Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

О Т Ч Е Т

по лабораторной работе № 2

Название: Оптимизация запросов. Основы EXPLAIN в PostgreSQL. Индексация

Дисциплина: Технология параллельных систем баз данных

Студент ИУ6-12М (Группа) _____ (Подпись, дата) С.В. Астахов (И.О. Фамилия)

Преподаватель _____ (Подпись, дата) _____ (И.О. Фамилия)

Москва, 2023

Введение

1. Цель работы: формирование следующей компетенции: студент должен получить навыки работы с командой EXPLAIN. Он также должен познакомиться с эффективными методами индексации в PostgreSQL.

Ход выполнения

2. Основы EXPLAIN.

Создадим таблицу, заполним ее миллионом записей, выполним команду EXPLAIN для запроса выборки всех записей, затем добавим записи в таблицу и выполним EXPLAIN повторно. Как видно из рисунка 1, при вставке новых записей в таблицу, ее статистика не обновляется автоматически, что приводит к повторяющимся результатам при вызове EXPLAIN.

```
trickster@Ubuntu1:~$ sudo -u admin psql iu6
[sudo] password for trickster:
could not change directory to "/home/trickster": Permission denied
psql (9.6.24)
Type "help" for help.

iu6=# CREATE TABLE foo (c1 integer, c2 text);
CREATE TABLE
iu6=# INSERT INTO foo
  SELECT i, md5(random()::text)
  FROM generate_series(1, 1000000) AS i;
INSERT 0 1000000
iu6=# EXPLAIN SELECT * FROM foo;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18918.18 rows=1058418 width=36)
(1 row)

iu6=# INSERT INTO foo
  SELECT i, md5(random()::text)
  FROM generate_series(1, 10) AS i;
INSERT 0 10
iu6=# EXPLAIN SELECT * FROM foo;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.00 rows=1000000 width=37)
(1 row)

iu6=# INSERT INTO foo
  SELECT i, md5(random()::text)
  FROM generate_series(1, 10) AS i;
INSERT 0 10
iu6=# EXPLAIN SELECT * FROM foo;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.00 rows=1000000 width=37)
(1 row)
```

Рисунок 1 — пример EXPLAIN без обновления статистики

Обновим статистику и убедимся, что результаты вызова EXPLAIN изменились, так как была собрана актуальная статистика (рисунок 2).

```
iu6=# ANALYZE foo;
ANALYZE
iu6=# EXPLAIN SELECT * FROM foo;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.20 rows=1000020 width=37)
(1 row)
```

Рисунок 2 — обновление статистики по таблице

Используем EXPLAIN ANALYZE, чтобы увидеть время, потраченное на реальное исполнение запроса и количество считанных строк.

```
iu6=# EXPLAIN ANALYZE SELECT * FROM foo;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.20 rows=1000020 width=37) (actual time=1.076..55.755 rows=1000020 loops=1)
 Planning time: 18.054 ms
 Execution time: 77.929 ms
(3 rows)
```

Рисунок 3 — пример выполнения EXPLAIN ANALYZE

Добавим дополнительное условие в запрос, увидим, что стоимость выполнения запроса изменилась, а количество выбранных строк уменьшилось (рисунок 4).

```
iu6=# EXPLAIN SELECT * FROM foo WHERE c1 > 500;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..20834.25 rows=999570 width=37)
   Filter: (c1 > 500)
(2 rows)
```

Рисунок 4 — исследование запроса с дополнительным условием

Создадим индекс для c1 и попробуем повторить запрос. Индекс использоваться не будет, так как выбирается большая доля записей из таблицы и использование индекса неэффективно (рисунки 5, 6).

```
iu6=# create index on foo (c1);
CREATE INDEX
```

Рисунок 5 — создание индекса

```
iu6=# EXPLAIN ANALYZE SELECT * FROM foo WHERE c1 > 500;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..20834.25 rows=999562 width=37) (actual time=0.160..75.926 rows=999500 loops=1)
   Filter: (c1 > 500)
   Rows Removed by Filter: 520
 Planning time: 4.802 ms
 Execution time: 97.368 ms
(5 rows)
```

Рисунок 6 — запрос после создания индекса

Изменим запрос так, чтобы выбиралось малое число записей. Теперь индекс будет использоваться (рисунок 7).

```
iu6=# EXPLAIN ANALYZE SELECT * FROM foo WHERE c1 < 500;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Index Scan using foo_c1_idx on foo  (cost=0.42..24.15 rows=458 width=37) (actual time=0.015..0.139 rows=519 loops=1)
   Index Cond: (c1 < 500)
 Planning time: 0.084 ms
 Execution time: 0.171 ms
(4 rows)
```

Рисунок 7 — запрос с использованием индекса

Усложним запрос, добавив условие по текстовому полю. Сначала будет проходить фильтрация, а затем — сканирование по индексу (рисунок 8).

```
iu6=# EXPLAIN SELECT * FROM foo WHERE c1 < 500 AND c2 LIKE 'abcd%';
                                    QUERY PLAN
-----------------------------------------------------------------------------
 Index Scan using foo_c1_idx on foo  (cost=0.42..25.30 rows=1 width=37)
   Index Cond: (c1 < 500)
   Filter: (c2 ~~ 'abcd%'::text)
(3 rows)
```

Рисунок 8 — запрос с фильтрацией по текстовому полю

Уберем условие по индексированному полю, теперь будет происходить полное сканирование таблицы, так как по полю c2 индекс не создан (рисунок 9).

```
iu6=# EXPLAIN ANALYZE SELECT * FROM foo WHERE c2 LIKE 'abcd%';
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..20834.25 rows=100 width=37) (actual time=11.103..70.313 rows=15 loops=1)
   Filter: (c2 ~~ 'abcd%'::text)
   Rows Removed by Filter: 1000005
 Planning time: 0.151 ms
 Execution time: 70.348 ms
(5 rows)
```

Рисунок 9 — запрос без условия по полю c1

Создадим индекс по полю c2, теперь он будет использоваться в запросе (рисунок 10).

```
iu6=# create index index_c2 ON foo (c2 text_pattern_ops);
CREATE INDEX
iu6=# EXPLAIN ANALYZE SELECT * FROM foo WHERE c2 LIKE 'abcd%';
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using index_c2 on foo  (cost=0.42..8.45 rows=100 width=37) (actual time=0.052..0.069 rows=15 loops=1)
   Index Cond: ((c2 ~>=~ 'abcd'::text) AND (c2 ~<~ 'abce'::text))
   Filter: (c2 ~~ 'abcd%'::text)
 Planning time: 8.961 ms
 Execution time: 0.082 ms
(5 rows)
```

Рисунок 10 — запрос по индексированному текстовому полю

Удалим индекс по полю c1 и выполним запрос с упорядочиванием, будет использован метод сортировки external merge.

```
iu6=# EXPLAIN ANALYZE SELECT * FROM foo ORDER BY c1;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=172685.18..175185.23 rows=1000020 width=37) (actual time=474.441..566.653 rows=1000020 loops=1)
   Sort Key: c1
   Sort Method: external merge  Disk: 45952kB
   ->  Seq Scan on foo  (cost=0.00..18334.20 rows=1000020 width=37) (actual time=0.007..89.545 rows=1000020 loops=1)
 Planning time: 0.246 ms
 Execution time: 595.282 ms
(6 rows)
```

Рисунок 11 — запрос с упорядочиванием

Увеличим объем используемой оперативной памяти — теперь будет использована сортировка quicksort (рисунок 12).

```
iu6=# SET work_mem TO '200MB';
SET
iu6=# EXPLAIN ANALYZE SELECT * FROM foo ORDER BY c1;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=117994.18..120494.23 rows=1000020 width=37) (actual time=343.783..393.886 rows=1000020 loops=1)
   Sort Key: c1
   Sort Method: quicksort  Memory: 102703kB
   ->  Seq Scan on foo  (cost=0.00..18334.20 rows=1000020 width=37) (actual time=0.008..70.568 rows=1000020 loops=1)
 Planning time: 0.068 ms
 Execution time: 426.486 ms
(6 rows)
```

Рисунок 12 — запрос с упорядочиванием после изменения настроек

Создадим индекс для поля c1 и повторим запрос. Так как в индексе значения упорядочены, сортировка не понадобится (рисунок 13).

```
iu6=# create index on foo (c1);
CREATE INDEX
iu6=# EXPLAIN ANALYZE SELECT * FROM foo ORDER BY c1;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using foo_c1_idx on foo  (cost=0.42..34327.72 rows=1000020 width=37) (actual time=0.167..141.383 rows=1000020 loops=1)
 Planning time: 0.123 ms
 Execution time: 168.310 ms
(3 rows)
```

Рисунок 13 — запрос с упорядочиванием по индексу

Создадим новую таблицу и соберем для нее статистику (рисунок 14).

```
iu6=# CREATE TABLE bar (c1 integer, c2 boolean);
INSERT INTO bar
  SELECT i, i%2=1
  FROM generate_series(1, 500000) AS i;
CREATE TABLE
INSERT 0 500000
iu6=# ANALYZE bar;
ANALYZE
```

Рисунок 14 — создание новой таблицы

Выполним соединение новой таблицы со старой, затем создадим для нее индекс и выполним соединение повторно. Тип соединения изменится из-за наличия индексов (рисунок 15).

```
iu6=# EXPLAIN ANALYZE SELECT * FROM foo JOIN bar ON foo.c1=bar.c1;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=13463.00..40547.28 rows=500000 width=42) (actual time=121.824..632.131 rows=500020 loops=1)
   Hash Cond: (foo.c1 = bar.c1)
   ->  Seq Scan on foo  (cost=0.00..18334.20 rows=1000020 width=37) (actual time=0.014..69.962 rows=1000020 loops=1)
   ->  Hash  (cost=7213.00..7213.00 rows=500000 width=5) (actual time=121.087..121.087 rows=500000 loops=1)
         Buckets: 524288  Batches: 1  Memory Usage: 22163kB
         ->  Seq Scan on bar  (cost=0.00..7213.00 rows=500000 width=5) (actual time=0.016..31.381 rows=500000 loops=1)
 Planning time: 1.856 ms
 Execution time: 661.091 ms
(8 rows)

iu6=# create index ON bar (c1);
CREATE INDEX
iu6=# EXPLAIN ANALYZE SELECT * FROM foo JOIN bar ON foo.c1=bar.c1;
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Merge Join  (cost=1.14..39889.82 rows=500000 width=42) (actual time=0.011..361.507 rows=500020 loops=1)
   Merge Cond: (foo.c1 = bar.c1)
   ->  Index Scan using foo_c1_idx on foo  (cost=0.42..34327.72 rows=1000020 width=37) (actual time=0.004..88.105 rows=500021 loops=1)
   ->  Index Scan using bar_c1_idx on bar  (cost=0.42..15212.42 rows=500000 width=5) (actual time=0.003..118.124 rows=500020 loops=1)
 Planning time: 0.280 ms
 Execution time: 381.128 ms
(6 rows)
```

Рисунок 15 — соединение таблиц

Очистим кэш PostgreSQL и 3 раза обратимся к таблице foo. При каждом запросе ее часть будет кэшироваться для ускорения последующих обращений (рисунок 16).

```
iu6=# \q
trickster@Ubuntu1:~$ sudo service postgresql stop
[sudo] password for trickster:
trickster@Ubuntu1:~$ sudo sync
trickster@Ubuntu1:~$ sudo su root
root@Ubuntu1:/home/trickster# echo 3 > /proc/sys/vm/drop_caches
root@Ubuntu1:/home/trickster# exit
exit
trickster@Ubuntu1:~$ sudo service postgresql start
trickster@Ubuntu1:~$ sudo -u admin psql iu6
could not change directory to "/home/trickster": Permission denied
psql (9.6.24)
Type "help" for help.

iu6=# EXPLAIN (ANALYZE, BUFFERS) select * from foo;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.20 rows=1000020 width=37) (actual time=0.678..609.805 rows=1000020 loops=1)
   Buffers: shared read=8334
 Planning time: 10.636 ms
 Execution time: 646.710 ms
(4 rows)

iu6=# EXPLAIN (ANALYZE, BUFFERS) select * from foo;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.20 rows=1000020 width=37) (actual time=0.032..73.622 rows=1000020 loops=1)
   Buffers: shared hit=32 read=8302
 Planning time: 0.036 ms
 Execution time: 114.004 ms
(4 rows)

iu6=# EXPLAIN (ANALYZE, BUFFERS) select * from foo;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.20 rows=1000020 width=37) (actual time=0.051..69.548 rows=1000020 loops=1)
   Buffers: shared hit=64 read=8270
 Planning time: 0.079 ms
 Execution time: 105.050 ms
(4 rows)
```

Рисунок 16 — работа кэша

4. Эффективные методы индексации в PostgreSQL

Переподключимся к БД от имени пользователя postgres, создадим новую таблицу и индексы в ней (рисунок 17).

```
iu6=# \q
trickster@Ubuntu1:~$ sudo -u postgres psql postgres
could not change directory to "/home/trickster": Permission denied
psql (9.6.24)
Type "help" for help.

postgres=# create table users
(
    id         serial    not null
        constraint users_pk
            primary key,
    email      text      not null,
    type       text      not null,
    extra      jsonb     not null,
    created_at timestamp not null,
    updated_at timestamp not null
);
CREATE TABLE
postgres=# alter table users
    owner to postgres;

create unique index users_email_uindex
    on users (email);

create unique index users_id_uindex
    on users (id);
ALTER TABLE
CREATE INDEX
CREATE INDEX
```

Рисунок 17 — создание новой БД

Заполним новую БД (рисунок 18).

```
postgres=# insert into users (email, created[redacted]t, updated[redacted]t, type, extra)
select 'user[redacted]' || seq || '@' || (
    CASE (RANDOM() * 2)::INT
        WHEN 0 THEN 'yandex'
        WHEN 1 THEN 'main'
        WHEN 2 THEN 'custom'
        END
    ) || '.ru' AS email,
       timestamp '2014-01-10 20:00:00' +
       random() * (timestamp '2016-01-10 20:00:00' -
                   timestamp '2014-01-10 10:00:00'),
       timestamp '2016-01-10 20:00:00' +
       random() * (timestamp '2018-01-20 20:00:00' -
                   timestamp '2016-01-10 10:00:00'),
       (
           CASE (RANDOM() * 9)::INT
               WHEN 0 THEN 'admin'
               ELSE 'user'
               END
           )    as type,
       (
           CASE (RANDOM() * 9)::INT
        WHEN 0 THEN '{"extra d[redacted]ata": true}'::jsonb
               ELSE '{}'
        END
    )
from generate[redacted]eries(1, 1000000) seq;
INSERT 0 1000000
```

Рисунок 18 — заполнение таблицы

Фрагмент заполненной таблицы представлен на рисунке 19.

```
postgres=# select * from users limit 10;
 id |       email       | type  | extra |         created[redacted]t         |         updated[redacted]t
----+-------------------+-------+-------+------------------------------------+------------------------------------
  1 | user[redacted]1@main.ru    | user  | {}    | 2014-03-20 15:01:08.54357  | 2016-06-29 07:09:04.50811
  2 | user[redacted]2@main.ru    | user  | {}    | 2014-06-23 02:04:01.943411 | 2016-07-12 07:22:32.417706
  3 | user[redacted]3@main.ru    | user  | {}    | 2015-09-25 20:44:32.299668 | 2016-12-02 14:40:39.278823
  4 | user[redacted]4@custom.ru  | admin | {}    | 2015-04-30 16:56:33.481312 | 2016-07-31 07:21:33.25988
  5 | user[redacted]5@yandex.ru  | user  | {}    | 2015-08-13 02:31:55.316984 | 2016-11-05 19:45:20.07277
  6 | user[redacted]6@main.ru    | user  | {}    | 2015-06-03 16:11:54.802037 | 2016-02-18 03:08:51.526172
  7 | user[redacted]7@main.ru    | user  | {}    | 2015-10-04 17:11:39.888311 | 2017-10-23 02:56:59.11233
  8 | user[redacted]8@custom.ru  | admin | {}    | 2014-07-29 02:28:48.641673 | 2017-09-07 17:25:57.696644
  9 | user[redacted]9@main.ru    | user  | {}    | 2014-11-29 05:53:07.110662 | 2017-06-19 08:52:07.530917
 10 | user[redacted]10@custom.ru | user  | {}    | 2015-06-01 06:51:19.15997  | 2017-11-16 22:20:34.150594
(10 rows)
```

Рисунок 19 — фрагмент заполненной таблицы

Сравним выполнение указанных в задании запросов без использования индексов и при их наличии. Очевидно, при использовании индексов по полям из условий (where) не проводилось полного сканирования, а использовались индексы, что значительно ускоряло выполнение запросов (рисунки 20-22).

```
postgres=# EXPLAIN SELECT COUNT(*) FROM users WHERE email ILIKE '%yandex.ru';
                            QUERY PLAN
-------------------------------------------------------------------
 Aggregate  (cost=23522.81..23522.82 rows=1 width=8)
   ->  Seq Scan on users  (cost=0.00..22942.00 rows=232323 width=0)
         Filter: (email ~~* '%yandex.ru'::text)
(3 rows)

postgres=# create index trgm_idx_users_email ON users USING gin (email gin_trgm_ops);
ERROR:  operator class "gin_trgm_ops" does not exist for access method "gin"
postgres=# create extension pg_trgm;
CREATE EXTENSION
postgres=# create index trgm_idx_users_email ON users USING gin (email gin_trgm_ops);
CREATE INDEX
postgres=# EXPLAIN SELECT COUNT(*) FROM users WHERE email ILIKE '%yandex.ru';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=16135.35..16135.36 rows=1 width=8)
   ->  Bitmap Heap Scan on users  (cost=2208.50..15554.54 rows=232323 width=0)
         Recheck Cond: (email ~~* '%yandex.ru'::text)
         ->  Bitmap Index Scan on trgm_idx_users_email  (cost=0.00..2150.42 rows=232323 width=0)
               Index Cond: (email ~~* '%yandex.ru'::text)
(5 rows)
```

Рисунок 20 — использование индекса по полю email

```
postgres=# EXPLAIN select * from users where updated_at = '2016-03-22 01:51:57.261738'::timestamp and type = 'admin';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on users  (cost=0.00..25442.00 rows=1 width=50)
   Filter: ((updated_at = '2016-03-22 01:51:57.261738'::timestamp without time zone) AND (type = 'admin'::text))
(2 rows)

postgres=# create index users_type_updated ON users (type, updated_at) WHERE type='admin'::text;
CREATE INDEX
postgres=# EXPLAIN select * from users where updated_at = '2016-03-22 01:51:57.261738'::timestamp and type = 'admin';
                                    QUERY PLAN
-----------------------------------------------------------------------------------
 Index Scan using users_type_updated on users  (cost=0.41..1251.17 rows=1 width=50)
   Index Cond: (updated_at = '2016-03-22 01:51:57.261738'::timestamp without time zone)
(2 rows)
```

Рисунок 21 — использование частичного индекса

```
postgres=# EXPLAIN select count(*) from users where extra @> '{"extra_data": true}'::jsonb;
                            QUERY PLAN
-------------------------------------------------------------------
 Aggregate  (cost=22944.50..22944.51 rows=1 width=8)
   ->  Seq Scan on users  (cost=0.00..22942.00 rows=1000 width=0)
         Filter: (extra @> '{"extra_data": true}'::jsonb)
(3 rows)

postgres=# create index json_idx ON users USING GIN (extra);
CREATE INDEX
postgres=# EXPLAIN select count(*) from users where extra @> '{"extra_data": true}'::jsonb;
                                    QUERY PLAN
-----------------------------------------------------------------------------------
 Aggregate  (cost=3372.32..3372.33 rows=1 width=8)
   ->  Bitmap Heap Scan on users  (cost=403.75..3369.82 rows=1000 width=0)
         Recheck Cond: (extra @> '{"extra_data": true}'::jsonb)
         ->  Bitmap Index Scan on json_idx  (cost=0.00..403.50 rows=1000 width=0)
               Index Cond: (extra @> '{"extra_data": true}'::jsonb)
(5 rows)
```

Рисунок 22 — использование индекса по json-полю

Пример записей, которые были подсчитаны при фильтрации по полю email приведен на рисунке 23.

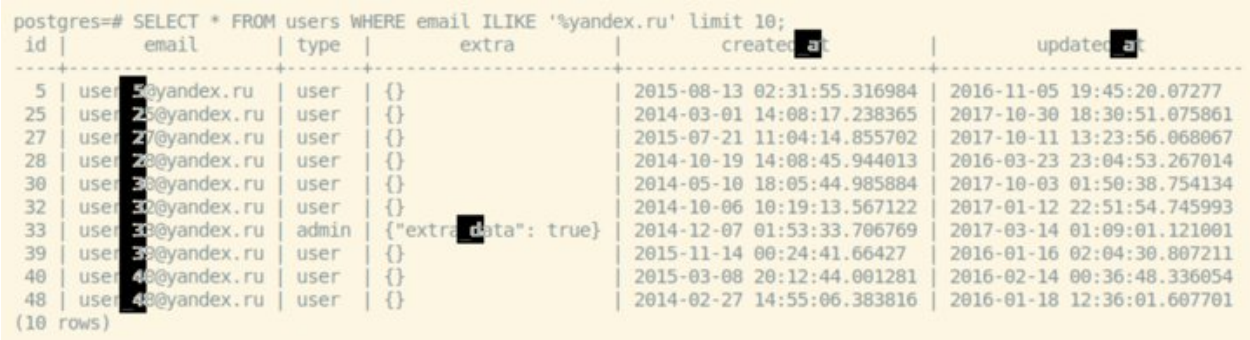

Рисунок 23 — пример отфильтрованных по email записей

Пример записей, которые были подсчитаны при фильтрации по полю extra приведен на рисунке 24.

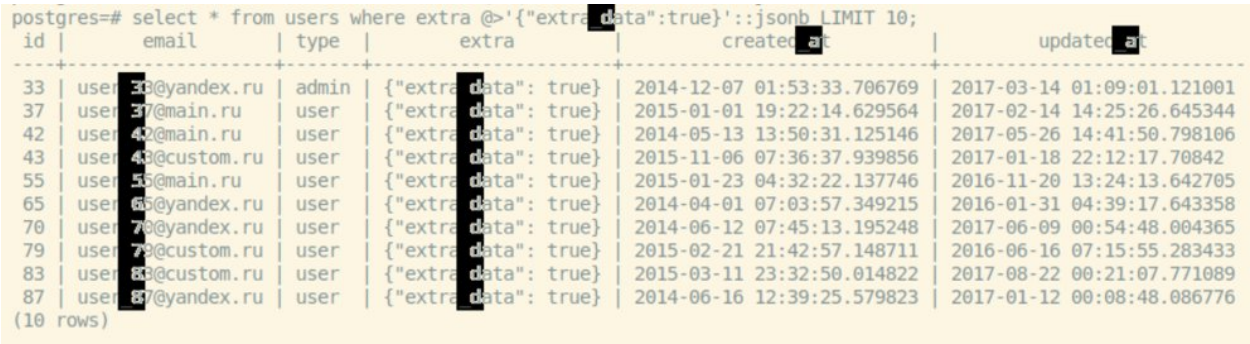

Рисунок 24 — пример отфильтрованных по extra записей

Вывод: в ходе лабораторной работы были получены навыки работы с командой EXPLAIN. Было произведено знакомство с эффективными методами индексации в PostgreSQL. Все задание были успешно выполнены, а их результаты соответствуют требованиям.