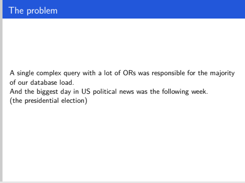
Dealing with a Monster Query

About Me

To be clear, I'm not the one who did the initial port to Elixir. There were contractors involved. I picked it up and started extending it, and it generally worked great...except for this one query.

About Me

In 2020, I learned Elixir because my team at Axios launched our mobile app and immediately crashed Apollo.
Every morning.
At 6AM.
I'm not a morning person.

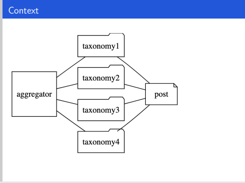


The problem

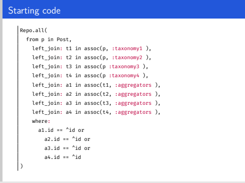
A single complex query with a lot of ORs was responsible for the majority of our database load.
And the biggest day in US political news was the following week.
(the presidential election)

Moving as much computation into the database and out of the code as possible is a pretty common piece of advice for optimization. You know, sort and limit on the database instead of in code, so the database has less stuff to return to start with.
This is a story of when to refactor the opposite direction and how a couple Elixir features helped.

If you've worked for a newspaper during a presidential election, you know that everyone is on call that night, and users are hammering the servers, trying to find out who won.


Context

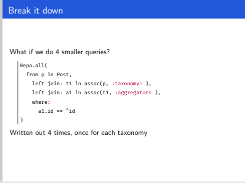
It's pretty standard for a CMS to support multiple taxonomies, like categories AND tags. We depended, at the time, on 4 separate taxonomies to decide what you see when you load up the Axios mobile app.



Starting code

```
Repo.all(
  from p in Post,
    left_join: t1 in assoc(p, :taxonomy1 ),
    left_join: t2 in assoc(p, :taxonomy2 ),
    left_join: t3 in assoc(p :taxonomy3 ),
    left_join: t4 in assoc(p :taxonomy4 ),
    left_join: a1 in assoc(t1, :aggregators ),
    left_join: a2 in assoc(t2, :aggregators ),
    left_join: a3 in assoc(t3, :aggregators ),
    left_join: a4 in assoc(t4, :aggregators ),
    where:
      a1.id == ^id or
        a2.id == ^id or
        a3.id == ^id or
        a4.id == ^id
)
```

We knew from AWS stats that this was the toughest query. I've simplified it here to leave out date ranges, sorting, etc, but the original Postgres analyze result was a cost of 3614 and 8 millisecond execution time. Let's walk through refactoring this to be super-fast.

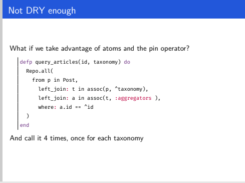

Break it down

What if we do 4 smaller queries?

```
Repo.all(
  from p in Post,
    left_join: t1 in assoc(p, :taxonomy1 ),
    left_join: a1 in assoc(t1, :aggregators ),
    where:
      a1.id == ^id
)
```

Written out 4 times, once for each taxonomy

That alone gets us down to Postgres saying its cost is 16 and execution time is 0.125ms, so we're on the right track. But it's no prettier than the original.



Not DRY enough

What if we take advantage of atoms and the pin operator?

```
defp query_articles(id, taxonomy) do
  Repo.all(
    from p in Post,
      left_join: t in assoc(p, ^taxonomy),
      left_join: a in assoc(t, :aggregators ),
      where: a.id == ^id
  )
end
```

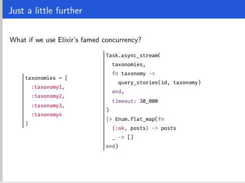
And call it 4 times, once for each taxonomy

Thanks to the ability to pass around atoms and dereference them inside Ecto queries, we can at least abstract that query into a function. But we still have to write 4 calls to that function, which is still a little ugly.

2023-02-05

Dealing with a Monster Query

└─Just a little further



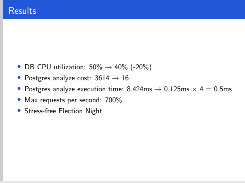
Just a little further

What if we use Elixir's famed concurrency?

```
taxonomies = [
  :taxonomy1,
  :taxonomy2,
  :taxonomy3,
  :taxonomy4
]
```

```
Task.async_stream(
  taxonomies,
  fn taxonomy ->
    query_stories(id, taxonomy)
  end,
  timeout: 30_000
)
|> Enum.flat_map(fn
  {:ok, posts} -> posts
  _ -> []
end)
```

Hey, there we go. Now, we can make all 4 queries at the same time, just passing in the list of taxonomies. That's really nice if you ever need more taxonomies. (Guess what? It was 5 by the time I left Axios.)



Results

- DB CPU utilization: 50% $\rightarrow$ 40% (-20%)
- Postgres analyze cost: 3614 $\rightarrow$ 16
- Postgres analyze execution time: 8.424ms $\rightarrow$ 0.125ms $\times$ 4 $=$ 0.5ms
- Max requests per second: 700%
- Stress-free Election Night

This wasn't the first newspaper I ever worked for. I remember the stress of mid-term elections, smooth sailing for the presidential was a nice change. And look, our day-to-day database CPU usage went down by 20% too.