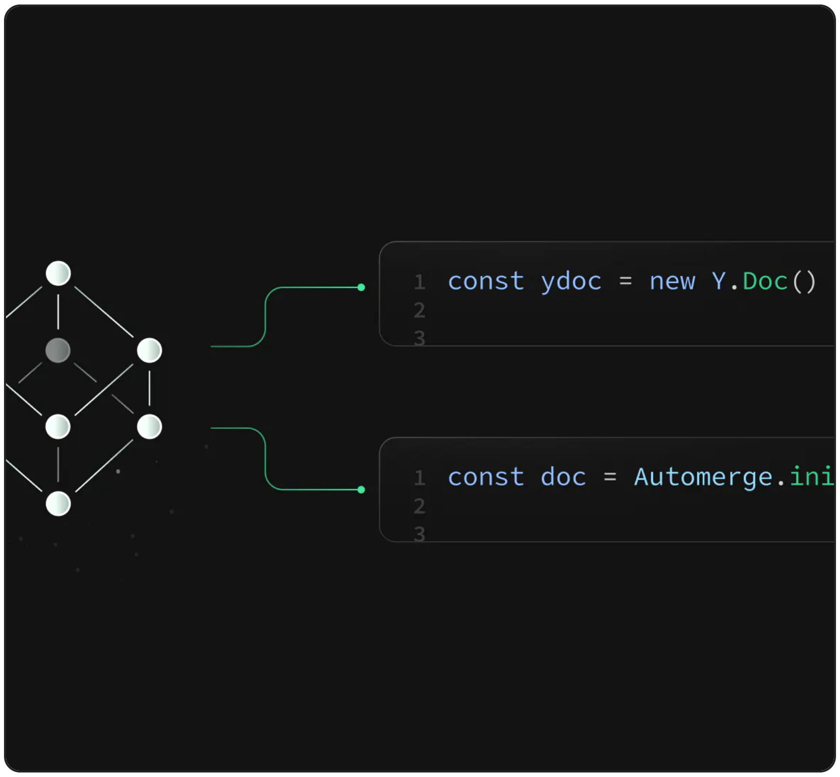
Back

Blog post

pg_crdt - an experimental CRDT extension for Postgres

2022-12-10 • 9 minute read

Paul Copplestone
CEO and Co-Founder



```
const ydoc = new Y.Doc()
```

```
const doc = Automerge.ini
```

Today we're open-sourcing an EXPERIMENTAL extension for CRDTs, `pg_crdt`. The GitHub repo is here. There are instructions for running it locally in the README.

When we released the new multiplayer features for our Realtime engine, it took 30 minutes for someone to ask if we'd add CRDT support.

> *"Anyone from Supabase here, do you have any plans to build in support for CRDT toolkits such as Yjs or AutoMerge for these features? It would make working with them so much easier if there was a plug and play backend."*

@samwillis on HackerNews

pg_crdt has not been released onto the Supabase platform (and it may never be). We're considering many options for offline-sync/support and, while CRDTs will undoubtedly factor in, we're not sure if this is the *right* approach. Hopefully this release generates a healthy discussion about the various ways we can do it at Supabase.

What's a CRDT?

A CRDT (Conflict-free Replicated Data Type) is a data structure. More accurately, a family of data structures. They enable collaborative apps like Figma.

You already know what a "data structure" is: an Array is a good example. CRDTs are a *special* type of data structure designed to solve a specific problem: they can merge changes in a way that the final state of the data will be the same, no matter the order in which the updates were applied.

In simple terms, a CRDT allows multiple users to make changes to the same data without the need for a central authority to coordinate their actions.

Let’s use our Array example to demonstrate the problem they solve. Imagine you have an array (which is not a CRDT):

```
let fruit = ['Apple', 'Banana', 'Orange']
```

Now imagine you have 2 developers updating this array on their local computers. They both want to replace the fruit at the start of the array. The first user, let’s call her “Jenny”, does this:

```
fruit[0] = 'Grape' // the array is ['Grape', 'B
```

The developer sitting next to her, let’s call him “Jonny”, does the same:

```
fruit[0] = 'Mango' // the array is ['Mango', 'B
```

And now, since they are both developing on different machines, they push their changes to a remote server. When the updates land on the remote system, what will `fruit[0]` be? “Grape” or “Mango”?

Since this is one of those *basic* arrays, the answer is "the fruit array that arrives last”.

1. If Jonny’s fruit array arrives first, it will be saved.
2. After that, Jenny’s fruit arrived and overwrites Jonny’s changes.

But therein lies the problem: Jonny was the last developer to change his array of fruit, shouldn’t his changes be the ones that are saved?

That’s one of the things that CRDTs solve. There is an “array CRDT” which, when merged, will always have the same result. It doesn’t matter if Jonny’s array arrives first, or if Jenny’s arrives first - every time you merge them they would be able to determine that the Jonny’s was the last change the fruit array.

How does it do that? Some sort of algorithm, but you can ask ChatGPT to explain that one.

Offline philosophy

Collaborative apps are becoming more prolific as legacy software is rebuilt within a browser environment.

Collaboration is largely a data problem - how do we get one user's changes (data) to merge with another user's changes (data)? As a database provider it's natural fit for Supabase to provide the tooling for developers to build collaborative apps.

Before going too far down the "solution" rabbit hole at Supabase, we try to think about the long-term implications of adopting any technology. We're pretty boring - we don't make bets on technologies unless we think they will exist in 20 years.

I personally believe CRDTs are the future. For *some things*. If databases were invented today, I'm certain most of the effort would be spent developing CRDT databases or something with "offline algorithms" built-in. But tech is a real-world demonstration of the Lindy effect: the longer something has existed, the longer it's likely to exist in the future. That's why we bet on Postgres - with more than 30 years history, it's here to stay.

Faced with this reality, we should consider how to solve *offline with Postgres* and where CRDTs might fit into that picture.

For a long time I thought there could be a way to shoehorn Postgres row-level data into a CRDT, to give *truly* offline support. This may even still be possible, and it's one of the ideas we'll continue to pursue. But Postgres is a rich and evolving ecosystem, and I don't know whether it will be possible to -

a) find a merge strategy for an entire row, while simultaneously:

b) finding a merge strategy for every data type *within* that row (especially with the number of data types available through extensions)

It's possible that developers will need to be selective about their "level" of offline support, at least if they plan to use an "incumbent" databases. For the cases, a simple "last write wins" strategy is probably acceptable (see the excellent Replicache and Watermelon libraries), but there will be important pieces of their application where it is not.

Take this table of blog posts as an example:

```
create table posts (
  id serial primary key,
  title text,
  content text default ''
);
```

Perhaps it's not that important if the `title` has a "last write wins" strategy, because it's rarely updated and less likely to have a merge conflict. But it is critical to use a smarter algorithm for the `content` of the blog post, especially in a team where multiple users are editing a blog post at the same time.

That's a lot of background to get to what you probably want to know about:

Postgres CRDT extension

`pg_crdt` is an extension which adds CRDT support to Postgres as a data type. For example, using our table from above:

```
create table posts (
  id serial primary key,
  title text,
  content crdt.ydoc default crdt.new_ydoc()
);
```

The `content` column is now a Yjs Doc. Updates to this column are *commutative* and *idempotent*. This means that they can be applied in any order and multiple times, and the result will always be the same. Two people can edit the blog content at the same time, and they won't have any issues when their changes are saved in the database.

Support for Yjs, Automerge, and beyond

The Yjs and Automerge teams have done some excellent work to create Rust libraries for their CRDTs implementations. It was relatively trivial to wrap the libraries into a Postgres extension using the pgx framework.

At this stage it makes sense to give developers as many choices as possible in one extension. The CRDT space is nascent the algorithms are rapidly changing.

Importantly, both Yjs and Automerge have *JavaScript* and Rust implementations, which means they work natively in both a browser and a Postgres environment. Since collaborative applications are largely a client-side problem, CRDTs are more useful for developers if they have robust JavaScript libraries. In the future, if this extension becomes the approach we take, then Supabase will focus some resources on building Yjs/Automerge libraries for mobile devices too (Swift for iOS, Kotlin for Android).

Why not build this into Realtime?

An alternative approach for CRDT support in Supabase is to build support directly into Realtime and use it as an Authoritative server. In this scenario, Realtime would serialize the CRDT and save it to Postgres (probably as a `bytea` data type). Yjs has the concept of Providers which would facilitate this. You can see the difference between the approaches in the diagram below - on the left, Realtime acts as the “middleman” for saving the CRDT in the database, whereas on the right the CRDT is pushed directly to the database, and Realtime receives updates from Postgres. (Note also that clients can send peer-to-peer updates.)

This might still be our best option, but it should not be our first attempt. If we make Realtime the authority, it strongly couples developers to the Supabase infrastructure. By placing the CRDT into the database, it simplifies the architecture, enables other tools (like Debezium), and provides the possibility to update the database directly (for semi-realtime events like counters or page-views).

The Realtime engine seems like a great *compliment* for `pg_crdt`, but before we put it into production we need to solve a few (major) limitations.

Limitations

These are a few of the *known* limitations:

- Realtime broadcasts database changes from the Postgres write ahead log (WAL). The WAL includes a complete copy of the the underlying data so small updates cause the entire document to broadcast to all collaborators
- Frequently updated CRDTs produce a lot of WAL and dead tuples
- Large CRDT types in Postgres generate significant serialization/deserialization overhead on-update.

We're likely to discover more (no doubt from a few friendly HN comments).

Next steps

If you want to help with `pg_crdt` the best way is to get involved in the GitHub repo. We have enabled Discussions for any and all ideas. If you have experience with CRDTs and you like this approach, don't hesitate to contact one of the team.

Share this article

Last post

New Supabase Docs, built with Next.js
12 December 2022

Next post

Who We Hire at Supabase
9 December 2022

Related articles

- Multi-factor Authentication via Row Level Security Enforcement
- Supabase Storage v2: Image resizing and Smart CDN
- New Supabase Docs, built with Next.js
- pg_crdt - an experimental CRDT extension for Postgres
- Who We Hire at Supabase

View all posts

Build in a weekend, scale to millions

Start your project