Ban 1+N in Django

Mar 26th, 2023

I always thought of 1+N as a thing that you just keep in your head, catch on code reviews or via performance regressions. This worked well for a long time, however, the less control we have over our SQL queries the more likely it will sneak through those guards.

A small history dive

This used to be very visible and meant almost “do not perform SQL queries in a cycle”:

```
books = c.execute("SELECT id, title, author_id FROM books").fetchall()  # 1
for id, title, author_id in books:
    c.execute("SELECT full_name FROM authors WHERE id=?", [author_id])  # +N
```

With ORM and lazy loading this became a little bit less obvious:

```
books = Book.objects.filter(...)  # 1
for book in books:
    print(f"Book title: {book.title}, author: {book.author.full_name}")  # +N
```

With something so innocent as an attribute access making an SQL query, it’s much easier to miss it. Especially when this code spreads out, the ORM objects are passed to templates, which also have loops and sure they can do attribute access.

As project grows, as its database schema becomes more complicated, as your team grows too, this keeps adding up. And magic also adds up. One particular mention should be a GraphQL library, which resolves onto ORM automatically.

Back to the present

I tumbled on a couple of 1+Ns while reading a project code for an unrelated reason and it got me thinking – do I ever want Django to do that lazy loading stuff? And the answer was never. This was a misfeature for me, the need for such thing is quite circumstantial, usually when you load a list of things you need the same data about all of them, so it doesn’t make sense to lazy load extra data for each object separately. Either eager load or batch lazy load, the latter Django does not do.

So, anyway, if I don’t need this than I might as well prohibit it, which turned out to be quite easy to do:

```
from django.db.models.query_utils import DeferredAttribute

```

```
3
4  def _DeferredAttribute_get(self, instance, cls=None):
5      if instance is None:
6          return self
7      data = instance.__dict__
8      field_name = self.field.attname
9      if field_name in data:
10         return data[field_name]
11
12     # Raise an exception to prevent an SQL query
13     attr = f"{instance.__class__.__name__}.{field_name}"
14     message = f"Lazy fetching of {attr} may cause 1+N issue"
15     raise LookupError(message)
16
17 DeferredAttribute.__get__ = _DeferredAttribute_get
```

This way 1+N will blow up instead. Great, we'll catch it during tests. The thing is, however, if 1+Ns were passing our defences before they will probably continue now and this will explode in production. With this in mind, a flood guard and some explanations it transforms into:

```
1  import logging
2  import os
3  from django.db.models.query_utils import DeferredAttribute
4
5  logger = logging.getLogger(__name__)
6  attrs_seen = set()
7
8
9  def _DeferredAttribute_get(self, instance, cls=None):
10     from django.conf import settings  # monkeys go early, settings might not be available yet
11
12     if instance is None:
13         return self
14     data = instance.__dict__
15     field_name = self.field.attname
16
17     # Normally this accessor won't be called if field_name is in __dict__,
18     # we need this part so that DeferredAttribute descendants with __set__ play nice.
19     if field_name in data:
20         return data[field_name]
21
22     # If it's not there already then prevent an SQL query or at least notify we are doing smth bad
23     attr = f"{instance.__class__.__name__}.{field_name}"
24     # Only trigger this check once per attr to not flood Sentry with identical messages
25     if attr not in attrs_seen:
26         attrs_seen.add(attr)
27         message = f"Lazy fetching of {attr} may cause 1+N issue"
28         # We stop in DEBUG mode and if inside tests but let production to proceed.
29         # Using LookupError instead of AttributeError here to prevent higher level "handling" this.
30         if settings.DEBUG or "PYTEST_CURRENT_TEST" in os.environ:
31             raise LookupError(message)
32         else:
33             logger.exception(message)
34
35     # Proceed normally
36     return _DA_get_original.original(self, instance, cls)
37
38 _DA_get_original, DeferredAttribute.__get__ = DeferredAttribute.__get__, _DeferredAttribute_get
```

Which is ready to be used as is. Simply need to put or import it somewhere.

P.S. A small bonus – how I tried to make ChatGPT write this post for me. It was mostly failure :), but refactoring the code sample was done nicely.

Posted by Alexander Schepanovski Mar 26th, 2023 CS, Python

Tweet « Metaprogramming Beyond Decency: Part 2

Comments, also at Hacker News and Reddit

About Me

I write in python, js and occasionally english. I also borrow ideas from a variety of languages. You might want to follow me or look up my Github profile.

Recent Posts

- Ban 1+N in Django
- Metaprogramming Beyond Decency: Part 2
- Metaprogramming Beyond Decency: Part 1
- Boiling React Down to a Few Lines in jQuery
- Growing Over Backward Incompatibility

GitHub Repos

- django-cacheops

  A slick ORM cache with automatic granular event-driven invalidation.

- funcy

  A fancy and practical functional tools

- CommentsAwareEnter

  Smart Enter in line comments in Sublime Text 2/3

- whatever

Easy anonymous functions by partial application of operators

- pg-bricks

  Higher level PostgreSQL client for Node.js

@Suor on GitHub