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 _____ Д.С. Каткова
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ А.Д. Пономарев
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

1. Цель лабораторной работы

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

2. Подключение к виртуальной машине и к базе данных

Выполним подключение к базе данных.

```
daria@ubuntu-01:~$ sudo -u admin psql iu6
[sudo] password for daria:
```

3. Основы EXPLAIN

В соответствии с заданием, создадим тестовую таблицу и включим в нее 1 млн. записей.

```
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
```

Прочитаем данные с помощью EXPLAIN.

```
                          QUERY PLAN
--------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.00 rows=1000000 width=37)
(1 row)
```

Добавим по аналогии 10 строк и повторим команду EXPLAIN.

```
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)
```

Можно заметить, что вывод не изменился. Это произошло потому, что хоть записи и были добавлены в таблицу, ее статистика обновлена не была. Чтобы обновить статистику по таблице, используем команду ANALYZE и повторим команду EXPLAIN. Теперь статистика обновлена.

```
iu6=# ANALYZE foo;
ANALYZE
iu6=# EXPLAIN SELECT * FROM foo;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.10 rows=1000010 width=37)
(1 row)
```

Далее используем параметры EXPLAIN и ANALYZE вместе.

```
                                                    QUERY PLAN

-------------------------------------------------------------------------------------------
-------------------------
 Seq Scan on foo  (cost=0.00..18334.10 rows=1000010 width=37) (actual time=0.053..75.
590 rows=1000010 loops=1)
 Planning time: 0.083 ms
 Execution time: 110.288 ms
(3 rows)
```

Добавим в запрос новое условие.

```
iu6=# EXPLAIN SELECT * FROM foo WHERE c1 > 500;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..20834.12 rows=999546 width=37)
   Filter: (c1 > 500)
(2 rows)
```

Как можно увидеть, индекс в данном запросе не используется, о чем говорит надпись Seq Scan. Теперь создадим индекс и повторим команды EXPLAIN и EXPLAIN ANALYZE.

```
iu6=# create index on foo (c1);
CREATE INDEX
iu6=# EXPLAIN SELECT * FROM foo WHERE c1 > 500;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..20834.12 rows=999526 width=37)
   Filter: (c1 > 500)
(2 rows)

iu6=# EXPLAIN ANALYZE  SELECT * FROM foo WHERE c1 > 500;
                                                    QUERY PLAN

-------------------------------------------------------------------------------------------
---------------------
 Seq Scan on foo  (cost=0.00..20834.12 rows=999526 width=37) (actual time=0.181..86.188
rows=999500 loops=1)
   Filter: (c1 > 500)
   Rows Removed by Filter: 510
 Planning time: 0.210 ms
 Execution time: 119.763 ms
(5 rows)
```

Хоть мы и создали индекс, он не использовался. Это произошло потому, что в данном случае для запроса выгоднее использовать последовательное сканирование. Индекс же будет использоваться только в том случае, если процент записей, удовлетворяющих условию, не превышает 5-10% от всех строк в таблице.

Немного изменим условие и снова произведем запрос.

```
iu6=# EXPLAIN SELECT * FROM foo WHERE c1 < 500;
                                QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using foo_c1_idx on foo  (cost=0.42..24.89 rows=484 width=37)
   Index Cond: (c1 < 500)
(2 rows)
```

Уже в этом случае, как можно увидеть, индекс используется. Это произошло по причине небольшого количества строк, удовлетворяющих условию (543 строк из 10000010, что составляет 0,005% от всех строк).

Дополнительно усложним условие. Используем еще одно текстовое поле и выполним команду EXPLAIN.

```
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..26.11 rows=1 width=37)
   Index Cond: (c1 < 500)
   Filter: (c2 ~~ 'abcd%'::text)
(3 rows)
```

Как можно увидеть, в данном запросе все еще используется индекс. Уберем условие с проверкой численного поля и оставим только проверку текстового поля.

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..20834.12 rows=100 width=37) (actual time=3.671..96.318 rows=25 loops=1)
   Filter: (c2 ~~ 'abcd%'::text)
   Rows Removed by Filter: 999985
 Planning time: 0.204 ms
 Execution time: 96.466 ms
(5 rows)
```

Как можно увидеть, здесь был произведен последовательный поиск. Это произошло по той причине, что никакого индекса на текстовой поле в нашей БД создано не было. Теперь создадим его и вновь проверим работу запроса.

```
iu6=# EXPLAIN SELECT * FROM foo WHERE c2 LIKE 'abcd%'
iu6-# ;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Index Scan using foo_c2_idx on foo  (cost=0.42..8.45 rows=100 width=37)
   Index Cond: ((c2 ~>=~ 'abcd'::text) AND (c2 ~<~ 'abce'::text))
   Filter: (c2 ~~ 'abcd%'::text)
(3 rows)
```

Как можно увидеть, после создания индекса он начал использоваться.

Удалим индекс, созданный для атрибута c1 таблицы foo. Выполним новую команду.

```
iu6=# drop index foo_c1_idx;
DROP INDEX
```

```
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=172684.01..175184.04 rows=1000010 width=37) (actual time=308.240..366.653 rows=1000010 loops=1)
   Sort Key: c1
   Sort Method: external merge  Disk: 45952kB
   ->  Seq Scan on foo  (cost=0.00..18334.10 rows=1000010 width=37) (actual time=0.027..54.531 rows=1000010 loops=1)
 Planning time: 0.225 ms
 Execution time: 392.284 ms
(6 rows)
```

Сейчас методом сортировки используется external merge. Увеличим объем используемой памяти и повторим данный запрос.

```
iu6=# SET work_mem TO '200MB';
SET
iu6=# EXPLAIN ANALYZE SELECT * FROM foo ORDER BY c1;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=117993.01..120493.04 rows=1000010 width=37) (actual time=222.998..258.961 rows=1000010 loops=1)
   Sort Key: c1
   Sort Method: quicksort  Memory: 102702kB
   ->  Seq Scan on foo  (cost=0.00..18334.10 rows=1000010 width=37) (actual time=0.022..60.684 rows=1000010 loops=1)
 Planning time: 0.124 ms
 Execution time: 283.802 ms
(6 rows)
```

Как можно увидеть, после увеличения количества используемого объема памяти метод сортировки поменялся с external merge на quicksort. Это произошло по той причине, что хоть quicksort и является самой быстрой, она расходует достаточно много памяти. Из-за этого, раньше использовать ее было бы нерационально.

Вновь создадим индекс для c1 таблицы foo и повторим предыдущий запрос.

```
                                                              QUERY PLAN

---------------------------------------------------------------------------------------------------------------------------------------
----------------------
 Index Scan using foo_c1_idx on foo  (cost=0.42..34317.58 rows=1000010 width=37) (actual time=0.056..101.753
 rows=1000010 loops=1)
 Planning time: 0.397 ms
 Execution time: 133.834 ms
(3 rows)
```

Как можно увидеть, теперь и для сортировки записей в таблице используется индекс.

Создадим новую таблицу, соберем с нее статистику и выполним новую команду.

```
                                              QUERY PLAN
----------------------------------------------------------------------------------------------------------
 Hash Join  (cost=13463.00..40547.14 rows=500000 width=42) (actual time=95.204..329.677 rows=500010 loops=1)
   Hash Cond: (foo.c1 = bar.c1)
   ->  Seq Scan on foo  (cost=0.00..18334.10 rows=1000010 width=37) (actual time=0.008..41.178 rows=1000010 loops=1)
   ->  Hash  (cost=7213.00..7213.00 rows=500000 width=5) (actual time=95.023..95.023 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=0.010..34.988 rows=500000 loops=1)
 Planning time: 0.442 ms
 Execution time: 341.415 ms
(8 rows)
```

Как можно увидеть, здесь планировщик выбирает соединение по хешу, при котором строки одной таблицы записываются в хеш-таблицу в памяти, после чего сканируется другая таблица и для каждой ее строки проверяется соответствие по хеш-таблице.

Далее создадим индекс для атрибута c1 таблицы bar и проверим изменения.

```
                                              QUERY PLAN
----------------------------------------------------------------------------------------------------------
 Merge Join  (cost=1.66..39669.76 rows=500000 width=42) (actual time=0.038..198.743 rows=500010 loops=1)
   Merge Cond: (foo.c1 = bar.c1)
   ->  Index Scan using foo_c1_idx on foo  (cost=0.42..34327.57 rows=1000010 width=37) (actual time=0.009..58.638 rows=500011 loops=1)
   ->  Index Scan using bar_c1_idx on bar  (cost=0.42..15212.42 rows=500000 width=5) (actual time=0.008..47.492 rows=500010 loops=1)
 Planning time: 0.365 ms
 Execution time: 212.743 ms
(6 rows)
```

После создания индекса время выполнения запроса уменьшилось.

Далее удалим индекс, созданный для c1 таблицы foo, выйдем из консоли, остановим PostgreSQL, принудительно зафиксируем изменения в файловой системе, очистим кеш и заново запустим PostgreSQL. Войдем в консоль и сделаем запрос из задания три раза.

```
iu6=# EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM foo;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.10 rows=1000010 width=37) (actual time=0.114..75.757 rows=1000010 loops=1)
   Buffers: shared hit=96 read=8238
 Planning time: 0.093 ms
 Execution time: 118.087 ms
(4 rows)

iu6=# EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM foo;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.10 rows=1000010 width=37) (actual time=0.110..73.839 rows=1000010 loops=1)
   Buffers: shared hit=128 read=8206
 Planning time: 0.074 ms
 Execution time: 116.534 ms
(4 rows)

iu6=# EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM foo;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on foo  (cost=0.00..18334.10 rows=1000010 width=37) (actual time=0.096..72.921 rows=1000010 loops=1)
   Buffers: shared hit=160 read=8174
 Planning time: 0.062 ms
 Execution time: 115.518 ms
(4 rows)
```

Как можно увидеть, с каждым выполнением запроса количество блоков, считанных из кэша PostgreSQL, становится все меньше.

4. Эффективные методы индексации в PostgreSQL

Выйдем из консоли PostgreSQL и подключимся к базе postgres как пользователь postgres, после чего создадим базу данных.

```
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;
ALTER TABLE
postgres=# create unique index users_email_uindex
    on users (email);
CREATE INDEX
postgres=# create unique index users_id_uindex
    on users (id);
CREATE INDEX
```

Теперь заполним базу данных.

```
postgres=# insert into users (email, created_at, updated_at, type, extra)
select 'user_' || 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_data": true}'::jsonb
               ELSE '{}'
        END
    )
from generate_series(1, 1000000) seq;
INSERT 0 1000000
```

Итоговая таблица выглядит следующим образом.

id	email	type	extra	created_at	updated_at
1	user_1@main.ru	user	{}	2014-06-07 15:15:55.552945	2017-10-02 22:19:23.178154
2	user_2@main.ru	user	{}	2015-01-08 13:46:29.724995	2017-07-16 10:58:45.453768
3	user_3@custom.ru	user	{}	2014-04-18 00:18:11.699125	2017-01-11 12:49:47.708015
4	user_4@yandex.ru	user	{}	2014-05-10 04:37:51.119542	2017-07-29 20:47:10.066259
5	user_5@custom.ru	user	{}	2014-11-02 00:15:43.642032	2016-03-12 06:48:49.899692
6	user_6@yandex.ru	user	{}	2014-04-25 20:37:31.044703	2016-12-30 20:28:26.208894
7	user_7@yandex.ru	user	{}	2015-05-18 23:11:26.422566	2017-04-10 07:15:41.42074
8	user_8@yandex.ru	user	{}	2014-11-02 06:42:58.200946	2016-11-09 05:02:31.631763
9	user_9@custom.ru	user	{}	2015-07-01 12:03:32.289652	2016-07-18 15:30:13.371268
10	user_10@main.ru	user	{}	2015-01-30 17:35:32.137624	2017-06-26 23:57:52.72475
11	user_11@custom.ru	user	{}	2014-05-18 16:37:16.839718	2016-06-02 05:36:51.742382
12	user_12@main.ru	user	{}	2014-04-24 06:12:37.66243	2016-03-23 17:14:55.583914
13	user_13@main.ru	user	{}	2015-12-23 11:55:21.162211	2017-07-21 12:07:42.553459
14	user_14@main.ru	user	{}	2014-05-05 15:20:02.100617	2016-09-13 02:14:35.715873
15	user_15@main.ru	user	{}	2015-11-29 20:16:43.558534	2017-05-21 11:35:01.149762

Задание 1. Создать индекс для поиска по контексту значения поля email.

При выполнении будем использовать модуль pg_trgm и оператор индекса gin. Выполнение самого SELECT выглядит следующим образом.

```
postgres=# SELECT COUNT(*) FROM users WHERE email ILIKE '%yandex.ru';
 count
--------
 250133
(1 row)
```

Выполнение SELECT

Выполним следующую команду до создания индекса.

EXPLAIN SELECT COUNT(*) FROM users WHERE email ILIKE '%yandex.ru';

```
postgres=# EXPLAIN SELECT COUNT(*) FROM users WHERE email ILIKE '%yandex.ru';
                              QUERY PLAN
-----------------------------------------------------------------------
 Aggregate  (cost=23496.56..23496.56 rows=1 width=8)
   ->  Seq Scan on users  (cost=0.00..22941.00 rows=222222 width=0)
         Filter: (email ~~* '%yandex.ru'::text)
(3 rows)
```

Создадим индекс и выполним предыдущую команду еще раз.

```
postgres=# CREATE EXTENSION pg_trgm;
CREATE EXTENSION
postgres=# CREATE INDEX trgm_idx_users_email ON users USING gin (email gin_trgm_ops);
CREATE INDEX
```

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Aggregate  (cost=15888.55..15888.56 rows=1 width=8)
   ->  Bitmap Heap Scan on users  (cost=2114.22..15333.00 rows=222222 width=0)
         Recheck Cond: (email ~~* '%yandex.ru'::text)
         ->  Bitmap Index Scan on trgm_idx_users_email  (cost=0.00..2058.67 rows=222222 width=0)
               Index Cond: (email ~~* '%yandex.ru'::text)
(5 rows)
```

В отличие от первого раза, теперь планировщиком используется индекс.

Задание 2. Создать индекс по update_at пользователей, но только для админов (type = ‘admin’)

При выполнении будем использовать частичные индексы. Выполнение самого SELECT выглядит следующим образом.

```
 id |       email       | type  | extra |         created_at         |         updated_at
----+-------------------+-------+-------+----------------------------+----------------------------
 36 | user_36@yandex.ru | admin | {}    | 2014-11-27 17:00:08.601445 | 2017-08-07 00:18:33.002939
(1 row)
```

Была найдена одна строка соответствующая условию type = ‘admin’ и update_at = '2017-08-07 00:18:33.002939'

Выполним следующую команду до создания индекса.

EXPLAIN SELECT * FROM users WHERE updated_at = '2017-08-07 00:18:33.002939'::timestamp and type = 'admin';

```
                                                QUERY PLAN

-----------------------------------------------------------------------------------------------
-----------------------
 Seq Scan on users  (cost=0.00..25441.00 rows=1 width=50)
   Filter: ((updated_at = '2016-03-22 01:51:57.261738'::timestamp without time zone) AND
(type = 'admin'::text))
(2 rows)
```

Создадим индекс и выполним предыдущую команду еще раз.

CREATE INDEX idx_users_type_updated ON users (type, updated_at) WHERE updated_at = '2016-03-22 01:51:57.261738'::timestamp AND type='admin';

```
postgres=# CREATE INDEX idx_users_type_updated ON users (type, updated_at) WHERE updat
ed_at = '2016-03-22 01:51:57.261738'::timestamp AND type='admin';
CREATE INDEX
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..25441.00 rows=1 width=50)
   Filter: ((updated_at = '2017-08-07 00:18:33.002939'::timestamp without time zone) AND (type = 'admin'::text))
(2 rows)
```

В отличие от первого раза, теперь планировщиком используется индекс.

Задание 3. Создать индекс для поля extra.

При выполнении будем использовать эффективное индексирование данных jsonb. Выполнение самого SELECT выглядит следующим образом.

```
postgres=# select count(*) from users where extra @> '{"extra_data": true}'::jsonb;
 count
-------
 55299
(1 row)
```

Выполним следующую команду до создания индекса.

EXPLAIN SELECT COUNT(*) FROM users WHERE extra @> '{"extra_data": true}'::jsonb;

```
postgres=# EXPLAIN select count(*) from users where extra @> '{"extra_data": true}'::jsonb;
                            QUERY PLAN
-------------------------------------------------------------------
 Aggregate  (cost=22943.50..22943.51 rows=1 width=8)
   ->  Seq Scan on users  (cost=0.00..22941.00 rows=1000 width=0)
         Filter: (extra @> '{"extra_data": true}'::jsonb)
(3 rows)
```

Создадим индекс и выполним предыдущую команду еще раз.

CREATE INDEX idx_users_extra ON users USING GIN (extra);

```
postgres=# CREATE INDEX idx_users_extra ON users USING GIN (extra);
CREATE INDEX
postgres=# EXPLAIN select count(*) from users where extra @> '{"extra_data": true}'::jsonb;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=3372.28..3372.29 rows=1 width=8)
   ->  Bitmap Heap Scan on users  (cost=403.75..3369.78 rows=1000 width=0)
         Recheck Cond: (extra @> '{"extra_data": true}'::jsonb)
         ->  Bitmap Index Scan on idx_users_extra  (cost=0.00..403.50 rows=1000 width=0)
               Index Cond: (extra @> '{"extra_data": true}'::jsonb)
(5 rows)
```

В отличие от первого раза, теперь планировщиком используется индекс.

В соответствии с заданием каждый запрос был покрыт индексом, вследствие чего при выполнении каждого запроса seq scan не наблюдался.

5. Вывод

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