OPTIMIZING POSTGRESQL PERFORMANCE IN DJANGO

Ahmad Fanaei

A BIT ABOUT ME

- Python & Django developer
- backend & data team @ **torob.com**

WHAT CAN DEGRADE DJANGO PERFORMANCE?

- n+1 select
- Select unnecessary stuff
- Blocking migrations
- Pagination
- Slow queries

WHAT TOOLS CAN BE USED TO FIND AND FIX THESE ISSUES

- django_debug_toolbar
- assertNumQueries
- squawk
- pg_stat_statment

DJANGO ORM IS AMAZING BUT ...

ALL NON-TRIVIAL ABSTRACTIONS, TO SOME DEGREE, ARE LEAKY. 1

- unexpected query
- not optimal queries
- blocking migrations

TOOLS

DJANGO_DEBUG_TOOLBAR 2

- this is an amazing test

ASSERT NUM QUERIES

```
def test_past_question(self):
    """
    The detail view of a question with a pub_date in the past
    displays the question's text.
    """
    past_question = create_question(question_text='Past Question.', days=-5)
    url = reverse('polls:detail', args=(past_question.id,))

    with self.assertNumQueries(1):
        response = self.client.get(url)
    self.assertContains(response, past_question.question_text)
```

ASSERT NUM QUERIES

```
root@6d9a68a2769f:/app# python manage.py test
polls.tests.QuestionDetailViewTests.test_past_question
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
======================================================================
FAIL: test_past_question (polls.tests.QuestionDetailViewTests)
The detail view of a question with a pub_date in the past
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/app/polls/tests.py", line 128, in test_past_question
    response = self.client.get(url)
  File "/usr/local/lib/python3.9/site-packages/django/test/testcases.py", line 86, in
__exit__
    self.test_case.assertEqual(
AssertionError: 2 != 1 : 2 queries executed, 1 expected
Captured queries were:
1. SELECT "polls_question"."id", "polls_question"."question_text",
"polls_question"."pub_date" FROM "polls_question" WHERE ("polls_question"."pub_date" <=
'2022-02-13T22:14:30.093869+00:00'::timestamptz AND "polls_question"."id" = 1) LIMIT 21
2. SELECT "polls_choice"."id", "polls_choice"."question_id",
"polls_choice"."choice_text", "polls_choice"."votes" FROM "polls_choice" WHERE
"polls_choice"."question_id" = 1

----------------------------------------------------------------------
Ran 1 test in 0.034s

FAILED (failures=1)
Destroying test database for alias 'default'...
```

SQUAWK 3

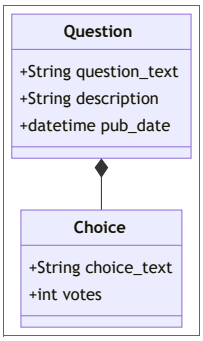
```
❯ squawk example.sql
example.sql:2:1: warning: prefer-text-field

   2 | --
   3 | -- Create model Bar
   4 | --
   5 | CREATE TABLE "core_bar" (
   6 |     "id" serial NOT NULL PRIMARY KEY,
   7 |     "alpha" varchar(100) NOT NULL
   8 | );

  note: Changing the size of a varchar field requires an ACCESS EXCLUSIVE lock.
  help: Use a text field with a check constraint.
```

PG_STAT_STATMENT

```
postgres=# CREATE EXTENSION pg_stat_statements;
ERROR:  extension "pg_stat_statements" already exists
postgres=# \d pg_stat_statements
                    View "public.pg_stat_statements"
       Column        |       Type       | Collation | Nullable | Default
---------------------+------------------+-----------+----------+---------
 query               | text             |           |          |
 calls               | bigint           |           |          |
 total_time          | double precision |           |          |
 min_time            | double precision |           |          |
 max_time            | double precision |           |          |
 mean_time           | double precision |           |          |
 stddev_time         | double precision |           |          |
```

EXAMPLE

DATA MODEL


Question
+String question_text
+String description
+datetime pub_date
Choice
+String choice_text
+int votes

API

```
urlpatterns = [
    path('', views.IndexView.as_view(), name='index'),
    path('<int:pk>/', views.DetailView.as_view(), name='detail'),
    path('<int:pk>/results/', views.ResultsView.as_view(), name='results'),
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

N+1 QUERY PROBLEM

Question number 1
- a1 -- 0 votes
- b1 -- 1 vote
- c1 -- 2 votes

Question number 2
- a2 -- 0 votes
- b2 -- 1 vote
- c2 -- 2 votes

Question number 3
- a3 -- 0 votes
- b3 -- 1 vote
- c3 -- 2 votes

Question number 4
- a4 -- 0 votes
- b4 -- 1 vote
- c4 -- 2 votes

Question number 5
- a5 -- 0 votes
- b5 -- 1 vote
- c5 -- 2 votes

```
class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_question_list'

    def get_queryset(self):
        """
        Return the last five published questions
        """
        return (
            Question.objects
            .filter(pub_date__lte=timezone.now())
            .order_by('-pub_date')[:5]
        )
```

```
<div style="padding: 16px;">
{% for question in latest_question_list %}
    <a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a>
    <ul style="margin-top: 0">
        {% for choice in question.choice_set.all %}
            <li>{{ choice.choice_text }} -- {{ choice.votes }} vote{{
choice.votes|pluralize }}</li>
        {% endfor %}
    </ul>
{% endfor %}
</div>
```

Question number 1
- c1 -- 2 votes
- b1 -- 1 vote
- a1 -- 0 votes

Question number 2
- c2 -- 2 votes
- b2 -- 1 vote
- a2 -- 0 votes

Question number 3
- c3 -- 2 votes
- b3 -- 1 vote
- a3 -- 0 votes

Question number 4
- c4 -- 2 votes
- b4 -- 1 vote
- a4 -- 0 votes

Question number 5
- c5 -- 2 votes
- b5 -- 1 vote
- a5 -- 0 votes

select_related: Returns a QuerySet that will “follow” foreign-key relationships
prefetch_related: Returns a QuerySet that will automatically retrieve, in a single batch, related objects

```
class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_question_list'

    def get_queryset(self):
        """
        Return the last five published questions
        """
        return (
            Question.objects
            .prefetch_related('choice_set')
            .filter(pub_date__lte=timezone.now())
            .order_by('-pub_date')[:5]
        )
```

Question number 1
- a1 -- 0 votes
- b1 -- 1 vote
- c1 -- 2 votes

Question number 2
- a2 -- 0 votes
- b2 -- 1 vote
- c2 -- 2 votes

Question number 3
- a3 -- 0 votes
- b3 -- 1 vote
- c3 -- 2 votes

Question number 4
- a4 -- 0 votes
- b4 -- 1 vote
- c4 -- 2 votes

Question number 5
- a5 -- 0 votes
- b5 -- 1 vote
- c5 -- 2 votes

ASSERT NUM QUERIES

```
def test_past_question(self):
    """
    Questions with a pub_date in the past are displayed on the
    index page.
    """
    create_question(question_text="Past question.", days=-30)
    create_question(question_text="Past question. 2", days=-15)
    with self.assertNumQueries(2):
        self.client.get(reverse('polls:index'))
```

EXCESS SELECTION

By default django selects all columns.

```
list(
    Question.objects
    .prefetch_related('choice_set')
    .filter(pub_date__lte=timezone.now())
    .order_by('-pub_date')[:5]
)
```

```
SELECT
  "polls_question"."id",
  "polls_question"."question_text",
  "polls_question"."description",
  "polls_question"."pub_date"
FROM "polls_question"
WHERE
  "polls_question"."pub_date" <= \'2022-02-14T21:19:25.545889+00:00\'::timestamptz
ORDER BY "polls_question"."pub_date"
DESC LIMIT 5
```

Limit selected columns using `values` or `only`

```
list(
    Question.objects
    .prefetch_related('choice_set')
    .only('question_text')
    .filter(pub_date__lte=timezone.now())
    .order_by('-pub_date')[:5]
)
```

```
SELECT
  "polls_question"."id",
  "polls_question"."question_text"
  FROM "polls_question"
  WHERE "polls_question"."pub_date" <= \'2022-02-14T21:21:36.022491+00:00
\'::timestamptz
  ORDER BY "polls_question"."pub_date"
  DESC LIMIT 5
```

A FEW MONTH LATER

Question 2 2022-02-13
- c4 -- 2 votes
- b4 -- 1 vote
- a4 -- 0 votes

Question 3 2022-02-13
- c7 -- 2 votes
- b7 -- 1 vote
- a7 -- 0 votes

Question 4 2022-02-13
- c10 -- 2 votes
- b10 -- 1 vote
- a10 -- 0 votes

Question 5 2022-02-13
- c13 -- 2 votes
- b13 -- 1 vote
- a13 -- 0 votes

Question 1 2022-02-13
- c1 -- 2 votes
- b1 -- 1 vote
- a1 -- 0 votes

ALWAYS PREFER `values` OVER `only`

BLOCKING MIGRATIONS

MAKE QUESTIONS UNIQUE

```
question_text = models.CharField(max_length=200, unique=True)
```

```
BEGIN;
--
-- Alter field question_text on question
--
ALTER TABLE "polls_question" ADD CONSTRAINT
"polls_question_question_text_e0c682c7_uniq" UNIQUE ("question_text");
CREATE INDEX "polls_question_question_text_e0c682c7_like" ON "polls_question"
("question_text" varchar_pattern_ops);
COMMIT;
```

> *… a standard index build locks out writes (but not reads) on the table until it's done*

SQUAWK

```
root@6d9a68a2769f:/app# ./squawk-linux-x86_64 out.sql
out.sql:2:1: warning: disallowed-unique-constraint

   2 | --
   3 | -- Alter field question_text on question
   4 | --
   5 | ALTER TABLE "polls_question" ADD CONSTRAINT
"polls_question_question_text_e0c682c7_uniq" UNIQUE ("question_text");

  note: Adding a UNIQUE constraint requires an ACCESS EXCLUSIVE lock which blocks
reads.
  help: Create an index CONCURRENTLY and create the constraint using the index.
```

```
out.sql:6:1: warning: require-concurrent-index-creation

   6 | CREATE INDEX "polls_question_question_text_e0c682c7_like" ON "polls_question"
("question_text" varchar_pattern_ops);

  note: Creating an index blocks writes.
  help: Create the index CONCURRENTLY.

find detailed examples and solutions for each rule at https://squawkhq.com/docs/rules
```

PAGINATION

WHAT IS THE PROBLEM

Question 10005 2022-02-13
- c30013 -- 2 votes
- b30013 -- 1 vote
- a30013 -- 0 votes

Question 10004 2022-02-13
- c30010 -- 2 votes
- b30010 -- 1 vote
- a30010 -- 0 votes

Question 10003 2022-02-13
- c30007 -- 2 votes
- b30007 -- 1 vote
- a30007 -- 0 votes

Question 10002 2022-02-13
- c30004 -- 2 votes
- b30004 -- 1 vote
- a30004 -- 0 votes

Question 10001 2022-02-13
- c30001 -- 2 votes
- b30001 -- 1 vote
- a30001 -- 0 votes

« Previous Next »

DEFAULT PAGINATION

```
class IndexView(generic.ListView):
    template_name = 'polls/index.html'
    context_object_name = 'latest_question_list'
    paginate_by = 5

    def get_queryset(self):
        """
        Return the last five published questions (not including those set to be
        published in the future).
        """
        return Question.objects.prefetch_related('choice_set').filter(
            pub_date__lte=timezone.now()
        ).order_by('-pub_date', '-id')
```

```
    <div class="pagination">
        <span class="previous">
            {% if page_obj.has_previous %}
                <a href="?page={{ page_obj.previous_page_number }}">&laquo;
Previous</a>
            {% endif %}
        </span>
        <span class="next">
            {% if page_obj.has_next %}
            <a href="?page={{ page_obj.next_page_number }}">Next &raquo;</a>
            {% endif %}
        </span>
    </div>
```

Question 20000 2022-02-13
- c59998 -- 2 votes
- b59998 -- 1 vote
- a59998 -- 0 votes

Question 19999 2022-02-13
- c59995 -- 2 votes
- b59995 -- 1 vote
- a59995 -- 0 votes

Question 19998 2022-02-13
- c59992 -- 2 votes
- b59992 -- 1 vote
- a59992 -- 0 votes

Question 19997 2022-02-13
- c59989 -- 2 votes
- b59989 -- 1 vote
- a59989 -- 0 votes

Question 19996 2022-02-13
- c59986 -- 2 votes
- b59986 -- 1 vote
- a59986 -- 0 votes

Next »

WHY IS DEFAULT PAGINATION SLOW

- Counting number of items. **why?**
  - wasn't it possible to store count seperately?
  - we have index counting number of items using an index should be trivial!
- offset-limit pagination becomes slower in high pages

LIMIT-OFFSET

```
SELECT *
  FROM "polls_question"
  WHERE "polls_question"."pub_date" <= '2022-02-14T22:07:52.796083+00:00'::timestamptz
  ORDER BY "polls_question"."id" DESC
  LIMIT 5
  OFFSET 9995
```

KEYSET

```
SELECT *
  FROM "polls_question"
  WHERE
    "polls_question"."pub_date" <= '2022-02-14T22:07:52.796083+00:00'::timestamptz
    and "polls_question"."id" <= ...
  ORDER BY "polls_question"."id" DESC
  LIMIT 5
```

DJANGO-KEYSET-PAGINATION-PLUS

KEYSET PAGINATION ISSUES

- User can’t see total number of pages
- User can’t paginate to a specific page and must skip pages one-by-one

SLOW QUERY

FINDING SLOW QUERIES

```
SELECT
  sum(total_time) / 1000 / 60 as total_time,
  sum(calls) as total_calls,
  query
FROM pg_stat_statements
GROUP BY query
ORDER BY 1 DESC
LIMIT 200;
```

SAMPLE OUTPUT

```
      total_time      | total_calls |        query

----------------------+-------------+-------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
------------------
        0.01313467055 |          12 | SELECT "polls_question"."id",
"polls_question"."question_text", "polls_question"."description",
"polls_question"."pub_date" FROM "polls_question" WHERE "polls_question"."pub_date" <=
$1::timestamptz ORDER BY "polls_question"."pub_date" DESC, "polls_question"."id" DESC
LIMIT $2 OFFSET $3
 0.000491445183333333 |          12 | SELECT COUNT(*) AS "__count" FROM
"polls_question" WHERE "polls_question"."pub_date" <= $1::timestamptz
          1.65117e-05 |          12 | SELECT "auth_user"."id", "auth_user"."password",
"auth_user"."last_login", "auth_user"."is_superuser", "auth_user"."username",
"auth_user"."first_name", "auth_user"."last_name", "auth_user"."email",
"auth_user"."is_staff", "auth_user"."is_active", "auth_user"."date_joined" FROM
"auth_user" WHERE "auth_user"."id" = $1 LIMIT $2
 1.45243333333333e-05 |          12 | SELECT "polls_choice"."id",
"polls_choice"."question_id", "polls_choice"."choice_text", "polls_choice"."votes" FROM
"polls_choice" WHERE "polls_choice"."question_id" IN ($1, $2, $3, $4, $5)
 1.18463333333333e-05 |          12 | SELECT "django_session"."session_key",
"django_session"."session_data", "django_session"."expire_date" FROM "django_session"
WHERE ("django_session"."expire_date" > $1::timestamptz AND
"django_session"."session_key" = $2) LIMIT $3
```

EXPLAIN OUTPUT

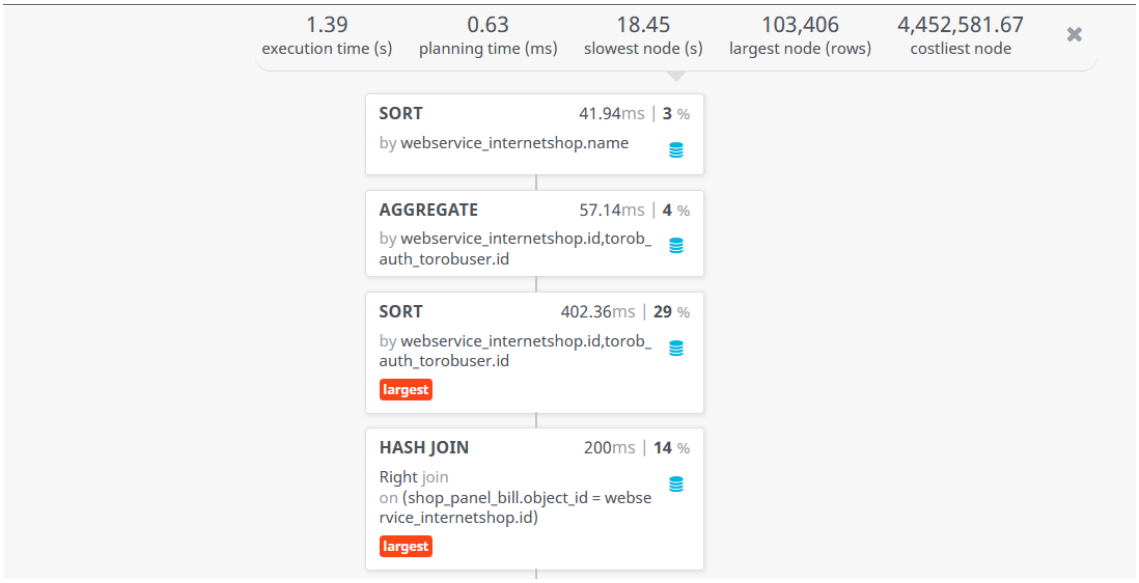
```
SELECT "polls_choice"."id",
       "polls_choice"."question_id",
       "polls_choice"."choice_text",
       "polls_choice"."votes"
  FROM "polls_choice"
 WHERE "polls_choice"."question_id" IN (202922, 202921, 202920, 202919, 202918)
```

EXPLAIN OUTPUT

```
QUERY PLAN
Index Scan using polls_choice_question_id_c5b4b260 on polls_choice  (cost=0.29..25.71
rows=15 width=18) (actual time=0.056..0.063 rows=15 loops=1)
  Index Cond: (question_id = ANY ('{202922,202921,202920,202919,202918}'::integer[]))
Planning Time: 1.078 ms
Execution Time: 0.139 ms
```

–

VISUALIZE EXPLAIN OUT PUT 4



HOW TO MAKE QUERIES FASTER

- add index or partial index
- increase statics collection
- tune auto-vaccum

THANK YOU

Premature optimization is the root of all evil – DonaldKnuth