Back

Blog post

Supabase Wrappers, a Postgres FDW framework written in Rust

2022-12-15 • 17 minute read

Paul Copplestone
CEO and Co-Founder

Bo Lu
Engineering

Oliver Rice
Engineering

Today we're releasing Supabase Wrappers, a framework for building Postgres Foreign Data Wrappers (FDW) which connects Postgres to external systems.

Foreign Data Wrappers are a core feature of PostgreSQL. We've extended this feature to query other databases or any other external system (including third-party APIs), using SQL.

We're releasing Wrappers today in Alpha, with support for Firebase and Stripe. Wrappers for Clickhouse, BigQuery, and Airtable are under development.

Example with Stripe

Let's step through a full example using the Stripe Wrapper.

Connecting to Stripe

First, let's give your Postgres database some authentication details to access your Stripe account:

```
create server stripe_server
foreign data wrapper stripe_wrapper
options (api_key 'sk_test_xxx');
```

Creating a Foreign Table

Now we can map your Stripe data to Foreign Tables, which are just like normal tables except that the data exists *outside* of your database.

```
-- Create a foreign table for your Stripe produ
create foreign table products (
  id text,
  name text,
  description text,
  default_price text
)
```

```
server my_stripe_server
options ( object 'products' );
```

Accessing remote data

After setting up your foreign table, you can query you Stripe products directly from your database:

```
-- Fetch all your stripe products in Postgres
select *
from products
limit 10;
```

Or from your application, using one of our client libraries:

```
import { createClient } from '@supabase/supabas

const SUPABASE_URL = 'https://xyzcompany.supaba
const SUPABASE_KEY = 'public-anon-key'

const supabase = createClient(SUPABASE_URL, SUF

const { data: stripeCustomers, error } = supaba
```

```
  .from('products')
  .select('id, name, description, default_price
  .limit(10)
```

Note: we kept this example simple, however for API security and code organization, you should store your foreign data in a separate schema.

Use cases

Once we've added more Wrappers, they enable various possibilities:

On-demand Big Data: Query your Data Warehouse on demand using Wrappers for Clickhouse, BigQuery, Snowflake, Oracle, and S3.

Simplified Onboarding: Migrate to Postgres from systems like Firebase, MySQL, and Airtable.

Simplified Development: Create a new Stripe customer object within a Postgres function.

AI capabilities: Run AI queries from your database using OpenAI's API.

Caching: use Postgres triggers to insert data into in-memory databases like Redis.

SRE and DevOps: Query your infrastructure state, like AWS and DNS records.

Web3 Apps: integrate with IPFS and blockchains like Ethereum.

Financial Apps: Build Financial applications using wrappers around Finance APIs.

Community analytics: Analyze your community engagement with GitHub, Slack, Discord, and Twitter wrappers.

On-demand ETL

In their 2017 paper, researchers from the University of Bologna investigated an approach to on-demand ETL:

> *"In traditional OLAP systems, the ETL process loads all available data in the data warehouse before users start querying them. In some cases, this may be either inconvenient (because data are supplied from a provider for a fee) or unfeasible (because of their size); on the other hand, directly launching each analysis query on source data would not enable data reuse, leading to poor performance and high costs. The alternative investigated in this paper is that of fetching and storing data on-demand."*

Their paper outlines the foundation for QETL (pronounced "kettle"): Query, Extract, Transform, Load. This differs from a more traditional ETL/ELT approach, where data is moved from one place to another. In QETL, the "Query" function allows data engineers to access data in a remote system even before moving it. This approach reduces the reliance on data engineering infrastructure, allowing teams to access operational insights faster.

The benefits of QETL

We've built upon this concept using PostgreSQL's FDW system. This a new tool for developers and data engineers, with several benefits:

1. **Simplicity:** the Wrappers API is just SQL, so data engineers don't need to learn new tools and languages.
2. **On-demand:** analytical data is available within your product without any additional infrastructure, and

the time it takes to retrieve that data is close to executing a query on the source.

3. **Always in sync:** data which isn't moved will always be in sync. Developers can set up local views which re-map remote data into clean local schemas.
4. **Integrated:** large datasets are available within your application, and can be joined with your operational/transactional data.
5. **Save on bandwidth:** only extract/load what you need.
6. **Save on time:** avoid setting up additional data pipelines.
7. **Save on Data Engineering costs:** less infrastructure to be managed.

QETL + Postgres

How does this look in action? Assuming that all of your analytical data is stored in Snowflake, you could create a foreign table inside your Supabase database:

```
create foreign table snowflake.order_history (
  id bigint,
  ts timestamptz,
  event text,
  user_id uuid
)
server my_snowflake_warehouse
options (table 'order_history', rowid_column 'i
```

Now from your Supabase database you can query your Snowflake data directly:

```
select * from snowflake.order_history limit 10(
```

You can even join remote data with your local tables to enrich existing operational data. For example, to figure

out how many times a user has purchased something from your store:

```
select
  users.id,
    count(order_history.event)
from
    snowflake.order_history
join
    auth.users on auth.users.id = snowflake.orc
where
  order_history.event = 'purchase' and
  order_history.user_id = '<some_user_id>';
```

We can either run these queries on demand or, for better query performance, we can run them in the background (using something like pg_cron), and materialize the data into a local table.

This gives us the basis of QETL:

> Query: run on-demand SQL queries, directly from your Postgres database.

Extract: run SQL `select` statements on external systems, either on demand or on a recurring basis.

Transform: use SQL aggregations and CTEs to transform the data.

Load: store transformed data into local tables for faster access.

This is a two-way process. It's equally useful to offload large datasets from your Postgres database to your Data Warehouse. With FDWs, this can be as simple as:

```
insert into snowflake.analytics
select * from analytics
where ts > (now() - interval '1 DAY');
```

On-demand ETL is a strong compliment for current ETL practices, and another tool in the toolbelt for Data Engineering and Developers that works with immediately with tools that interface with Postgres.

Postgres, the everything interface

In a recent Software Engineering episode Andy Pavlo (database Professor at Carnegie Mellon and Co-Founder of OtterTune), explored the future between “better databases” and “better interfaces” [00:37:18]:

> *“Specialized engines are always going to outperform general-purpose ones. The question is whether the specialized engine is going to have such a significant difference in performance that it can overcome the limitations of a general purpose one.*

...

The challenge oftentimes is this: is the benefit you're getting from a specialized engine because the Engine is different or the API is different?."

 Andy Pavlo

He goes on to explore the benefits of a Graph database vs a Relational database.

Our recent release of pg_graphql closes the gap on the graph use-case by building a GraphQL API directly into Postgres as an extension. While a specialized graph database might provide performance benefits over Postgres, perhaps one of the largest benefits is simply the Graph API which makes it easier to reason about the data.

With the introduction of Wrappers, we're hoping to close the gap on even more of these type of workloads.

An exciting part of the FDW approach is that it provides a common interface to the world: SQL. While it has many shortcomings, SQL is the lingua franca of data. Postgres' FDWs transform any API into a data set with a common interface. This "interface aggregator" is similar to the promise of GraphQL engines. The benefit of embedding this functionality in the database is that it exists at the *lowest level of the tech stack*. Everything that is built on top can leverage it. While Postgres cannot easily access the functionality of a GraphQL server, a GraphQL server can easily access the functionality of Postgres.

The FDW interface also future-proofs Postgres. Instead of keeping up with the latest technological advances,

Postgres can instead act as an interface whenever they develop. The recent advance in AI and ML is a great example of this: AI technology is developing faster than the time it would take to build a new “AI database”. With a FDW, Postgres can become the interface to this technology and many other technological advances in the future.

Developing Wrappers

Postgres has a builtin “Postgres FDW” allows querying one Postgres database from another. We've extended this functionality to support a variety of data sources, from Data Warehouses to APIs. This release includes two initial wrappers: Stripe and Firebase

Integration	Platform	Self-hosted	select	insert	update	delete
Firebase	🚧	✅	✅	🚧	🚧	🚧
Postgres	✅	✅	✅	✅	✅	✅
Stripe	🚧	✅	✅	🚧	🚧	🚧

With several more under development:

Integration	select	insert	update	delete
Airtable	✅	🚧	🚧	🚧
BigQuery	✅	✅	✅	✅
ClickHouse	✅	✅	✅	✅

Wrappers used pgx, extending it with FDW support. pgx is a Postgres extension framework written in Rust. Wrappers is very similar to Steampipe or Multicorn. We

opted to develop our own framework for several reasons:

> The current state of FDWs is in disarray. It's hard to know which FDWs are supported and functional. We think there's a benefit to colocating all FDWs in a single repository using modern tooling. This makes contributing simpler and maintenance faster.
>
> Wrappers has async support, which enables the development of RESTful API-based FDWs, like Stripe.
>
> It's written in Rust, which provides reliable performance, strong typing, data security, and “push down” is supported through the framework.

Supported types and Push Down

Wrappers supports a variety of types, including: `bool`, `i8`, `i16`, `f32`, `i32`, `f64`, `i64`, `String`, `Date`, `Timestamp`, and `JsonB`.

Foreign Data Wrappers have a concept of "push down". This means that the FDW runs the query on *remote* data source. This is useful for performance reasons, as the remote data source can often perform the query more efficiently than Postgres. Push down is also useful for security reasons, as the remote data source can enforce access control. Limited push-down support has been added as a Proof of Concept, but this will be a key focus of Wrappers.

You can follow development of all the Wrappers in the official GitHub Repo.

Next Steps

We're not "officially" releasing Wrappers onto the platform yet, although the brave and curious might be able to figure out how to use it "unofficially". Caveat emptor: there will be breaking changes.

We're excited to see what the community does with Wrappers. We're hoping that Wrappers will help to accelerate the adoption of Postgres as a data hub. If you're interested in getting involved or building your own Wrapper, don't hesitate to jump into the code and start developing with us.

> Star & Watch the GitHub repo: github.com/supabase/wrappers
>
> View the documentation: supabase.github.io/wrappers
>
> Supabase Launch Week: supabase.com/launch-week

More Launch Week 6

> Day 1: New Supabase Docs, built with Next.js

Share this article

Next post

Multi-factor Authentication via Row Level Security Enforcement

14 December 2022

Related articles

Supabase Wrappers, a Postgres FDW framework written in Rust

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

View all posts

Build in a weekend, scale to millions

Start your project

Product

- Database
- Auth
- Functions
- Realtime
- Storage
- Pricing
- Beta

Resources

- Support
- System Status
- Integrations
- Experts
- Brand Assets / Logos
- DPA
- SOC2

Developers

- Documentation
- Changelog
- Contributing
- Open Source

Company

- Blog
- Careers
- Company
- Terms of Service

SupaSquad

DevTo

RSS

Privacy Policy

Acceptable Use Policy

Service Level Agreement

Humans.txt

Lawyers.txt

Security.txt