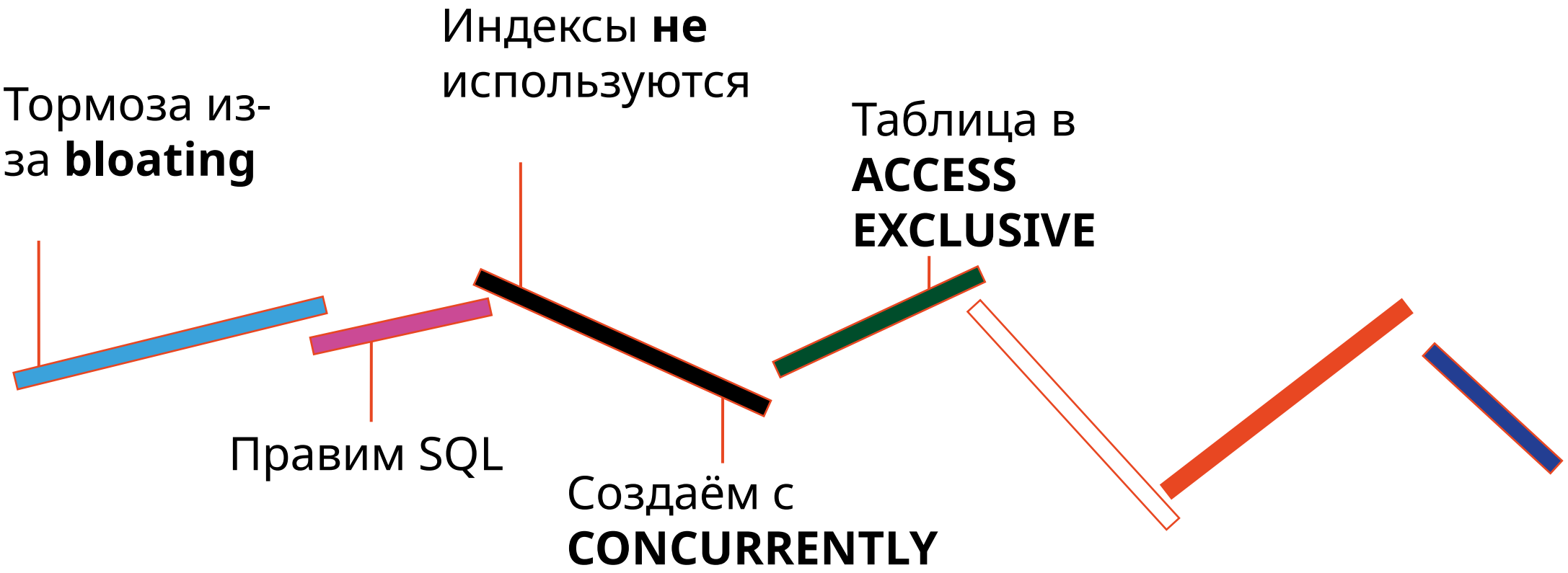
Тормоза из-за **bloating**



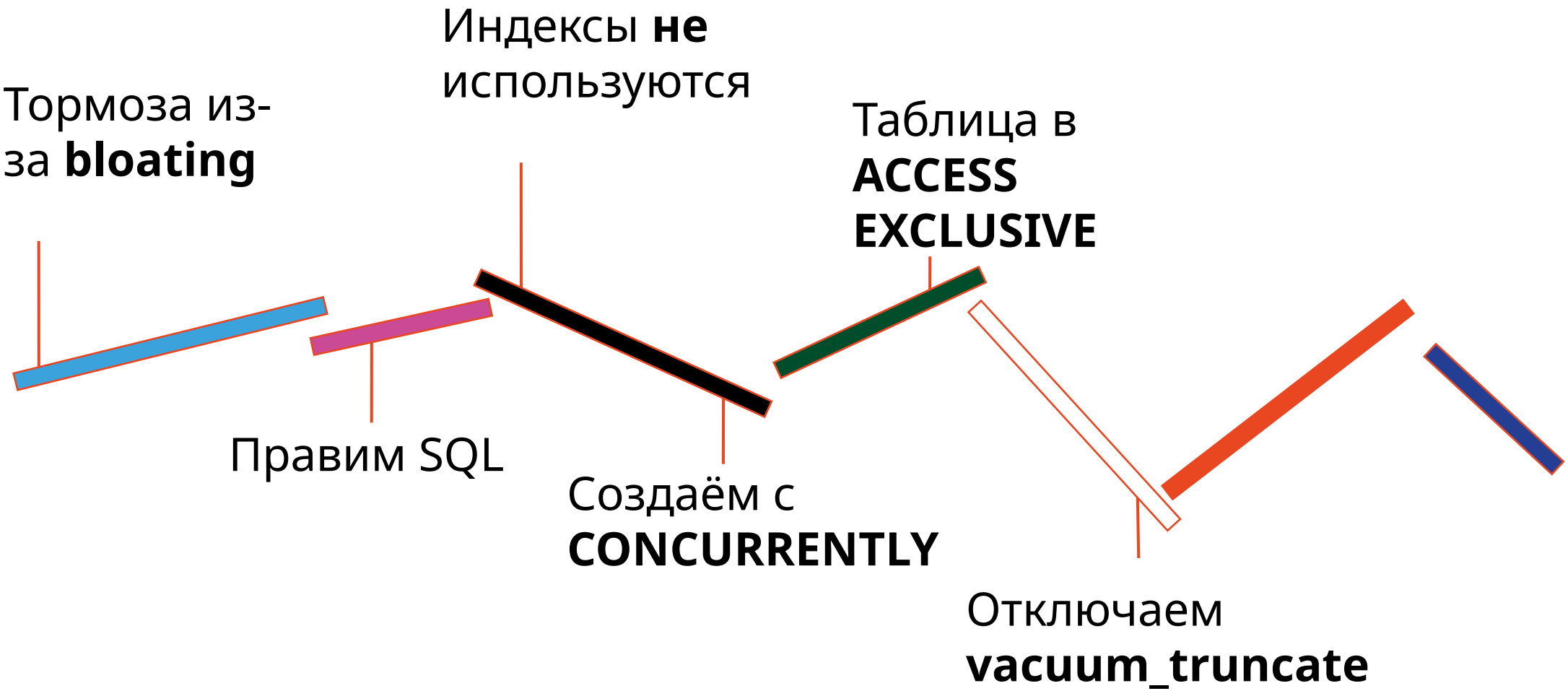
Правим SQL

Создаём с **CONCURRENTLY**

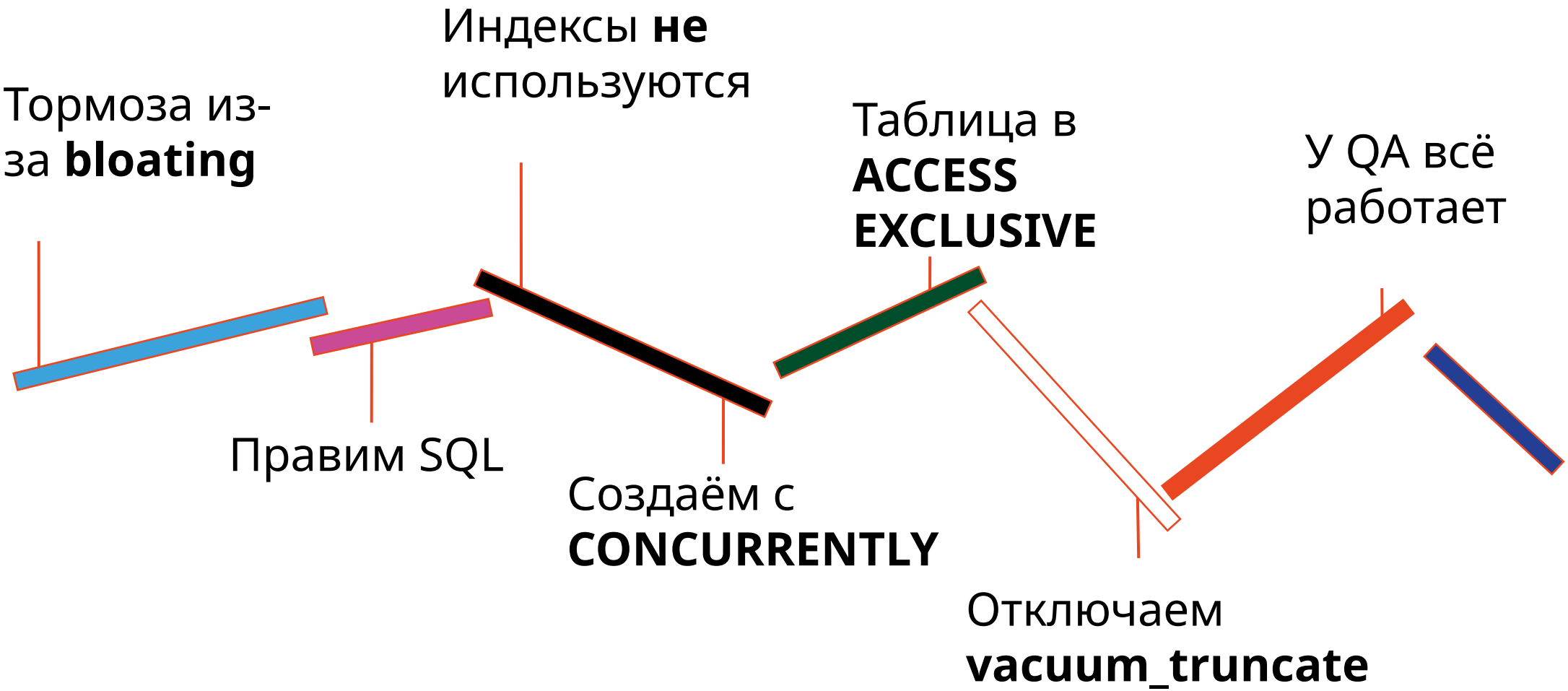
PostgresPro



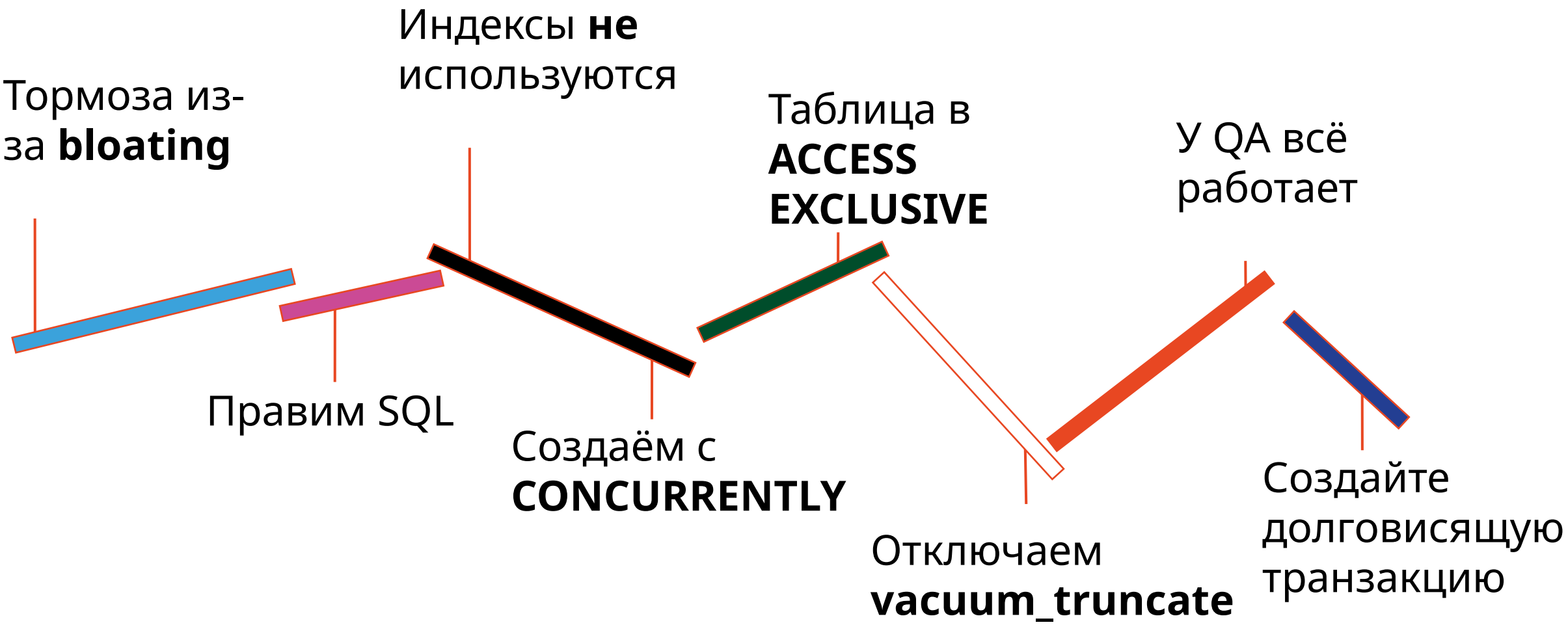
PostgresPro



PostgresPro



PostgresPro



PostgresPro

Полезные ссылки

- Бесплатные книги: https://postgrespro.ru/education/books
  - ”Postgres изнутри” Егор Рогов
  - ”Мониторинг PostgreSQL” Алексей Лесовский
- Бесплатные курсы: https://postgrespro.ru/education/courses
  - DBA2 — MVCC / WAL / блокировки
  - QPT — основы работы планировщик
- Конференция PgConf.Russia / СПб

Спасибо за внимание!

Михаил Жилин

Tg: @mizhka