For Want of a JOIN

Originally posted 2022-11-27

Tagged: software_engineering

Obligatory disclaimer: all opinions are mine and not of my employer

---

A story of a data pipeline gone wrong.

For want of a JOIN

Once upon a time, I was on a small team tasked with building a data pipeline. On this team was someone we'll call Bob, who was fresh out of grad school and hadn't had much experience with software engineering, and no experience with SQL. As part of this data pipeline, Bob had to combine the data from two tables. JOIN is a core feature of SQL, so this ought to be simple, right? Unfortunately, Bob hadn't learned any SQL past SELECT and WHERE.

So, instead of joining the two tables:

```
python_client.query("SELECT * FROM A JOIN B ON A.id = B.id WHERE...")
```

Bob wrote:

```
rows = python_client.query("SELECT id FROM B WHERE...")
result = python_client.query("SELECT * FROM A WHERE id IN (%s)" % ',
'.join(rows))
```

It worked, and other than the unnecessary second round-trip to the database, this worked just fine for our launch dataset. We had deadlines looming, so this code got checked in as is.

For want of a bounded query string

We launched, and started running this pipeline daily on the latest data dumps. One day, we hit the following error message:

```
ERROR: SQL query may not exceed 10,000 characters
```

Oops. As the JOIN condition grew, the generated SQL query was growing longer and longer, until BigQuery rejected it outright. Bob knew how to fix this problem! The IDs were at most 8 characters each, so accounting for the commas and whitespace, he could make sure to stay under the character limit:

```
rows = python_client.query("SELECT id FROM B WHERE...")
result = []
for chunk in more_itertools.chunked(rows, 1000):
    result.extend(python_client.query("SELECT * FROM A WHERE id IN (%s)" %
', '.join(chunk)))
```

The code chugged along.

For want of a fast pipeline

As our dataset continued to grow, the number of chunks in this `for` loop ticked upwards. Eventually, as the pipeline execution time started to push several hours, the team made a push to optimize the pipeline. Bob added concurrent queries to parallelize the code:

```
rows = python_client.query("SELECT id FROM B WHERE...")
pending_queries = []
for chunk in more_itertools.chunked(rows, 1000):
    pending_queries.append(python_client.run("SELECT * FROM A WHERE id IN
(%s)" % ', '.join(chunk)))

results = list(itertools.chain(*(q.result() for q in pending_queries)))
```

This brought execution time back down to a more tolerable slowness.

For want of parallel queries

Unfortunately, this solution was but a band-aid. We ran into the following error not long afterwards:

```
WARNING: Concurrent INTERACTIVE query limit reached (50). To increase
concurrent queries, drop to BATCH priority, or contact us to increase your
quota limits.
```

Bob did both, firing off an email to our GCP contact and dropping our query priority to BATCH. I don’t remember what happened to our quota, but what I do remember is that our pipeline didn’t get any faster… BATCH priority meant that our queries could take arbitrarily long to execute, depending on whether BigQuery was having a busy day! Our pipeline execution times bounced around between a few hours to half a day. Despite each day’s pipeline run kicking off around midnight, we started getting questions from our analysts around 8AM about where today’s data was.

The project…?

I didn’t stay around long enough to see the next iteration of this train wreck.

As they say, “A month in the lab saves an hour in the library”, and I’d definitely commented on the JOIN statements during the initial code review. Unfortunately this change had been buried in a 2,000-line diff, and with looming launch deadlines, our TL/M told us to ship it.

In retrospect, the missing JOINs were just a smoulder. That smoulder eventually turned into a fire, but many other things were already on fire. The aforementioned TL/M, a first time homeowner, had their water pipes burst over the holidays, just weeks before our launch date in January. I’ve never seen someone under so much stress to try and make an immovable launch deadline! We also had to politely tell our partners that, no, CSV didn’t just mean “wrap the fields in double quotes and join them with commas and newlines”. Our on-call shifts were best described as disappointing. Imagine being woken up at 4AM to find out that some user submitted a fake phone number, uncovering an edge case where the frontend phone number validator disagreed with the backend validator. Of course, our leadership, who mandated that all signup errors warranted paging the on-call, was equally to blame! And when the fix was submitted for the phone number bug, bureaucratic Q/C processes meant that our on-calls continued to be paged for bogus phone numbers for a few more weeks. So yes, much fire.

What lessons should you take away from this episode? Here’s a few:

- Learn the capabilities of your tools. With the exception of NPM modules, most tools are designed to solve problems, possibly the ones you have.
- Don’t let junior SWEs get 2000 lines into a change before submitting a pull request.
- Operate on data where it resides. Python is a control plane, not a data plane.
- If you encounter an unusually round system limit, you’re probably using the system in a way its designers never imagined.
- Parallelizing something you haven’t optimized first is as likely to make it slower as it is to speed it up.
