Начинать оптимизацию запроса следует с построения и анализа его плана выполнения

```
ok_ru=# EXPLAIN ANALYZE SELECT DISTINCT
ok_ru-#     first_name,
ok_ru-#     last_name,
ok_ru-#     g.name,
ok_ru-#     p.url
ok_ru-#     FROM users
ok_ru-#         JOIN groups g ON users.id = g.creator_id
ok_ru-#                 LEFT JOIN pictures p ON users.id = p.owner_id
ok_ru-#     WHERE name NOTNULL AND url NOTNULL;
                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=14.15..15.15 rows=100 width=78) (actual time=0.465..0.497 rows=92 loops=1)
   Group Key: users.first_name, users.last_name, g.name, p.url
   Batches: 1  Memory Usage: 40kB
   ->  Hash Join  (cost=8.77..13.15 rows=100 width=78) (actual time=0.285..0.381 rows=92 loops=1)
         Hash Cond: (p.owner_id = users.id)
         ->  Seq Scan on pictures p  (cost=0.00..3.00 rows=100 width=49) (actual time=0.023..0.060 rows=100 loops=1)
               Filter: (url IS NOT NULL)
         ->  Hash  (cost=7.52..7.52 rows=100 width=41) (actual time=0.249..0.250 rows=100 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 16kB
               ->  Hash Join  (cost=5.25..7.52 rows=100 width=41) (actual time=0.122..0.201 rows=100 loops=1)
                     Hash Cond: (g.creator_id = users.id)
                     ->  Seq Scan on groups g  (cost=0.00..2.00 rows=100 width=24) (actual time=0.019..0.046 rows=100 loops=1)
                           Filter: (name IS NOT NULL)
                     ->  Hash  (cost=4.00..4.00 rows=100 width=17) (actual time=0.090..0.091 rows=100 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 14kB
                           ->  Seq Scan on users  (cost=0.00..4.00 rows=100 width=17) (actual time=0.022..0.047 rows=100 loops=1)
 Planning Time: 2.094 ms
 Execution Time: 0.634 ms
```

Необходимо проанализировать структуру запроса и переработать его, если структура не выглядит оптимальной.

Следующим шагом необходимо проверить, существуют ли индексы, необходимые для быстрой работы данного запроса.

В нашем случае необходимо проверить наличие индексов на столбцы, которые используются в условии объединения:

```
ON users.id = g.creator_id
ON users.id = p.owner_id
```

Проверим, какие индексы созданы для таблиц пользователей, групп и фотографий:

```
ok_ru=# SELECT indexname FROM pg_indexes WHERE tablename = 'users';
    indexname
-----------------
 users_email_key
 users_pkey
(2 строки)


ok_ru=# SELECT indexname FROM pg_indexes WHERE tablename = 'groups';
    indexname
-----------------
 groups_name_key
 groups_pkey
(2 строки)


ok_ru=# SELECT indexname FROM pg_indexes WHERE tablename = 'pictures';
    indexname
------------------
 pictures_pkey
 pictures_url_key
(2 строки)
```

Мы видим что в таблицах есть индексы на столбцы первичных ключей users.id, groups.id и pictures.id, но нет индексов на столбцы внешних ключей pictures.owner_id и groups.creator_id.

Создаем нужные индексы:

```
ok_ru=# CREATE INDEX pictures_owner_id_fk ON pictures (owner_id);
NOTICE:  snitch: ddl_command_start CREATE INDEX
CREATE INDEX
ok_ru=# CREATE INDEX groups_creator_id_fk ON groups (creator_id);
NOTICE:  snitch: ddl_command_start CREATE INDEX
CREATE INDEX
```

То, что подкрашено красной линией отрабатывает ранее созданный триггер на команды.

Для того чтобы заставить планировщик использовать индексы отключим последовательное сканирование явным образом:

```
ok_ru=# SET enable_seqscan TO OFF;
SET
```

И перестроим план выполнения запроса:

```
ok_ru=# EXPLAIN ANALYZE SELECT DISTINCT
ok_ru-#     first_name,
ok_ru-#     last_name,
ok_ru-#     g.name,
ok_ru-#     p.url
ok_ru-#     FROM users
ok_ru-#         JOIN groups g ON users.id = g.creator_id
ok_ru-#                 LEFT JOIN pictures p ON users.id = p.owner_id
ok_ru-#     WHERE name NOTNULL AND url NOTNULL;
                                                                      QUERY PLAN

--------------------------------------------------------------------------------------------------------------------------------------------------------
---
 HashAggregate  (cost=51.79..52.79 rows=100 width=78) (actual time=0.320..0.340 rows=92 loops=1)
   Group Key: users.first_name, users.last_name, g.name, p.url
   Batches: 1  Memory Usage: 40kB
   ->  Hash Join  (cost=13.45..50.79 rows=100 width=78) (actual time=0.115..0.265 rows=92 loops=1)
         Hash Cond: (users.id = p.owner_id)
         ->  Merge Join  (cost=0.29..35.87 rows=100 width=41) (actual time=0.027..0.138 rows=100 loops=1)
               Merge Cond: (users.id = g.creator_id)
               ->  Index Scan using users_pkey on users  (cost=0.14..21.60 rows=100 width=17) (actual time=0.013..0.045 rows=100 loops=1)
               ->  Index Scan using groups_creator_id_fk on groups g  (cost=0.14..13.64 rows=100 width=24) (actual time=0.009..0.046 rows=100 loops=
1)
                     Filter: (name IS NOT NULL)
         ->  Hash  (cost=11.92..11.92 rows=100 width=49) (actual time=0.078..0.078 rows=100 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 17kB
               ->  Bitmap Heap Scan on pictures p  (cost=8.92..11.92 rows=100 width=49) (actual time=0.022..0.046 rows=100 loops=1)
                     Recheck Cond: (url IS NOT NULL)
                     Heap Blocks: exact=2
                     ->  Bitmap Index Scan on pictures_url_key  (cost=0.00..8.89 rows=100 width=0) (actual time=0.017..0.017 rows=102 loops=1)
                           Index Cond: (url IS NOT NULL)
 Planning Time: 0.666 ms
 Execution Time: 0.469 ms
(19 строк)
```

После ввода индексов, есть небольшой прирост производительности.

Графическое представление каждого шага запроса: