Database-centric Backends: What has your experience been like?

ask databases

▲ 6 authored by ratsclub 5 hours ago | 1 comment

I started my programming career working on a proprietary ERP (Enterprise Resource Planning) system that still runs on a modified Clipper compiler. Most of my teammates had been working on this system longer than I have been alive, and throughout the years, they would always stress the importance of writing a “database-centric” backend. Now, what they meant by “database-centric” (consider “database” to refer to either Microsoft SQL Server or PostgreSQL) were as follows:

1. Views were the contracts; you would write them thinking of your team and others.
2. GET HTTP endpoints should avoid holding logic as they should be contained within the views.
3. Inserting and modifying data should be done through the HTTP Endpoints to avoid giving write access to the database (database access was usually given to third parties too).

We followed these rules, and they worked quite well. I recall that we would catch contract breakages quite often because we were also consuming the views we wrote, and performance was easier to tackle as we only had to tune the database (and the tooling for SQL databases is awesome).

Given the context, I now have the same amount of experience working as a regular backend software engineer. I write RESTful or gRPC APIs in Go, .NET, and Python, though I use the same databases as in my previous experience. However, I have noticed that people usually avoid using the databases in the same way as I used to. Instead of views, we use OpenAPI or Protocol Buffer to define our contracts. Querying data is also done through HTTP endpoints instead of the views. I see that access to databases is usually discouraged altogether, which I find strange as, in my experience, the database is even the piece that has the least downtime of the entire stack.

I must confess that I still haven’t had the chance to work with the “database-centric” setup in my newest experiences. For this reason, I want to ask you! Have you ever worked this way?

- What were the main pain-points?
- Which tools did you use?
- Would you have done anything differently?

Feel free to describe your experiences with similar setups!

You must be logged in to leave a comment.

Post Preview

▲ Student 29 minutes ago | link

The main pain points of views as interfaces are:

1. If your consumers can’t handle making joins you can end up with views that have a lot of joins, hitting speed (or if you change the data model you can end up there)
2. Your consumers need to actually be able to handle using a sql database or you end maintaining their queries for them
3. Your consumers can write stupid queries that take down the database.
4. Versioning is hard because you can only have one version of a view at a time unless you do search path hacks, and that still doesn’t help with the data model backing the view changing.