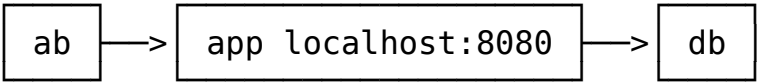


# Отчет о нагрузочном тестировании

## 1. Схема стенда



Apache Benchmark тестирует **http://127.0.0.1:8080/v1/search/%D1%91%D1%82%D1%80**

- URL: **http://127.0.0.1:8080/v1/search/**
- Аргумент поиска (GET параметр после search, URL-encoded): «ётр»

Дополнительно к аргументам ab добавляется опция -C token=... для успешной авторизации.

SQL запрос для поиска:

```
let sql = "SELECT * FROM users
  WHERE email LIKE ($1)
  OR first_name LIKE ($1)
  OR second_name LIKE ($1)
  OR city LIKE ($1)
  ORDER BY user_id";
let us = sqlx::query_as::(<, User> (sql)
  .bind(format!("{0}", text))
  .fetch_all(&state.db)
  .await?;
```

API возвращает все совпадения без пагинации.

## 2. Результаты до создания индекса

### 2.1. До создания индекса, app\_pg\_max\_connections=5

Результаты представлены в виде таблицы.

| Тест               | Failed Reqs | RPS   | Request time (ms) |       |       | Total time (s) |
|--------------------|-------------|-------|-------------------|-------|-------|----------------|
|                    |             |       | Min               | Mean  | Max   |                |
| ab -n 100 -c 1     | 0           | 3.18  | 283               | 315   | 358   | 31             |
| ab -n 100 -c 10    | 0           | 7.78  | 335               | 1226  | 2074  | 13             |
| ab -n 100 -c 100   | 0           | 7.92  | 296               | 6436  | 12340 | 13             |
| ab -n 1000 -c 1000 | 699 (70%)   | 26.38 | 515               | 26756 | 37725 | 38             |

## 2.2. До создания индекса, app\_pg\_max\_connections=50

Результаты представлены в виде таблицы.

| Тест               | Failed Reqs | RPS   | Request time (ms) |       |       | Total time (s) |
|--------------------|-------------|-------|-------------------|-------|-------|----------------|
|                    |             |       | Min               | Mean  | Max   |                |
| ab -n 100 -c 1     | 0           | 3.01  | 287               | 332   | 380   | 33             |
| ab -n 100 -c 10    | 0           | 8.35  | 297               | 1120  | 1999  | 12             |
| ab -n 100 -c 100   | 0           | 8.62  | 306               | 7758  | 11303 | 12             |
| ab -n 1000 -c 1000 | 612 (61%)   | 22.65 | 125               | 28016 | 43814 | 44             |

## 2.3. До создания индекса, app\_pg\_max\_connections=100

Результаты представлены в виде таблицы.

| Тест               | Failed Reqs | RPS   | Request time (ms) |       |       | Total time (s) |
|--------------------|-------------|-------|-------------------|-------|-------|----------------|
|                    |             |       | Min               | Mean  | Max   |                |
| ab -n 100 -c 1     | 0           | 3.02  | 291               | 331   | 388   | 33             |
| ab -n 100 -c 10    | 0           | 8.30  | 358               | 1117  | 2004  | 12             |
| ab -n 100 -c 100   | 0           | 8.57  | 347               | 9517  | 11336 | 12             |
| ab -n 1000 -c 1000 | 537 (54%)   | 18.82 | 109               | 30559 | 52742 | 53             |

## 2.4. До создания индекса, app\_pg\_max\_connections=500 (psql=100)

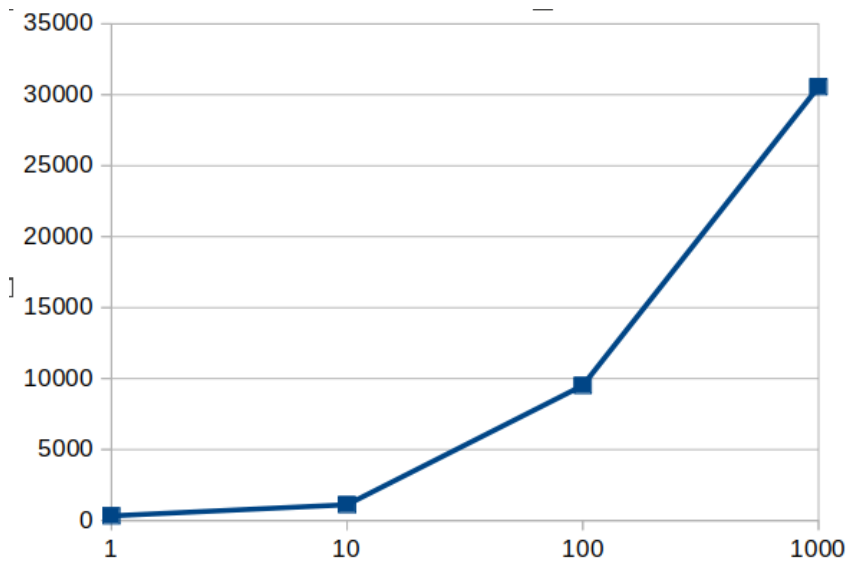
Особенность теста в том, что на стороне postgresql не повышалось число max коннекшнов (default: 100), повышалось только на стороне приложения.

Результаты представлены в виде таблицы.

| Тест               | Failed Reqs | RPS   | Request time (ms) |       |       | Total time (s) |
|--------------------|-------------|-------|-------------------|-------|-------|----------------|
|                    |             |       | Min               | Mean  | Max   |                |
| ab -n 100 -c 1     | 0           | 3.05  | 284               | 328   | 373   | 33             |
| ab -n 100 -c 10    | 0           | 8.33  | 308               | 1112  | 2062  | 12             |
| ab -n 100 -c 100   | 0           | 8.87  | 351               | 9243  | 10931 | 11             |
| ab -n 1000 -c 1000 | 896 (90%)   | 29.32 | 113               | 13427 | 33680 | 34             |

По итогу — max\_connections в приложении будет установлено равным настройкам psql, то есть 100. В дальнейших тестах будет использовано только значение 100.

## 2.5. Итоги



На графике представлен рост среднего времени ответа API от числа одновременных запросов для варианта `app_pg_max_connections=100`.

## 3. Создание индекса

Необходимость создания индекса определяется следующей диагностикой:

```
hlsocial=# explain analyze SELECT * FROM users
WHERE email LIKE '%@rp%'
OR first_name LIKE '%@rp%'
OR second_name LIKE '%@rp%'
OR city LIKE '%@rp%'
ORDER BY user_id;

                                QUERY PLAN
-----
Gather Merge  (cost=25595.66..26427.09 rows=7126 width=84) (actual time=220.498..227.434 rows=3200 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Sort  (cost=24595.64..24604.55 rows=3563 width=84) (actual time=217.368..218.651 rows=1067 loops=3)
        Sort Key: user_id
        Sort Method: quicksort  Memory: 198kB
        Worker 0:  Sort Method: quicksort  Memory: 154kB
        Worker 1:  Sort Method: quicksort  Memory: 181kB
        -> Parallel Seq Scan on users  (cost=0.00..24385.44 rows=3563 width=84) (actual time=4.269..215.932 rows=1867 loops=3)
              Filter: (((email)::text ~ '%@rp%':text) OR ((first_name)::text ~ '%@rp%':text) OR ((second_name)::text ~ '%@rp%':text) OR ((city)::text ~ '%@rp%':text))
              Rows Removed by Filter: 338911
Planning Time: 0.470 ms
Execution Time: 231.283 ms
(13 rows)
```

До создания индекса поиск происходит по алгоритму Parallel Seq Scan и занимает 231ms.

При создании индекса необходимо учесть все колонки, используемые в запросе.

### Попытка номер 1:

```
# CREATE EXTENSION btree_gin;
# CREATE INDEX users_gin_idx ON users USING GIN
  (first_name,second_name);
CREATE INDEX

# explain analyze SELECT * FROM users
  WHERE first_name LIKE '@rp%'
  OR second_name LIKE '@rp%';
...
-> Parallel Seq Scan on users  (cost=0.00..22260.58 rows=30 width=84)
(actual time=73.797..73.799 rows=0 loops=3)
      Filter: (((first_name)::text ~ '@rp%':text) OR ((second_name)::text ~ '@rp%':text))
```

```
Rows Removed by Filter: 339978
Planning Time: 0.354 ms
Execution Time: 83.702 ms
(8 rows)
```

Результат — индекс не работает. Нужно создавать иначе.

## Попытка номер 2:

```
# ALTER TABLE users ADD COLUMN first_name_ts tsvector;
ALTER TABLE
# ALTER TABLE users ADD COLUMN second_name_ts tsvector;
ALTER TABLE
# ALTER TABLE users ADD COLUMN email_ts tsvector;
ALTER TABLE
# ALTER TABLE users ADD COLUMN city_ts tsvector;
ALTER TABLE
# UPDATE users SET first_name_ts = to_tsvector(first_name);
UPDATE 1019933
# UPDATE users SET second_name_ts = to_tsvector(second_name);
UPDATE 1019933
# UPDATE users SET email_ts = to_tsvector(email);
UPDATE 1019933
# UPDATE users SET city_ts = to_tsvector(city);
UPDATE 1019933

# SELECT second_name FROM users WHERE first_name_ts @@ to_tsquery('ётр') LIMIT
1;
 second_name
-----
(0 rows)

# SELECT second_name FROM users WHERE first_name_ts @@ to_tsquery('пётр') LIMIT
1;
 second_name
-----
 ПИМЕНОВ
(1 row)
```

Результат — поиск по части слова не работает. Хотя ровно так индекс предлагали создавать на видео из обучения (на видео тоже не работало). **Вопрос к лектору — как же так???**

## Попытка номер 3:

```
# CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE EXTENSION
# CREATE INDEX IF NOT EXISTS users_gin_trgm_idx ON users USING gin (first_name gin_trgm_ops,
second_name gin_trgm_ops, email gin_trgm_ops, city gin_trgm_ops);
CREATE INDEX
```

```
# SELECT (first_name, second_name) FROM users
  WHERE first_name LIKE '%ётр%'
  OR second_name LIKE '%ётр%' LIMIT 1;
  row
-----
(Пётр,Пименов)
(1 row)

# explain analyze SELECT (first_name, second_name) FROM users
  WHERE first_name LIKE '%ётр%'
  OR second_name LIKE '%ётр%'
  OR city LIKE '%ётр%';

               QUERY PLAN
-----
Bitmap Heap Scan on users (cost=79.79..8321.35 rows=3368 width=32) (actual time=0.755..8.550 rows=3200 loops=1)
  Recheck Cond: (((first_name)::text ~~ '%ётр%':text) OR ((second_name)::text ~~ '%ётр%':text) OR ((city)::text ~~ '%ётр%':text))
  Heap Blocks: exact=2646
   -> BitmapOr (cost=79.79..79.79 rows=3368 width=0) (actual time=0.463..0.473 rows=0 loops=1)
     -> Bitmap Index Scan on users_gin_trgm_idx (cost=0.00..44.28 rows=3237 width=0) (actual time=0.438..0.440 rows=3200 loops=1)
        Index Cond: ((first_name)::text ~~ '%ётр%':text)
     -> Bitmap Index Scan on users_gin_trgm_idx (cost=0.00..16.48 rows=64 width=0) (actual time=0.009..0.011 rows=0 loops=1)
        Index Cond: ((second_name)::text ~~ '%ётр%':text)
     -> Bitmap Index Scan on users_gin_trgm_idx (cost=0.00..16.50 rows=67 width=0) (actual time=0.008..0.010 rows=0 loops=1)
        Index Cond: ((city)::text ~~ '%ётр%':text)
  Planning Time: 0.211 ms
  Execution Time: 12.857 ms
(12 rows)
```

И вот это работает, но только если не добавлять в поиск поле email.  
При добавлении email работать перестаёт:

```
# explain analyze SELECT (first_name, second_name) FROM users
  WHERE first_name LIKE '%ётр%'
  OR second_name LIKE '%ётр%'
  OR email LIKE '%ётр%'
  OR city LIKE '%ётр%';

               QUERY PLAN
-----
-----
Gather (cost=1000.00..26230.44 rows=8450 width=32) (actual time=1.180..257.006 rows=3200 loops=1)
  Workers Planned: 2
  Workers Launched: 2
   -> Parallel Seq Scan on users (cost=0.00..24385.44 rows=3521 width=32) (actual time=1.525..249.838 rows=1067 loops=3)
      Filter: (((first_name)::text ~~ '%ётр%':text) OR ((second_name)::text ~~ '%ётр%':text) OR ((email)::citext ~~ '%ётр%':citext) OR ((city)::text ~~ '%ётр%':text))
      Rows Removed by Filter: 338911
  Planning Time: 0.448 ms
  Execution Time: 261.288 ms
(8 rows)
```

**Вопрос: почему перестаёт работать при добавлении email?**

Результаты измерений по данной попытке представлены в виде таблицы.

НО: поиск по email исключен, т. к. он не работает в выбранном индексе.

| Тест               | Failed Reqs | RPS | Request time (ms) |      |       | Total time (s) |
|--------------------|-------------|-----|-------------------|------|-------|----------------|
|                    |             |     | Min               | Mean | Max   |                |
| ab -n 100 -c 1     | 0           | 93  | 42                | 92   | 115   | 9              |
| ab -n 100 -c 10    | 0           | 93  | 56                | 95   | 193   | 1              |
| ab -n 100 -c 100   | 0           | 112 | 105               | 421  | 788   | 1              |
| ab -n 1000 -c 1000 | 0           | 95  | 157               | 4901 | 10373 | 10             |

#### Попытка номер 4.

Была уточнена задача. Требуется использовать оператор AND при поиске. И поиск должен производиться только по имени и фамилии.

```
# explain analyze SELECT (first_name, second_name) FROM users
  WHERE second_name LIKE '%три%'
  AND first_name LIKE '%три%';
          QUERY PLAN
-----
Bitmap Heap Scan on users (cost=29.03..410.39 rows=101 width=32) (actual time=0.158..0.403 rows=98 loops=1)
  Recheck Cond: (((first_name)::text ~~ '%три% '::text) AND ((second_name)::text ~~ '%три% '::text))
  Rows Removed by Index Recheck: 26
  Heap Blocks: exact=44
-> Bitmap Index Scan on users_gin_trgm_idx (cost=0.00..29.01 rows=101 width=0) (actual time=0.141..0.143
rows=124 loops=1)
  Index Cond: (((first_name)::text ~~ '%три% '::text) AND ((second_name)::text ~~ '%три% '::text))
Planning Time: 0.222 ms
Execution Time: 0.605 ms
(8 rows)
```

В случае, если пересоздать индекс, чтобы первым аргументом был second\_name, как было рекомендовано преподавателем, тайминги по сути не меняются:

```
# CREATE INDEX IF NOT EXISTS users_gin_trgm_idx_s ON users USING gin (second_name gin_trgm_ops,
first_name gin_trgm_ops);
CREATE INDEX
# explain analyze SELECT (first_name, second_name) FROM users
  WHERE second_name LIKE '%три%'
  AND first_name LIKE '%три%' ORDER BY user_id;
          QUERY PLAN
-----
Sort (cost=405.75..406.00 rows=101 width=40) (actual time=0.573..0.725 rows=98 loops=1)
  Sort Key: user_id
  Sort Method: quicksort  Memory: 34kB
-> Bitmap Heap Scan on users (cost=21.03..402.39 rows=101 width=40) (actual time=0.149..0.405 rows=98
loops=1)
  Recheck Cond: (((second_name)::text ~~ '%три% '::text) AND ((first_name)::text ~~ '%три% '::text))
  Rows Removed by Index Recheck: 26
  Heap Blocks: exact=44
-> Bitmap Index Scan on users_gin_trgm_idx_s (cost=0.00..21.01 rows=101 width=0) (actual time=0.133..0.135
rows=124 loops=1)
  Index Cond: (((second_name)::text ~~ '%три% '::text) AND ((first_name)::text ~~ '%три% '::text))
Planning Time: 0.299 ms
Execution Time: 0.904 ms
(11 rows)
```

Результаты измерений по данной попытке представлены в виде таблицы.

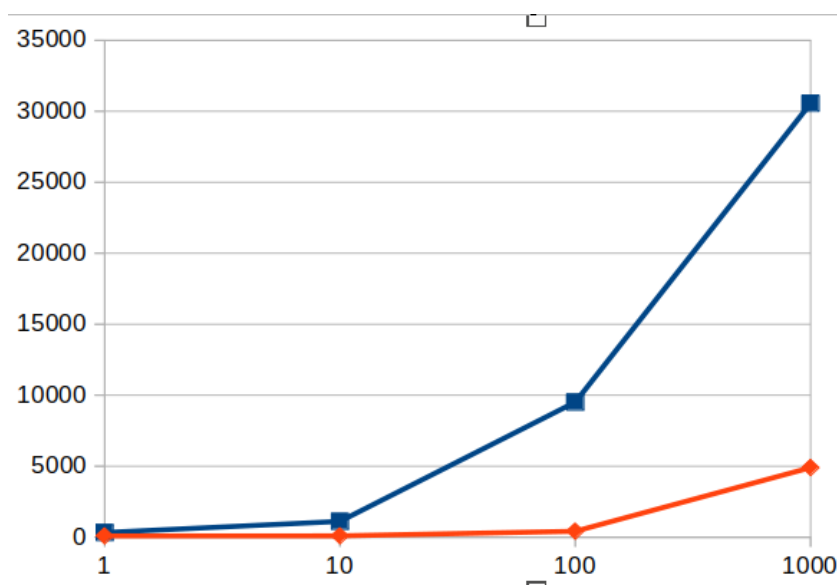
| Тест               | Failed Reqs | RPS | Request time (ms) |      |      | Total time (s) |
|--------------------|-------------|-----|-------------------|------|------|----------------|
|                    |             |     | Min               | Mean | Max  |                |
| ab -n 100 -c 1     | 0           | 15  | 39                | 65   | 95   | 6              |
| ab -n 100 -c 10    | 0           | 85  | 59                | 105  | 212  | 1.2            |
| ab -n 100 -c 100   | 0           | 63  | 92                | 920  | 1497 | 1.6            |
| ab -n 1000 -c 1000 | 0           | 110 | 190               | 4698 | 8942 | 9              |

Таким образом, результаты получились незначительно лучше предыдущего теста, где поиск производился более честно, с использованием операции OR, и по большему числу колонок.

## 4. График времени ответа после создания индекса

На графике представлено время ответа в зависимости от числа запросов в секунду.

Красным на графике отмечены новые средние значения времени выполнения одного запроса, синим — старые цифры без оптимизации:



Вывод: выбранный индекс действительно увеличивает производительность требуемых GET запросов.

## 5. Другие тесты

| Описание теста | Тест   | RPS   | Mean time |
|----------------|--|-------|-----------|
| Авторизация    | ab -n 100 -c 100 -T application/json -p test 'http://127.0.0.1:8080/v1/login/'   | 85.09 | 678       |
|                | ab -n 1000 -c 1000 -T application/json -p file 'http://127.0.0.1:8080/v1/login/' | 88.41 | 5212      |
| Self profile   | ab -n 100 -c 1 -C token="\$TOKEN" 'http://127.0.0.1:8080/v1/users/1'             | 691   | 1         |
|                | ab -n 100 -c 10 -C token="\$TOKEN" 'http://127.0.0.1:8080/v1/users/1'            | 3968  | 2         |
|                | ab -n 100 -c 100 -C token="\$TOKEN" 'http://127.0.0.1:8080/v1/users/1'           | 3487  | 16        |
|                | ab -n 1000 -c 1000 -C token="\$TOKEN" 'http://127.0.0.1:8080/v1/users/1'         | 1796  | 126       |
|                | ab -n 100000 -c 1000 -C token="\$TOKEN" 'http://127.0.0.1:8080/v1/users/1'       | 1747  | 463       |