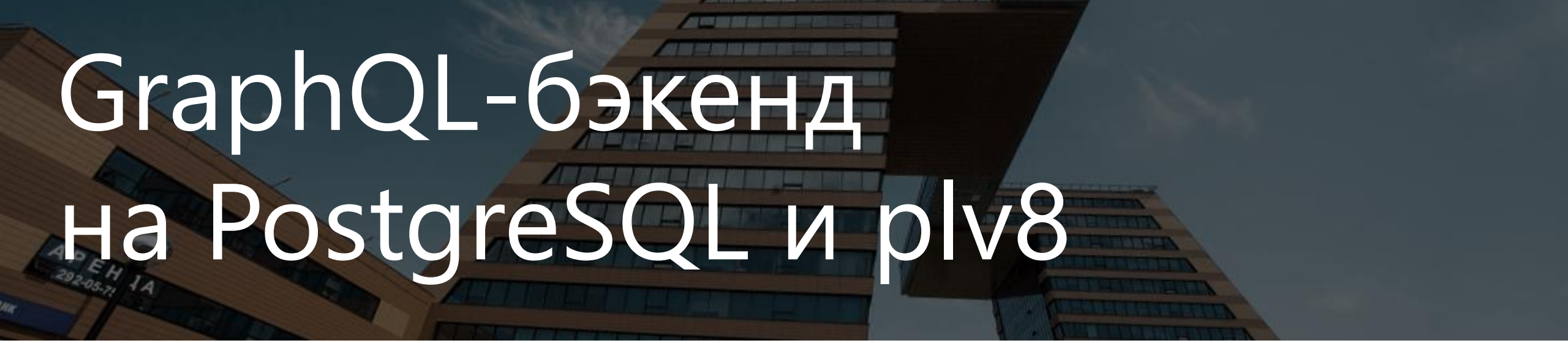


GraphQL-бэкенд на PostgreSQL и plv8

Обо мне

Фадеев Алексей Сергеевич

Старший разработчик .NET

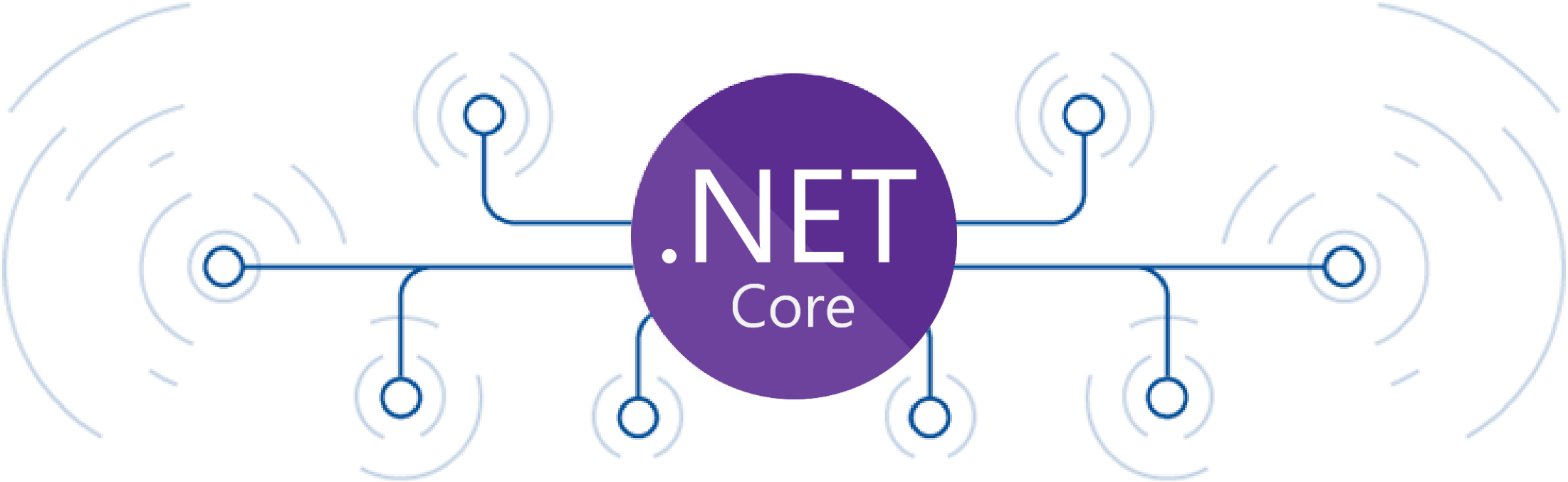
Компания sibedge, г.Томск

Сфера деятельности

Backend-разработка

Работа с СУБД, оптимизация

Используем: PostgreSQL, Microsoft SQL Server


.NET
Core

GraphQL – язык запросов

Разработка Facebook

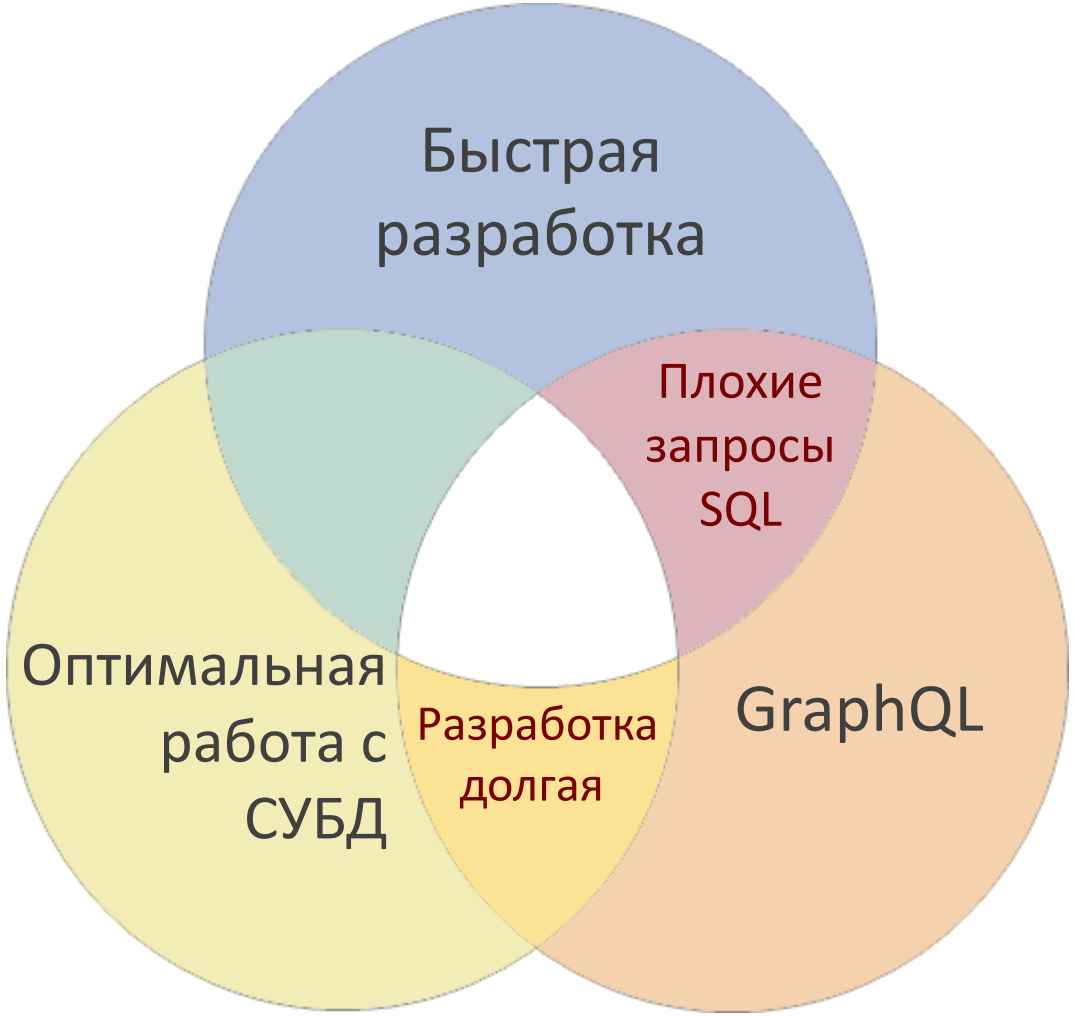
```
query {
  post {
    id
    title
    user {
      name
      email
    }
  }
}
```

```
{
  "data": {
    "post": [
      {
        "id": "ck0qw1hyb0e2v0116rlv3tqeb",
        "title": "janner2k19",
        "user": {
          "name": "Paul Rudd",
          "email": "admin1@gmail.com"
        }
      },
      {
        "id": "ck0qw1u6n0dxq0163kkhdysvc",
        "title": "2k20",
        "user": null
      }
    ]
  }
}
```

GraphQL

```
query {
  user {
    id
    name
    email
    comments {
      id
      text
    }
  }
}
```


```
{
  "data": {
    "user": [
      {
        "id": "ck1tt3zvb0n8t0144323ks031",
        "name": "James Michael Tyler",
        "email": "ssssccv@ukr.net",
        "comments": []
      },
      {
        "id": "ck2cspv6z0a7m0166a3oiq53u",
        "name": "Huyen",
        "email": "huyen@gmail.com",
        "comments": [
          {
            "id": "ck2x7y6390fwt015796sd4pha",
            "text": "Hi 1"
          },
          {
            "id": "ck2x7y63a0fwu0157xy6vo8tx",
            "text": "Add new comment 123"
          },
          {
            "id": "ck2x7y63a0fwv0157k1qici1w",
            "text": "awgawg"
          }
        ]
      },
      {
        "id": "ck2ks14gy1pgi0155u83ek72f",
        "name": "novy",
        "email": "s",
        "comments": []
      }
    ]
  }
}
```


Быстрая разработка
Плохие запросы SQL
Оптимальная работа с СУБД
Разработка долгая
GraphQL

GraphQL-запрос

Динамический набор полей

Иерархия любой вложенности

```
query {
  post {
    id
    title
    user {
      name
      email
      comments {
        text
        createdAt
      }
    }
    text
  }
}
```

GraphQL-запрос

Динамический набор полей

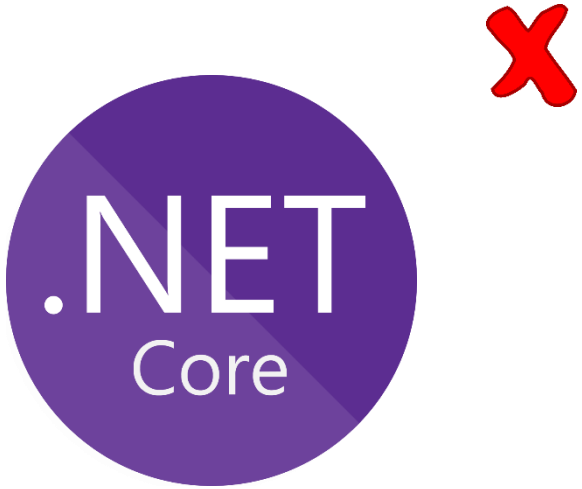
Иерархия любой вложенности

Статическая типизация

Плоские таблицы

GraphQL-запрос

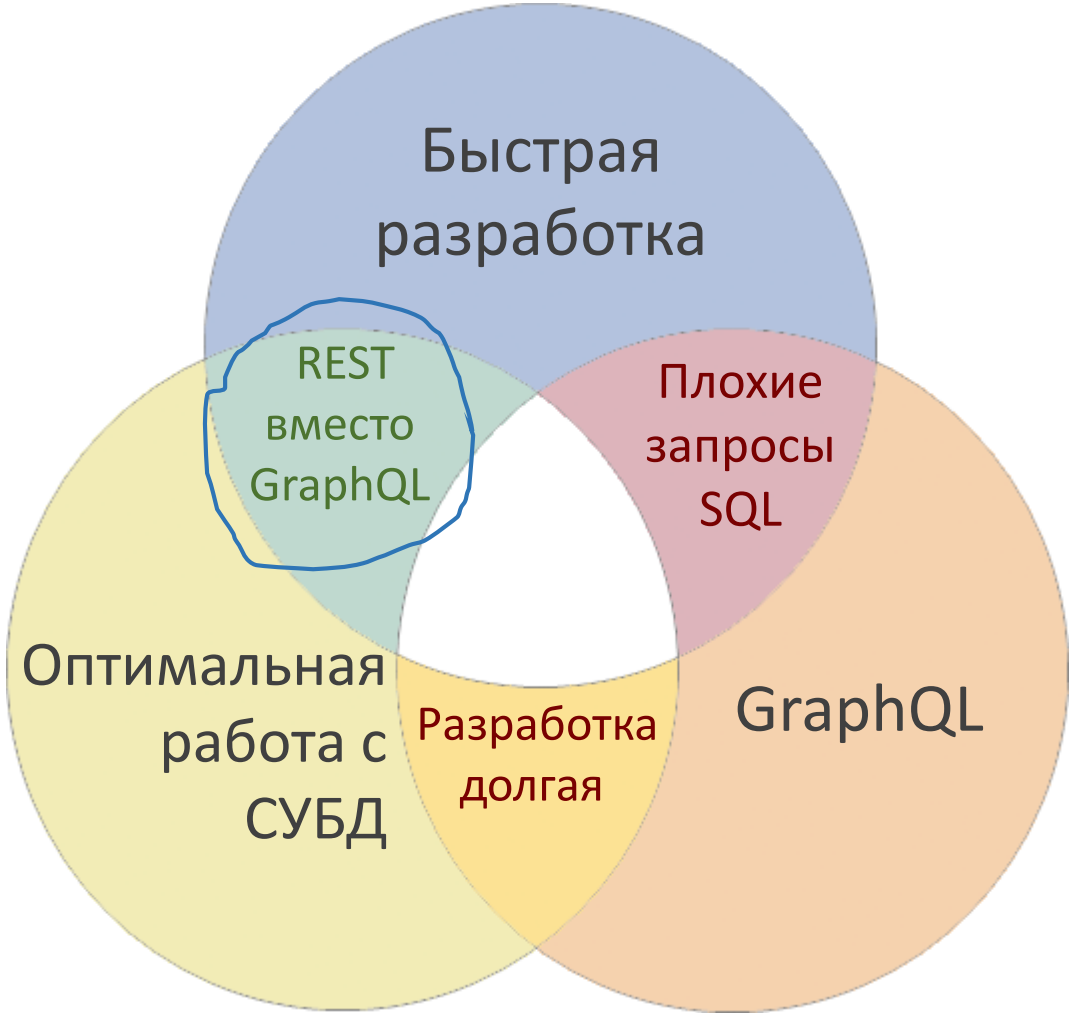
Динамический набор полей



Статическая типизация

Иерархия любой вложенности

Плоские таблицы


Быстрая разработка
REST вместо GraphQL
Плохие запросы SQL
Оптимальная работа с СУБД
Разработка долгая
GraphQL

ВСЁ!

ВСЁ!

PostgreSQL – объектно-реляционная СУБД

JSONB

plv8

Бэкенд внутри СУБД

plv8 - javascript

Функция на plv8

1. Разбираем входной объект
2. Выполняем запросы
3. Собираем объект-результат
4. Возвращаем JSONB

```sql
CREATE MATERIALIZED VIEW IF NOT EXISTS schema_columns
AS
SELECT column_name, table_name FROM information_schema.columns;


CREATE MATERIALIZED VIEW IF NOT EXISTS schema_foreign_keys
AS
SELECT kcu.column_name, ccu.table_name AS foreign_table_name,
       ccu.column_name AS foreign_column_name, tc.table_name
    FROM information_schema.table_constraints AS tc
    JOIN information_schema.key_column_usage AS kcu
       ON tc.constraint_name = kcu.constraint_name
       AND tc.table_schema = kcu.table_schema
    JOIN information_schema.constraint_column_usage AS ccu
       ON ccu.constraint_name = tc.constraint_name
       AND ccu.table_schema = tc.table_schema
    WHERE tc.constraint_type = 'FOREIGN KEY';
```

```
CREATE OR REPLACE FUNCTION execute_qraphql(gqlquery JSONB, schema text)
RETURNS JSONB
AS $$
   function viewTable(table, tableName, result, where, level)
   {
     var tableKeys = Object.keys(table);

     var fkQuery = "SELECT column_name, foreign_table_name, foreign_column_name "
           + "FROM schema_foreign_keys "
           + "WHERE table_name='"
           + tableName + "';";

     var fkRows = plv8.execute(fkQuery);
     fkRows = fkRows.filter(item => tableKeys.includes(item.foreign_table_name.toLowerCase()));

     var fkFields = fkRows.map(a => a.column_name);
     var allFields = fkFields.concat(tableKeys);
     var allFieldsFiltered = allFields.filter((item, pos) => allFields.indexOf(item) === pos);
   …

$$ LANGUAGE plv8;
```

Запрос – входные данные

```
{
  "Bottle": {
    "id": {},
    "bottle_picture": {},
    "bottle_size_id": {},
    "bottle_front_label": {},
    "beer": {
      "id": {},
      "kind_id": {},
      "style": {
        "id": {},
        "original_gravity": {}
      }
    }
  }
}
```

Альтернативный вариант

Запрос с агрегатными функциями JSONB

```
WITH s AS
  (SELECT s.id, original_gravity FROM style s),
br AS
  (SELECT br.id, kind_id, s AS style, style _id FROM beer br
    LEFT JOIN s ON w.style_id=s.id),
bng AS
  (SELECT b.id, br as beer, beer_id, bottle_picture,
       bottle_size_id, bottle_front_label FROM bottle b
    LEFT JOIN br ON b.beer_id=br.id),
bg AS (SELECT jsonb_agg(bng) AS "Bottle" FROM bng),

result AS
  (SELECT * FROM bg)

SELECT jsonb_agg(result)->0 FROM result;
```

JSON

```
{
  "Bottle": {
    "id": {},
    "bottle_picture": {},
    "bottle_size_id": {},
    "bottle_front_label": {},
    "beer": {
      "id": {},
      "kind_id": {},
      "style": {
        "id": {},
        "original_gravity": {}
      }
    }
  }
}
```

GraphQL-запрос

```
query {
  Bottle {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      kind_id
      style {
        id
        original_gravity
      }
    }
  }
}
```

JSON

```
{
  "Bottle": {
    "id": {},
    "bottle_picture": {},
    "bottle_size_id": {},
    "bottle_front_label": {},
    "beer": {
      "id": {},
      "kind_id": {},
      "style": {
        "id": {},
        "original_gravity": {}
      }
    }
  }
}
```

GraphQL-запрос

```
query {
  Bottle (filter: { id: { less: 1000 } }) {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      kind_id
      style
        id
        original_gravity
      }
    }
  }
}
```

Парсинг GraphQL-запроса

npm-пакет graphql-tag

PostgreSQL + webpack

Способ с jsonb_agg

Динамический набор полей

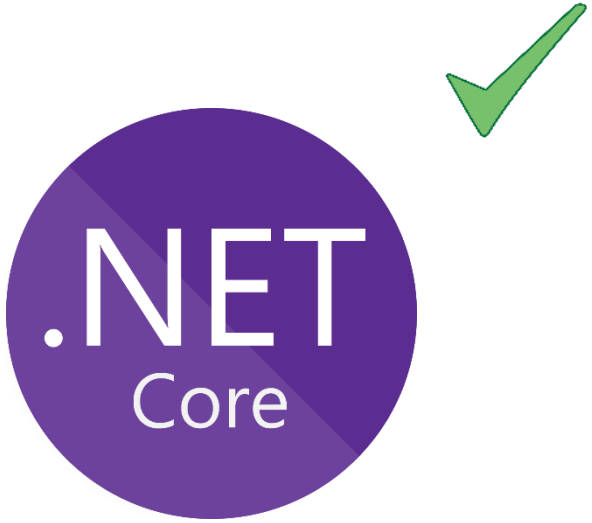
Доступна библиотека-парсер GraphQL

Иерархия любой вложенности

Результат запроса – готовый JSON

Способ с jsonb_agg

Динамический набор полей



Доступна библиотека-парсер GraphQL

Иерархия любой вложенности

Результат запроса – готовый JSON

Способ с jsonb_agg

На стороне бэкенда
(.NET Core, Java …)

Проще отладка,
поддержка

На стороне БД
(plv8)

Универсальность,
переносимость

Мутации

GraphQL-запросы на изменение данных

```
mutation {
  addUser(name: "Петров", typeId: 5) {
    id
    name
    type {
      name
    }
  }
}
```

```
{
  "data": {
    "addUser": {
      "id": 1374,
      "name": "Петров",
      "type": {
        "name": "Оператор"
      }
    }
  }
}
```

Мутации

GraphQL-запросы на изменение данных

```
mutation {
  addUser(name: $name, typeId: $typeId) {
    id
    name
    type {
      name
    }
  }
}
```

```
{
  "Name": "Петров",
  "TypeId": 5
}
```

```
{
  "data": {
    "addUser": {
      "id": 1374,
      "name": "Петров",
      "type": {
        "name": "Оператор"
      }
    }
  }
}
```

Сравнение производительности

plv8: запросы к таблицам,
сбор объекта

Один SQL-запрос с
агрегатными функциями

Измерения:

Автоматизировано с помощью ПО

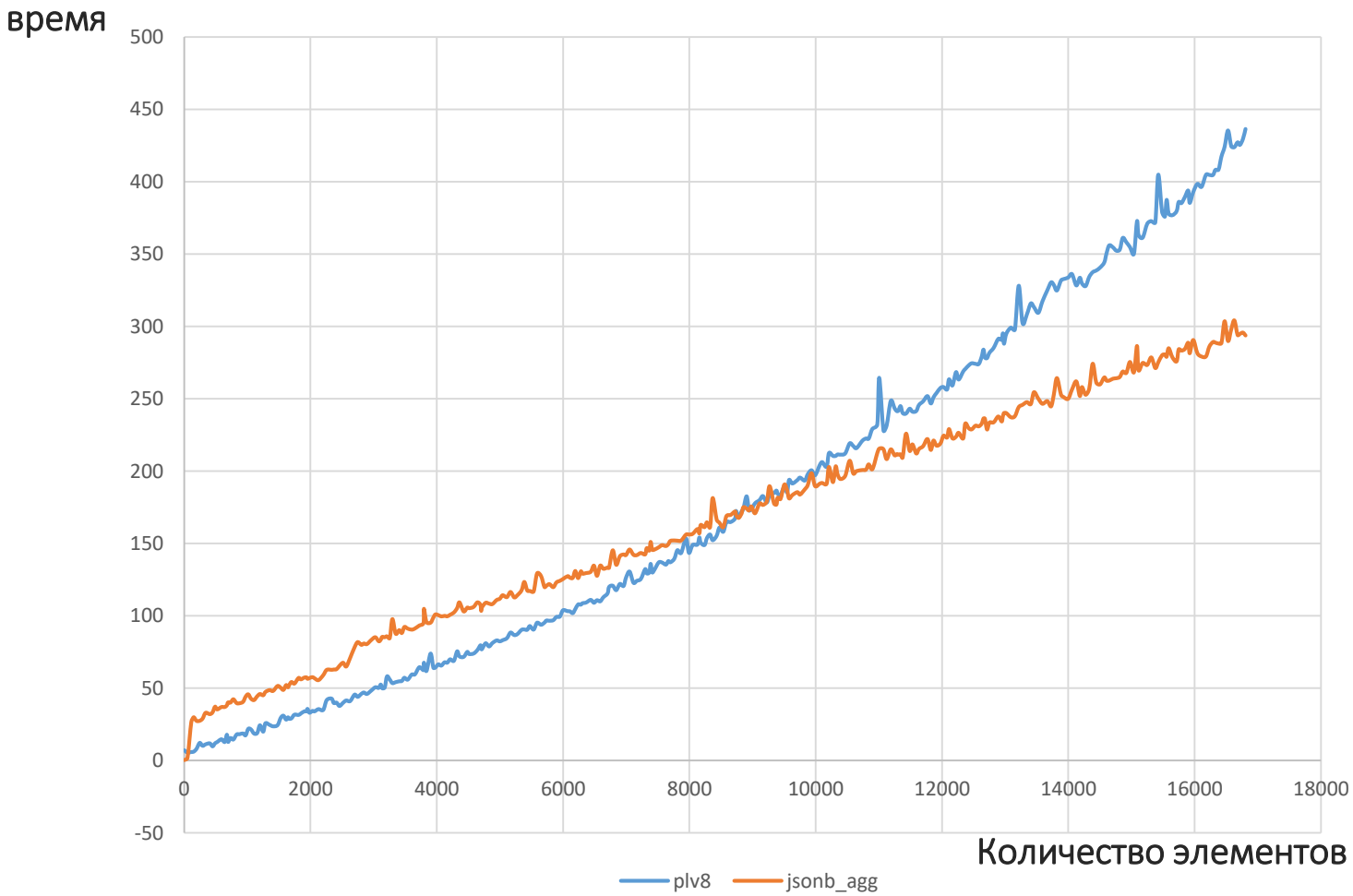
EXPLAIN ANALYZE – Execution Time

Каждый запрос: 4 раза

Отбрасываем самый быстрый и самый медленный

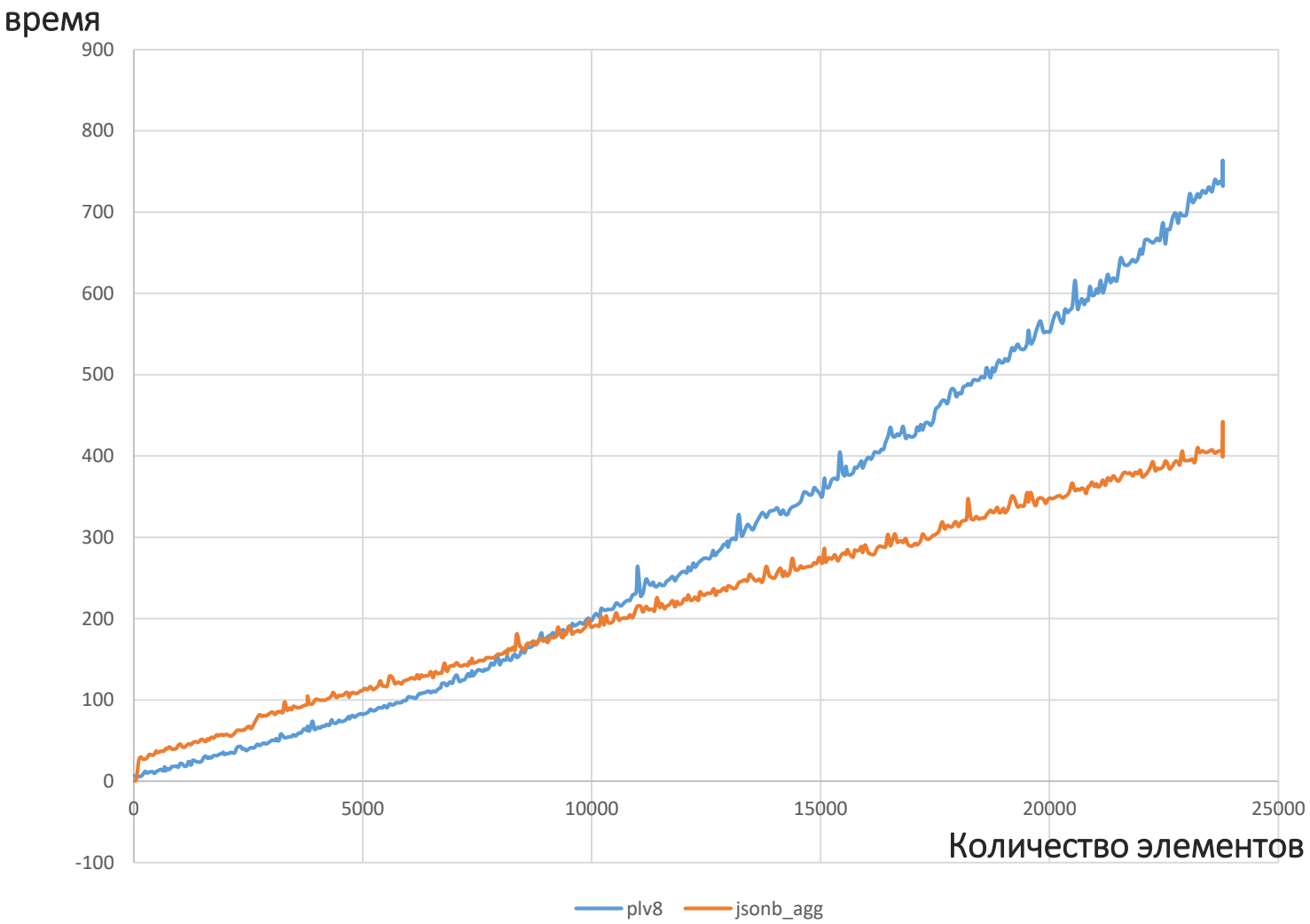
Сравнение производительности

```
{
  Bottle {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      kind_id
      style {
        id
        original_gravity
      }
    }
  }
}
```



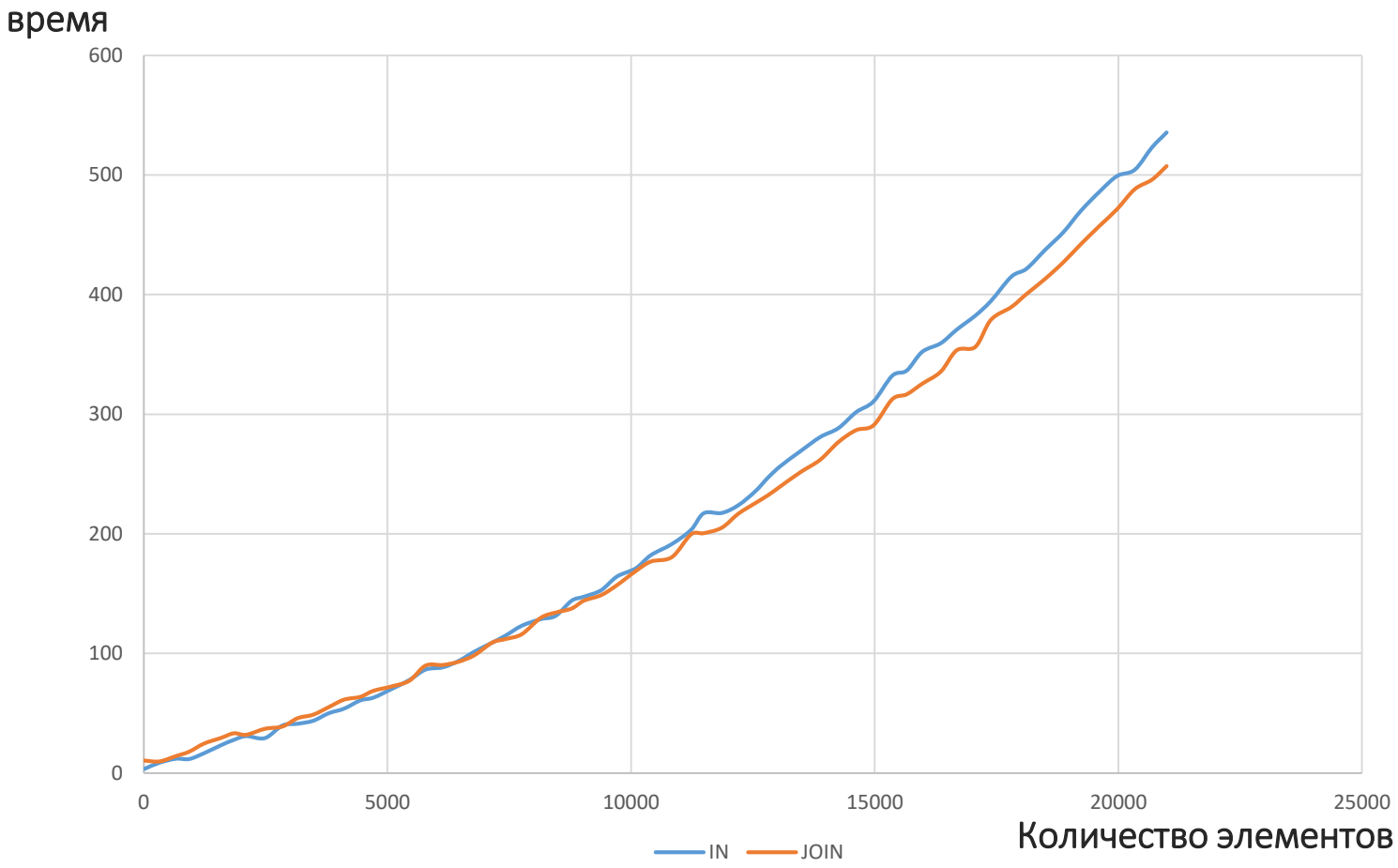
Сравнение производительности

```
{
  Bottle {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      kind_id
      style {
        id
        original_gravity
      }
    }
  }
}
```

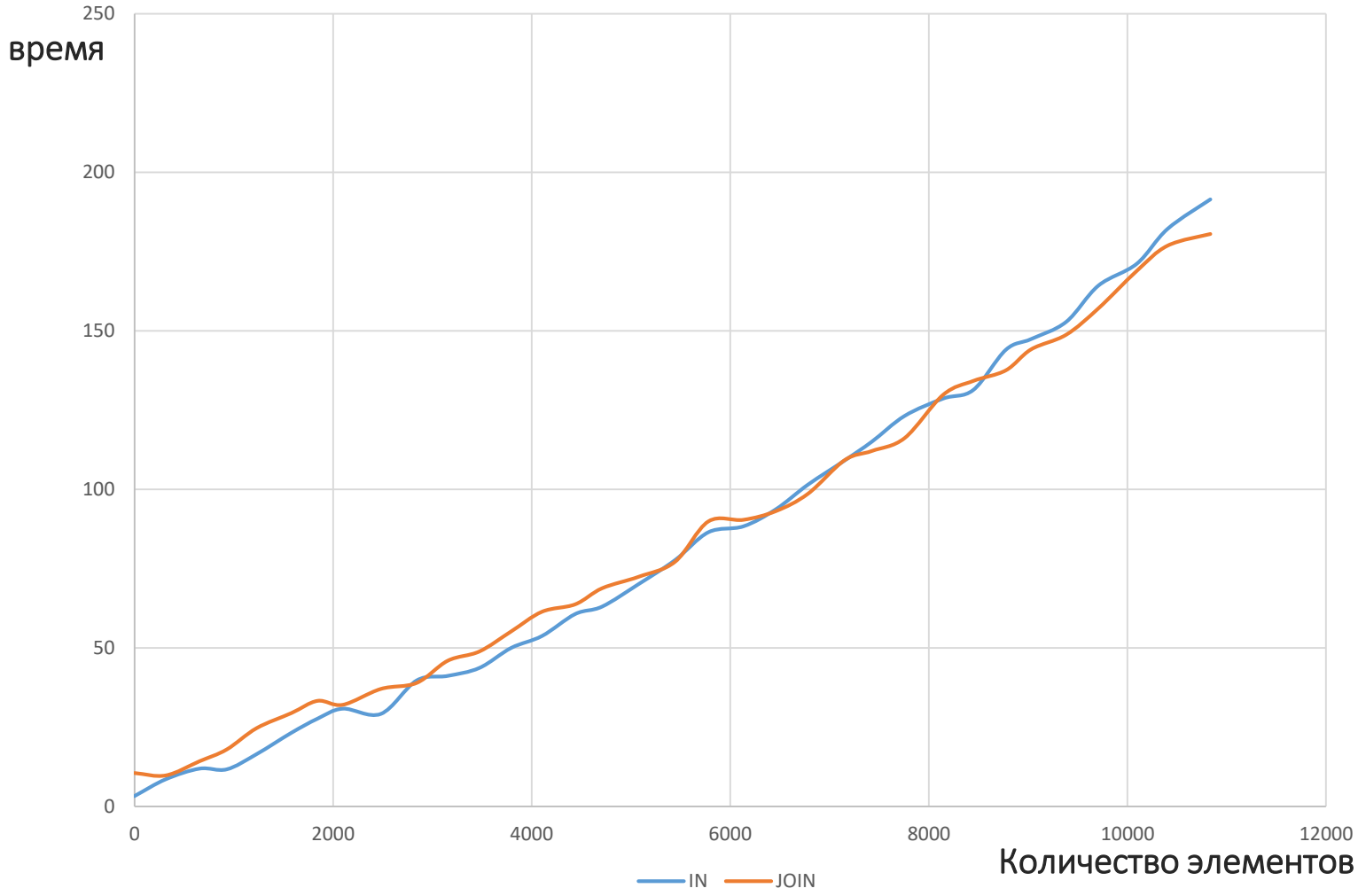


```
SELECT kind_id, id FROM beer WHERE id IN
  (5111, 4595, 5211, 4870, 5471, 5294, 4690,
   4691, 5314, 4925, 5026, 5034, 5374, 5307,
   4688, 4979, 4998, 5279, 4537, 5097, 4562,
   5107, 4601, 4677, 4634, 4878, 5222, 5350,
   4560, 4559, 5383, 4721, 5206, 4642, 5062,
   4763, 5239, 5242, 5105, 5429, 4805, 5371,
   5304, 4817, 4522, 5052, 4810, 4656, 4809,
   5198, 4761, 4556, 4786, 5053, 4957, 5362,
   5078, 5190, 4534, 4535, 4564, 5315, 5433,
   5432, 5451, 5028, 5144, 4613, 5327, 5480,
   5375, 5452, 5473, 4670, 4728, 4811, 5370);
```

IN (...) vs JOIN


время
600
500
400
300
200
100
0
0
5000
10000
15000
20000
25000
Количество элементов
IN
JOIN

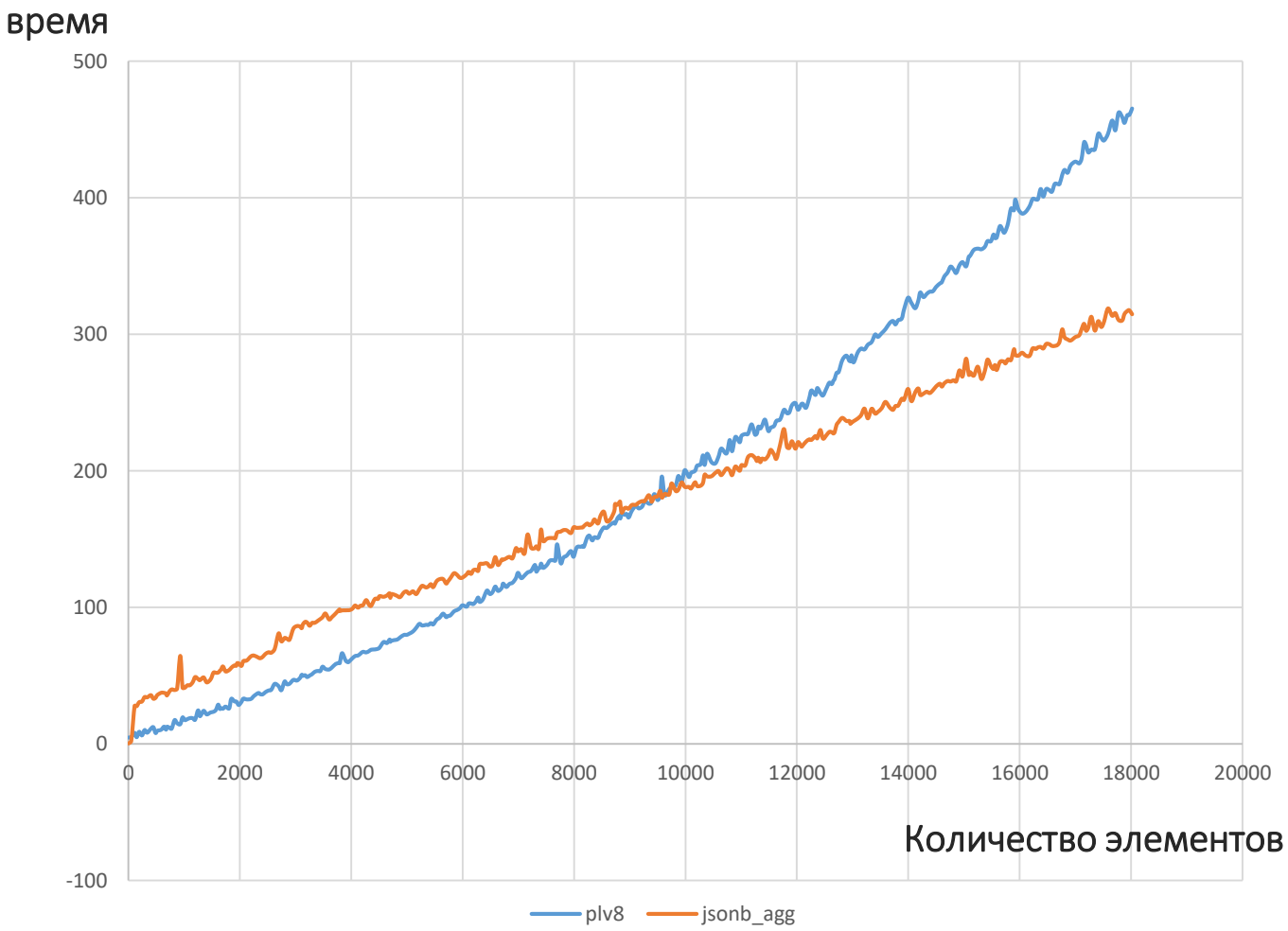
IN (...) vs JOIN


250
время
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000
Количество элементов
IN
JOIN

Сравнение производительности

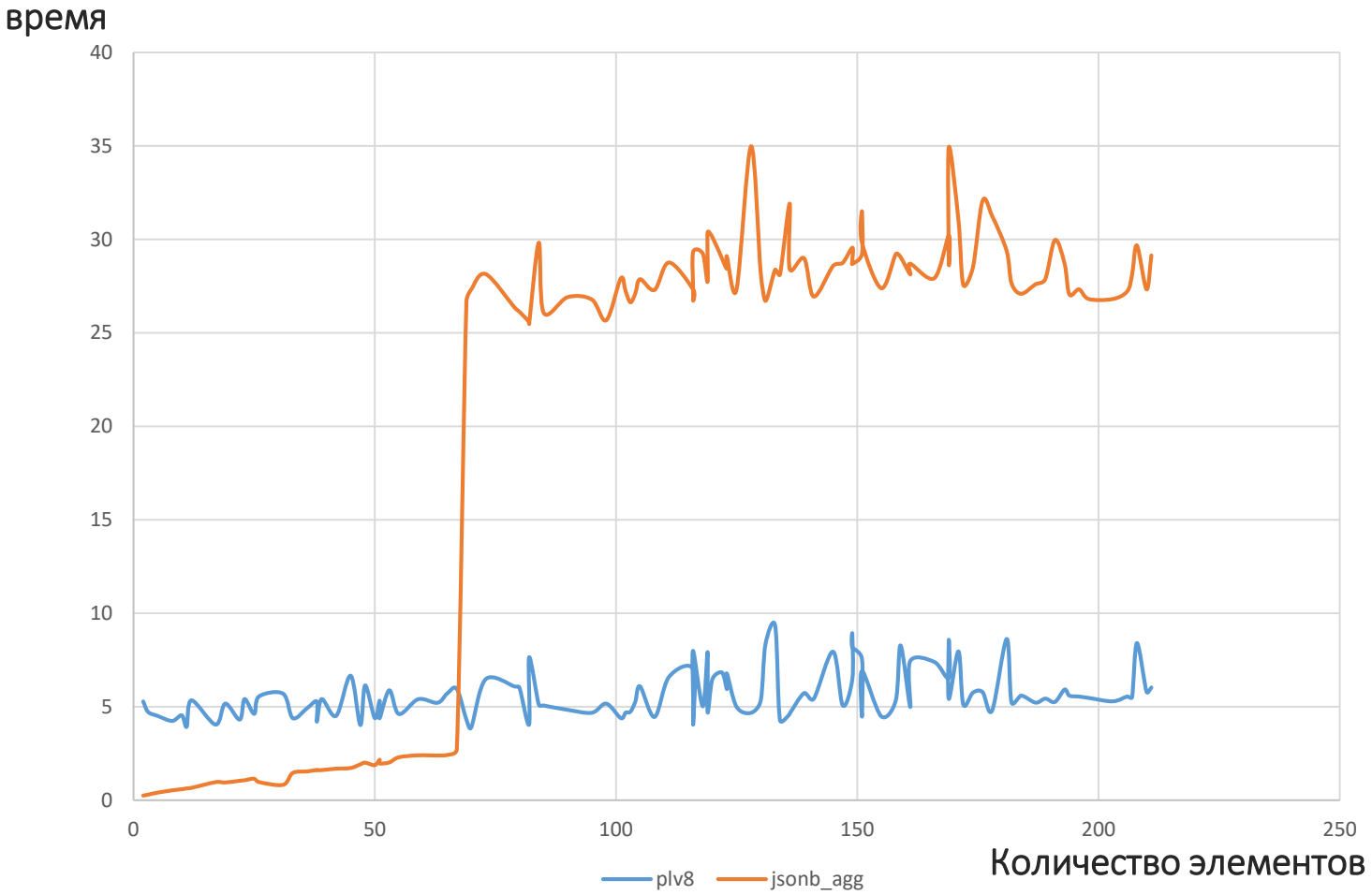
```
{
  Bottle {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      kind_id
      style {
        id
        original_gravity
      }
    }
  }
}
```

8800 -> 9500



Сравнение производительности

```
{
  Bottle {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      kind_id
      style {
        id
        original_gravity
      }
    }
  }
}
```



```
Aggregate  (cost=2252.93..2252.94 rows=1 width=32) (actual time=2.085..2.085 rows=1 loops=1)
 ->  Aggregate  (cost=2252.91..2252.92 rows=1 width=32) (actual time=1.363..1.364 rows=1 loops=1)
     ->  Nested Loop Left Join  (cost=11.55..2251.99 rows=366 width=115) (actual time=0.047..0.487 rows=67 loops=1)
         ->  Bitmap Heap Scan on "Bottle" b  (cost=11.13..442.24 rows=366 width=83) (actual time=0.022..0.072 rows=67 loops=1)
             Recheck Cond: (id < 4708)
             Heap Blocks: exact=18
             ->  Bitmap Index Scan on "Index_Bottle_BottleId"  (cost=0.00..11.04 rows=366 width=0) (actual time=0.013..0.013 rows=67 loops=1)
                 Index Cond: (id < 4708)
         ->  Nested Loop Left Join  (cost=0.43..4.93 rows=1 width=36) (actual time=0.005..0.006 rows=1 loops=67)
             ->  Index Scan using "Index_beer_beerId" on "beer" w  (cost=0.29..4.78 rows=1 width=12) (actual time=0.002..0.003 rows=1 loops=67)
                 Index Cond: (id = b.beer_id)
             ->  Index Scan using "Index_style_styleId" on "style" s  (cost=0.14..0.16 rows=1 width=36) (actual time=0.001..0.001 rows=1 loops=67)
                 Index Cond: (id = w.sweetness_level_id)
Planning Time: 0.837 ms
Execution Time: 2.218 ms
```

```
Aggregate  (cost=2234.24..2234.25 rows=1 width=32) (actual time=32.165..32.165 rows=1 loops=1)
 ->  Aggregate  (cost=2234.22..2234.23 rows=1 width=32) (actual time=31.722..31.722 rows=1 loops=1)
     ->  Hash Right Join  (cost=448.23..2233.30 rows=367 width=115) (actual time=23.577..31.208 rows=68 loops=1)
         Hash Cond: (w.id = b.beer_id)
         ->  Hash Left Join  (cost=1.38..1193.35 rows=47154 width=36) (actual time=0.091..26.696 rows=47154 loops=1)
             Hash Cond: (w.sweetness_level_id = s.id)
             ->  Seq Scan on "beer" w  (cost=0.00..1043.54 rows=47154 width=12) (actual time=0.053..6.300 rows=47154 loops=1)
             ->  Hash  (cost=1.17..1.17 rows=17 width=36) (actual time=0.028..0.028 rows=17 loops=1)
                 Buckets: 1024  Batches: 1  Memory Usage: 10kB
                 ->  Seq Scan on "style" s  (cost=0.00..1.17 rows=17 width=36) (actual time=0.010..0.019 rows=17 loops=1)
         ->  Hash  (cost=442.26..442.26 rows=367 width=83) (actual time=0.146..0.146 rows=68 loops=1)
             Buckets: 1024  Batches: 1  Memory Usage: 12kB
             ->  Bitmap Heap Scan on "Bottle" b  (cost=11.13..442.26 rows=367 width=83) (actual time=0.034..0.113 rows=68 loops=1)
                 Recheck Cond: (id < 4709)
                 Heap Blocks: exact=18
                 ->  Bitmap Index Scan on "Index_Bottle_BottleId"  (cost=0.00..11.04 rows=367 width=0) (actual time=0.018..0.018 rows=68 loops=1)
                     Index Cond: (id < 4709)
Planning Time: 1.366 ms
Execution Time: 32.321 ms
```

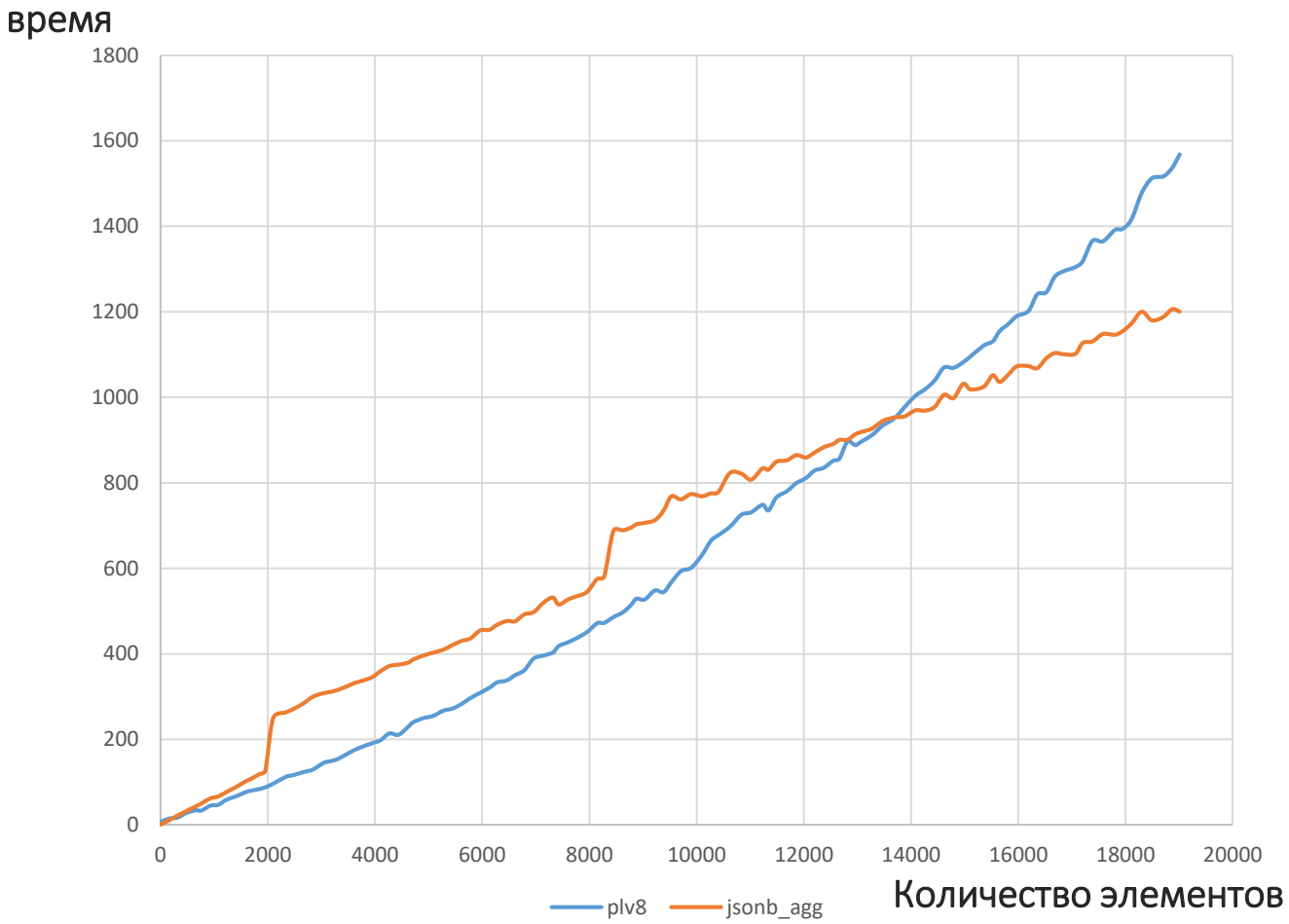
Сравнение производительности

```
{
  Bottle
  {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      record_last_update
      producer {
        id
        trade_name
      }
      beername {
        id
        name
      }
      terminalGravity {
        id
        level
      }
      beertype {
        id
        volume_by_type
      }
      country {
        id
        iso
        country_code
      }
      beerownertype { …
      }
      style { …
      }
    }
  }
}
```

```
WITH sl AS (SELECT s.id, original_gravity FROM hw."style" s),
v AS (SELECT v.id, level FROM hw."terminal_gravity" v),
c AS (SELECT c.id, iso, country_code FROM hw."country" c),
p AS (SELECT p.id, trade_name FROM hw."producer" p),
wn AS (SELECT wn.id, name FROM hw."beerName" wn),
wt AS (SELECT wt.id, volume_by_type FROM hw."beer_type" wt),
wo AS (SELECT wo.id, name FROM hw."beer_owner_type" wo),
w AS (SELECT w.id, p AS producer, producer_id, c AS country, country_id,
      v AS terminalGravity, terminal_gravite_id, wn AS beername, beer_name_id,
wt AS beertype, beer_type_id, wo AS beerownertype, beer_owner_type_id,
       sl AS style, style_id, FROM hw."beer" w
    LEFT JOIN sl ON w.style_id=sl.id
    LEFT JOIN c ON w.country_id=c.id
    LEFT JOIN p ON w.producer_id=p.id
    LEFT JOIN wn ON w.beer_name_id=wn.id
    LEFT JOIN wt ON w.beer_type_id=wt.id
    LEFT JOIN wo ON w.beer_owner_type_id=wo.id
    LEFT JOIN v ON w. original_gravity_id=v.id),
bng AS (SELECT b.id, w as beer, beer_id, bottle_picture, bottle_size_id,
bottle_front_label
        FROM hw."Bottle" b
    LEFT JOIN w ON b.beer_id=w.id),
bg AS (SELECT jsonb_agg(bng) AS "Bottle" FROM bng),
result AS
  (SELECT * FROM bg)
SELECT jsonb_agg(result)->0 FROM result
```

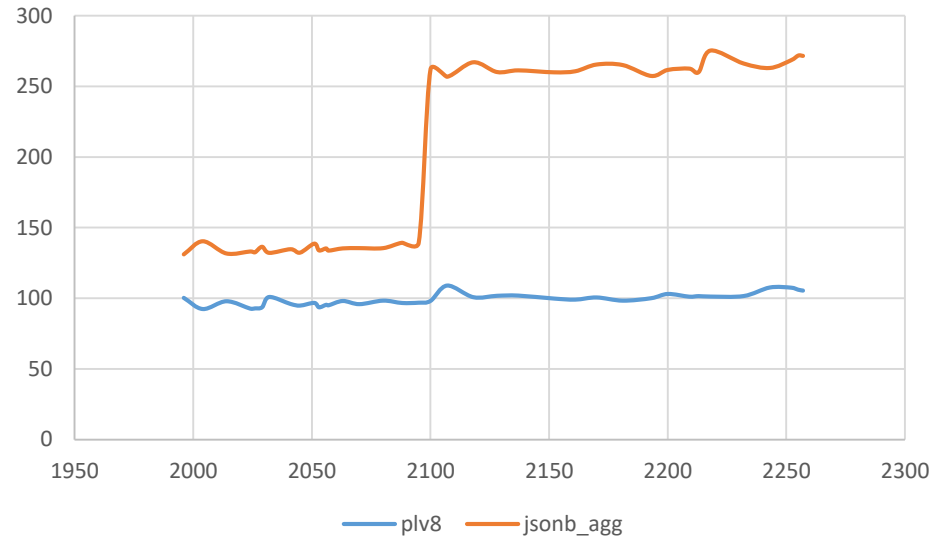
Сравнение производительности

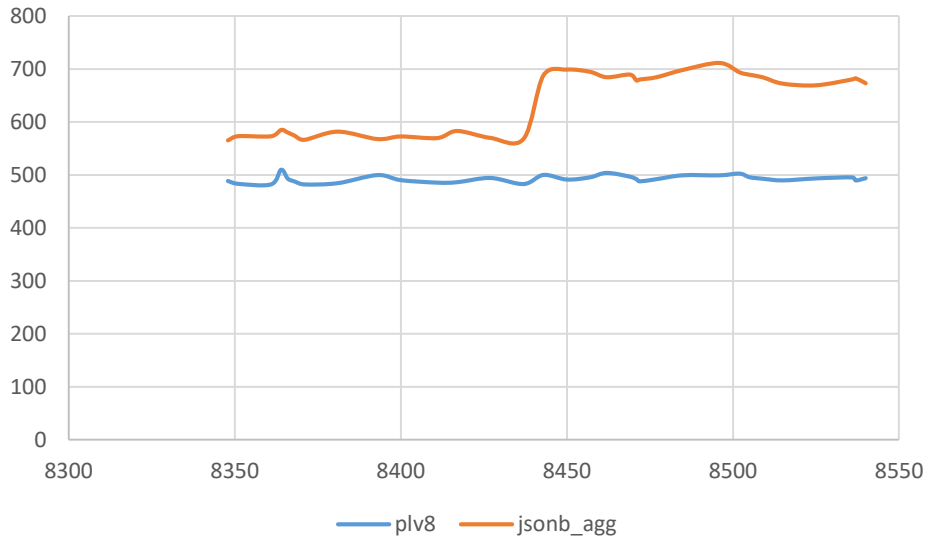
```
{
  Bottle
  {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      record_last_update
      producer {
        id
        trade_name
      }
      beername {
        id
        name
      }
      terminalGravity {
        id
        level
      }
      beertype {
        id
        volume_by_type
      }
      country {
        id
        iso
        country_code
      }
      beerownertype { …
      }
      style { …
      }
    }
  }
}
```



Сравнение производительности

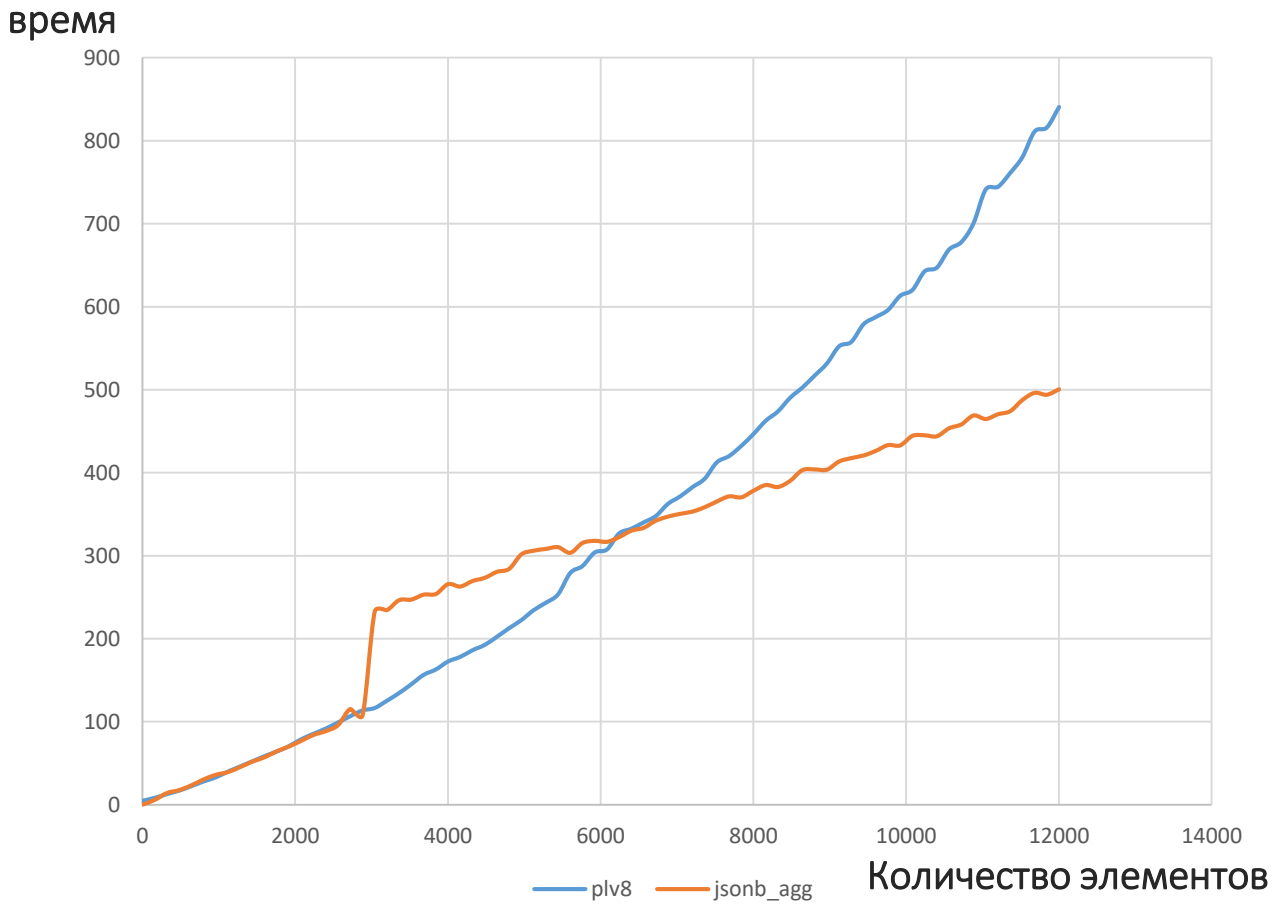
```
{
  Bottle
  {
    id
    bottle_picture
    bottle_size_id
    bottle_front_label
    beer {
      id
      record_last_update
      producer {
        id
        trade_name
      }
      beername {
        id
        name
      }
      terminalGravity {
        id
        level
      }
      beertype {
        id
        volume_by_type
      }
      country {
        id
        iso
        country_code
      }
      beerownertype { …
      }
      style { …
      }
    }
  }
}
```





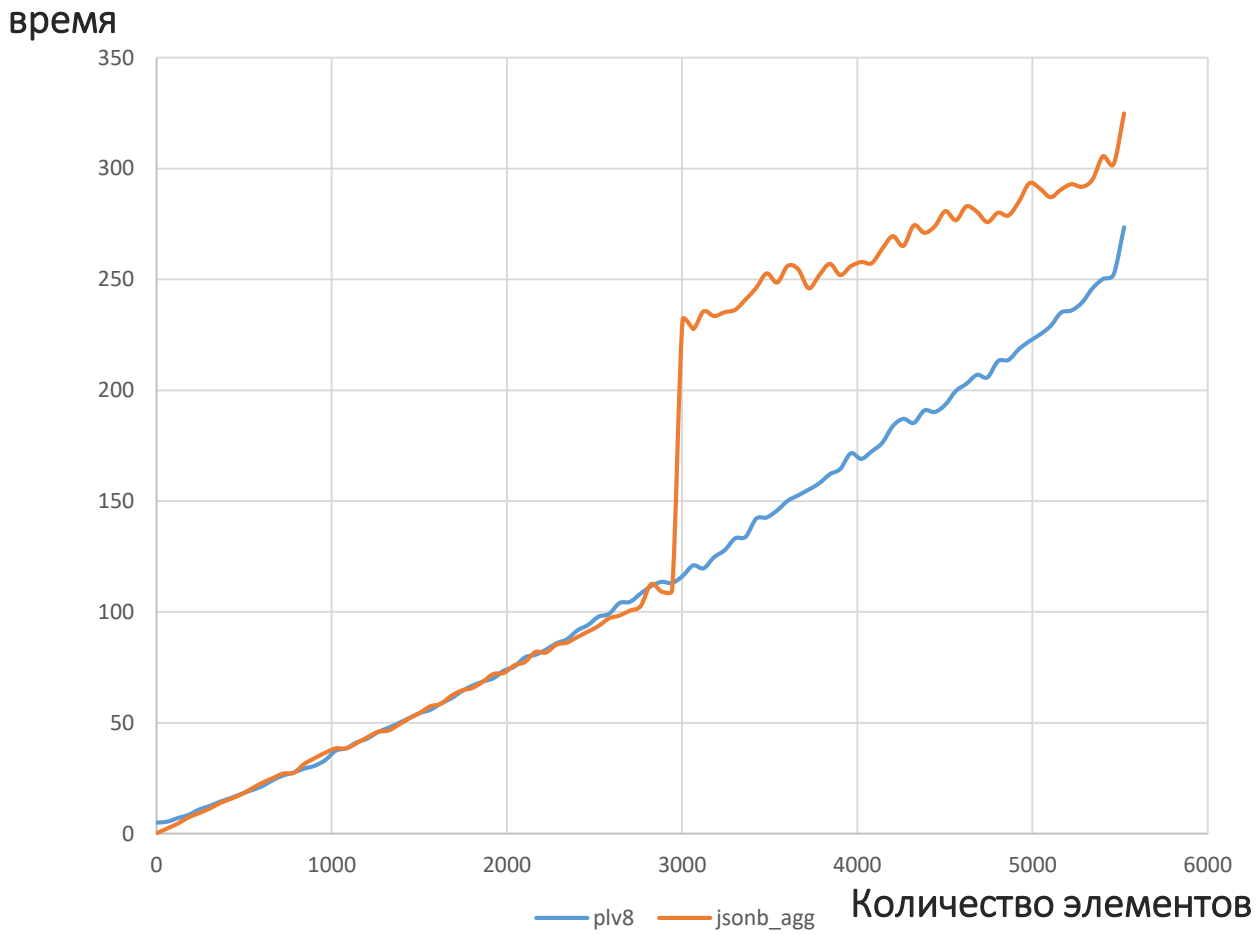
Сравнение производительности

```
{
  Batch {
    id
    batch_barcode
    productcase {
      id
      case_height
      product {
        id
        record_last_update
        bottle {
          id
          bottle_picture
          bottle_size_id
          bottle_front_label
          beer {
            id
            final_gravity_id
            style {
              id
              original_gravity
            }
          }
        }
      }
    }
  }
}
```



Сравнение производительности

```
{
  Batch {
    id
    batch_barcode
    productcase {
      id
      case_height
      product {
        id
        record_last_update
        bottle {
          id
          bottle_picture
          bottle_size_id
          bottle_front_label
          beer {
            id
            final_gravity_id
            style {
              id
              original_gravity
            }
          }
        }
      }
    }
  }
}
```



plv8

запросы к таблицам,
сбор объекта

Быстрее на малых и средних выборках

Скорость зависит от общего числа данных

Внутри БД

Чтение и запись

jsonb_agg

Один SQL-запрос с агрегатными функциями

Быстрее на больших выборках

Скорость зависит от иерархичности (скачки)

На бэкенде / внутри БД

Только чтение

Планы

Мутации

Поддержка полей JSONB

Конфигурация связей

Создание расширения для GraphQL

Проблемы GraphQL

Нет разделения, данные в одной куче

Должна ли волновать фронтэнд структура БД?

Разграничение прав доступа

Когда использовать GraphQL?

Требования заказчика/генподрядчика

Простая логика: данные как есть

Нехватка времени/ресурсов

Сложно/дорого обновлять бэкенд

В качестве средства отладки

Сборка и установка plv8

https://plv8.github.io

https://github.com/plv8/plv8/archive/v2.3.14.tar.gz

```
$ make
$ make install
```

Около 6 ГБ

Готовая сборка для Windows:

http://www.postgresonline.com/journal/archives/367-PLV8-binaries-for-PostgreSQL-9.6-windows-both-32-bit-and-64-bit.html

Контакты

FadeevAS@sibedge.com

https://vk.com/fadeev

https://www.facebook.com/alexey.fadeev.3745

Материалы

http://graphql.alexfadeev.net


КОНЕЦ