Back

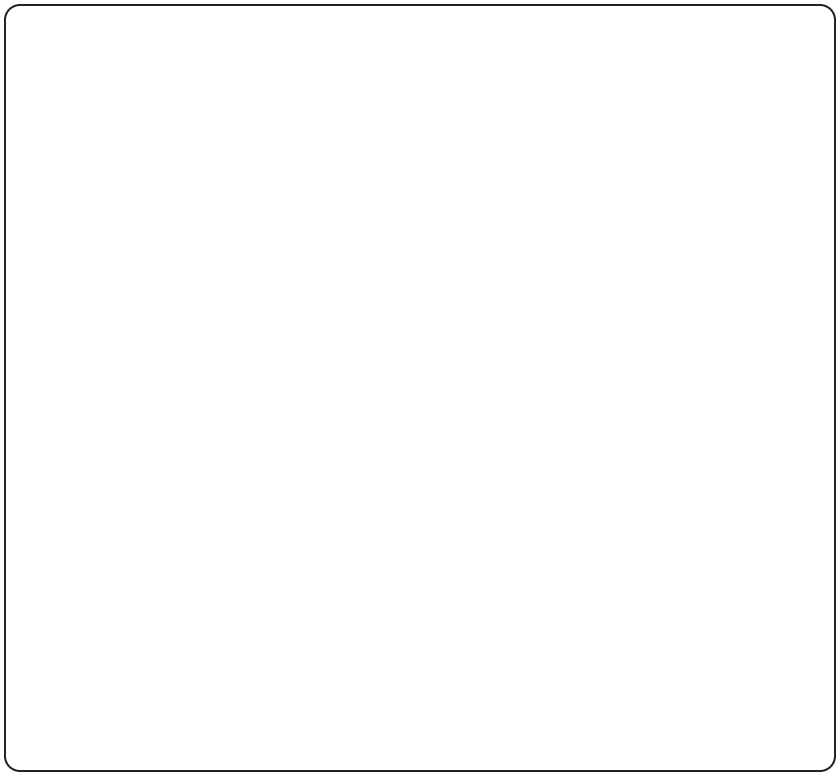
Blog post

dbdev: PostgreSQL Package Manager

2023-04-14 • 11 minute read

Oliver Rice
Engineering

John Dalton
Product Management Leader @ AWS



Today we're publicly previewing `database.dev`, a PostgreSQL package manager. At this stage the package registry is read-only. We've preloaded it with a handful of packages, or pglets (PostGres appLETs), to showcase some of the more interesting possibilities.

`dbdev` fills the same role for PostgreSQL as `npm` for JavaScript, `pip` for Python and `cargo` for Rust in that it enables publishing libraries and applications for repeatable deployment. We'll be releasing the tooling necessary for third-parties to publish `pglets` to the registry once we've collected some community feedback and incorporate any great new ideas. Our goal is to create an open ecosystem for packaging and discovering SQL.

The initial preview is compatible with new projects on the Supabase platform. It can also be installed on any

PostgreSQL instance that support `pg_tle` and `pgsql-http`.

Get Started with dbdev

The in-database client is the easiest way to get started. You can setup the installer by executing the SQL snippet available at database.dev/installer.

Once the `dbdev` client is present, `pglet`s can be installed from the registry as shown below:

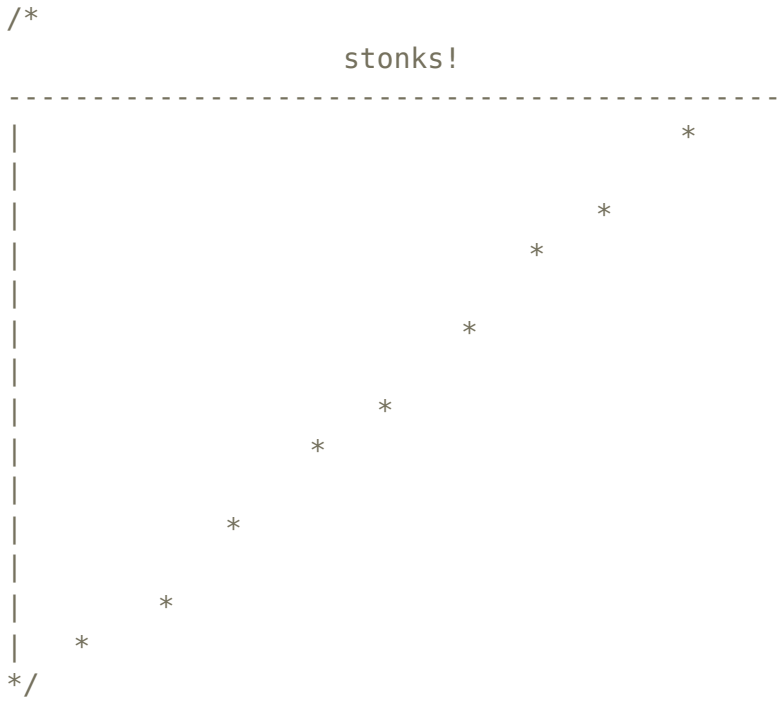
```
-- Load the package from the package index
select
  dbdev.install ('olirice-asciiplot');

-- Enable the extension
create extension "olirice-asciiplot" version '0
```

You can explore all available `pglet`s on database.dev.

Notice that PostgreSQL sees the `olirice-asciiplot` `pglet` as a native extension, rather than a raw snippet of SQL. That approach allows us to leverage PostgreSQL's builtin tooling for extension management.

With our extension installed, you can use it like any other PostgreSQL extension. Continuing with the `olirice-asciiplot` example, we can call the `scatter` function it provides to create an ASCII scatterplot:

```
select
  scatter(
    val::numeric, -- x
    val::numeric, -- y
    'stonks!',    -- title
    15,           -- height
    50            -- width
  )
from
  generate_series(1,10) z(val);
```



PostgreSQL's extension tooling is excellent, but it predates some practices learned from best-in-class package indexes like crates.io. To give developers a more modern development experience, we opted to layer additional strictness on top of what PostgreSQL imposes:

Versioning

The extension system has full support for versioning and migrations. Officially, PostgreSQL has loose constraints for version names. We made the choice to enforce a lite version of Semantic Versioning that restricts version numbers to `major.minor.patch` so authors can communicate bug-fixes, features, and breaking changes in a familiar way.

Namespaces

Two common challenges faced by package indexes are name squatting and typo squatting.

> Name squatting: reserving names for future use

Typo squatting: reserving misspelling of existing package

The ethics of name squatting get dicey at scale while typo squatting is widely viewed as malicious behavior. To mitigate both issues, all `pglet`s published to database.dev are namespaced to their owning organization or user's handle. For example a `pglet` named `olirice-index_advisor` was created by the account `olirice` under the name `index_advisor`. If another user, `some_user`, forks and republishes the project, it would be available under `some_user-index_advisor`. Problem solved ✅

Running on Supabase

database.dev is not coupled to the Supabase platform. `dbdev` can load SQL libraries on any PostgreSQL instance with the required base extensions. However, using `dbdev` in tandem with Supabase yields some extra possibilities.

Supabase reflects APIs directly from your database's structure, so a `pglet` can contain an entire stateful application, pre-configured with authentication, REST, GraphQL, and realtime change data capture all baked in!

For example, our friends at LangChain published a Supabase backend for their docs search tool that uses a hybrid of document embeddings and full text search to find relevant documents for a user's query

Its available at `langchain-hybrid_search` and here's how you'd set it up:

```
select
  dbdev.install ('langchain-hybrid_search');
```

```
create extension if not exists vector;

create extension "langchain-hybrid_search" sche
```

That creates the relevant `documents` table and associated search functions. Then, you can immediately hit it from your front end for best-in-class document search.

```
import { OpenAIEmbeddings } from 'langchain/emb
import { createClient } from '@supabase/supabas
import { SupabaseHybridSearch } from 'langchain

const privateKey = process.env.SUPABASE_PRIVATE
if (!privateKey) throw new Error(`Expected env

const url = process.env.SUPABASE_URL
if (!url) throw new Error(`Expected env var SUP

export const run = async () => {
  const client = createClient(url, privateKey)

  const embeddings = new OpenAIEmbeddings()

  const retriever = new SupabaseHybridSearch(em
    client,
    //  Below are the defaults, expecting that
    similarityK: 2,
    keywordK: 2,
    tableName: 'documents',
    similarityQueryName: 'match_documents',
    keywordQueryName: 'kw_match_documents',
  })

  const results = await retriever.getRelevantDo

  console.log(results)
}
```

Package Highlights

That's it for the `dbdev` announcement, but a package index is less interesting than what you can do with it! In

that vein, the following highlights a few of packages I thought were interesting enough to callout:

burggraf-pg_headerkit

`burggraf-pg_headerkit` is a toolkit for adding advanced features to PostgREST APIs (including Supabase REST):

- rate limiting
- IP allowlisting/denylisting
- request logging

and more.

For example, you could apply a deny listing to your API using `hdr.in_deny_list()` in a row level security policy or view:

```
select
  *
from
  app.memos
where
  not hdr.in_deny_list ();
```

olirice-index_advisor

`olirice-index_advisor` is one of the projects we cut from Launch Week 7. It is simple tool that takes a query and recommends indexes to minimize the “total_cost” according to the query’s explain plan.

We ultimately ran out of time to squeeze the feature in, but the optimizer works just fine:

```
select dbdev.install('olirice-index_advisor');
create extension if not exists hypopg;
create extension "olirice-index_advisor";
```

```
-- Create a dummy table
create table account(
    id int primary key,
    name text
);

-- Search for indexes to optimize "select id fr
select
    *
from
    index_advisor($$select id from account wher
```

which shows

```
| startup_cost_before | startup_cost_after | to
| ------------------- | ------------------ | --
| 0.00                | 1.17               | 25
```

In other words, it recommends the index `CREATE INDEX ON public.account USING btree (name)` which is expected to reduce the total cost from 25.88 to 6.40 for a 4x decrease.

`olirice-index_advisor` is compatible with tables, views, and materialized views. It can also see through views to find relevant indexes on underlying tables, and supports generic query arguments. For example, `$1` in `select id from account where name = $1`, which makes it compatible with queries from `pg_stat_statements` and queries generated by the REST API.

Keep an eye open for it in Launch Week 8.

michelp-adminpack

`michelp-adminpack` is a collection of tools helpful for administrating your database that we often use internally at Supabase. It holds views for reviewing useful info for debugging and optimizing performance like duplicate indexes, index usage, and table size, to name a few.

For example, to identify potentially unused indexes that can be dropped, you could use the `index_usage` view, which has columns for:

Column	Type
schemaname	name
tablename	name
num_rows	bigint
table_size	text
index_name	name
index_size	text
unique	text
number_of_scans	bigint
tuples_read	bigint
tuples_fetched	bigint

Limitations

There are several procedural languages (PL) that can be embedded in PostgreSQL and used to define functions. The ones that ship with stock PostgreSQL are `SQL`, and `pl/pgSQL` but there others that can be installed separately, including `pl/v8` for JavaScript, or `pl/perl` for Perl. A **trusted** language has been restricted to remove potentially hazardous functionality like access to the network stack and file system. `pl/v8` and `pl/perl` are examples of trusted languages. In contrast, `pl/python3u` is **untrusted**.

A Trusted Language Extension (TLE) is a PostgreSQL extension, written exclusively using trusted languages. In some ways that makes them less flexible than classic

extensions, which can have C language components (more on that in a second). The advantage to TLEs is that they don't require direct access to the PostgreSQL server's file system to install. That enables TLEs to be installed by end-users rather than by database administrators or hosting providers. TLEs are the enabling technology that allows a package manager like `dbdev` to function on hosted PostgreSQL platforms like Supabase.

For a more in-depth explanation of Trusted Language Extensions checkout AWS's pg_tle on Supabase blog post or dive into the code at github.com/aws/pg_tle.

A recent development in the PostgreSQL extension ecosystem is the 1.0 release of a new trusted language, `pl/rust`, allowing users to define SQL functions written in Rust. As a compiled language, `pl/rust` functions can execute an order of magnitude faster than `pl/pgSQL` for computationally heavy workloads. That closes the biggest capability gap between native extensions with C components and TLEs. `pl/rust` hasn't released to Supabase yet, but we're excited about rolling it out in the coming weeks.

Please Give Feedback

As this is a preview, we anticipate that there may be a few rough edges. If you do take the time to explore `db dev` at this stage, please contribute to its development at github.com/supabase/dbdev.

We are particularly interested in hearing about:

1. Any issues or bugs you encounter
2. Feature requests and suggestions for improvement

3 Contributions in the form of code, documentation, or testing

---

More Launch Week 7

- Designing with AI
- Supavisor
- Open Source Logging
- Self-hosted Deno Edge Functions
- Storage v3: Resumable Uploads with support for 50GB files
- Supabase Auth: SSO, Mobile, and Server-side support
- Community Highlight
- Studio Updates
- dbdev

Postgres TLE

Share this article

Last post

Supabase Studio 2.0: help when you need it most

14 April 2023

Next post

Trusted Language Extensions for Postgres

14 April 2023

Related articles

Supabase Studio 2.0: help when you need it most

dbdev: PostgreSQL Package Manager

Trusted Language Extensions for Postgres

Launch Week 7 Community Highlights

Supabase Auth: SSO, Mobile, and Server-side support

View all posts

Start your project

Product

- Database
- Auth
- Functions
- Realtime
- Storage
- Pricing
- Launch Week 7

Resources

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

Developers

- Documentation
- Changelog
- Contributing
- Open Source
- SupaSquad
- DevTo
- RSS

Company

- Blog
- Customer Stories
- Careers
- Company
- Terms of Service
- Privacy Policy
- Acceptable Use Policy
- Service Level Agreement
- Humans.txt
- Lawyers.txt

Security.txt