Back

Blog post

What's New in pg_graphql v1.2

2023-04-21 • 6 minute read

Oliver Rice
Engineering

It's been 4 months since the 1.0.0 release of pg_graphql. Since then, we've pushed several features to improve the APIs that `pg_graphql` produces.

In this article, we'll walk through those features and show examples of each.

> 📢 *These features are only available on projects with Postgres version `15.1.0.63` or higher. For help with upgrading, please review the migrating and upgrading projects guide.*

View Support

Prior to `v1.1`, `pg_graphql` would only reflect standard tables. Since then, views, materialized views, and foreign tables are now also reflected in the GraphQL schema.

For example:

```
create view "ProjectOwner" as
  select
    acc.id,
    acc.name
  from
    account as acc
    join role as r on r.id = acc.role_id
  where acc.role = 'project_owner';
```

Since all entities exposed by `pg_graphql` require primary keys, we must define that constraint for the view. We do that using a comment directive:

```
comment on view "ProjectOwner"
  is '@graphql({"primary_key_columns": ["id"]})
```

Which yields the GraphQL type:

```
type ProjectOwner implements Node {
  nodeId: ID!
  id: UUID!
```

```
    name: String
  }
```

With associated `Edge` and `Connection` types. That enables querying via:

```
  {
    projectOwnerCollection(first: 2) {
      edges {
        node {
          nodeId
          name
        }
      }
    }
  }
```

Additionally, simple views automatically support mutation events like inserts and updates. You might use these to migrate underlying tables while maintaining backwards compatibility with previous API versions.

Filtering

Filtering in SQL is endlessly flexible. We've taken two incremental steps to bring more of that flexibility to the GraphQL interface.

`is null` and `is not null`

Handling `null` values can be tricky in both SQL and GraphQL. However, there are similarities we can take advantage of. In `pg_graphql`, every scalar data type has its own filter type, such as `IntFilter` and `StringFilter`. Each of these filter types now includes an `is` argument, which allows you to filter based on whether a value is null or not null. You can do this by using `{is: NULL}` for `null` values and `{is: NOT_NULL}` for non-`null` values.

```
enum FilterIs {
    NULL
    NOT_NULL
}

type IntFilter {
    ...
    is: FilterIs
}
```

For example:

```
{
  blogCollection(filter: { name: {is: NULL}}) {
    ...
  }
}
```

to return all `blog`s where the `name` is `null`.

`like`, `ilike`, and `startsWith`

Text filtering options in `pg_graphql` have historically been restricted to equality checks. The hesitation was due to concerns about exposing a default filter that is difficult to index. The combination of citext and PGroonga available on the platform solves those scalability risks and enabled us to expand the `StringFilter` with options for `like` `ilike` and `startsWith`.

```
input StringFilter {
  eq: String
  ...
  startsWith: String
  like: String
  ilike: String
}
```

Note that `startsWith` filters should be preferred where appropriate because they can leverage simple B-

Tree indexes to improve performance.

```
{
  generalLedgerCollection(filter: { identifier(
    edges {
      node {
        nodeId
        identifierCode
        amount
      }
    }
  }
}
```

GraphQL directives `@skip` and `@include`

The GraphQL spec has evolved over time. Although the spec is clear, it is common for GraphQL servers to selectively omit some chunks of functionality. For example, some frameworks intentionally do not expose an introspection schema as a form of security through obscurity.

`pg_graphql` aims to be unopinionated and adhere exactly to the spec. The `@skip` and `@include` directives are part of the GraphQL core specification and are now functional.

The `@skip` directive in GraphQL is used to conditionally skip a field or fragment during query execution based on a Boolean variable. It can be used to make the query more efficient by reducing the amount of data retrieved from the server.

The `@include` directive is the mirror of `@skip` where a field or fragment is conditionally included depending on the value of a Boolean variable.

Here's an example of how the `@skip` directive can be used in a GraphQL query:

```
query getBooks($includeDetails: Boolean!) {
  booksCollection {
    edges {
      node {
        id
        title
        description @skip(if: $includeDetails)
      }
    }
  }
}
```

User Defined Descriptions

Users can now use the comment directive system to assign descriptions to tables, views and columns.

```
create table public.book(
    id int primary key,
    title text not null
);

comment on table public.book
is e'@graphql({"description": "a library book"}

comment on column public.book.title
is e'@graphql({"description": "the title of the
```

GraphQL IDEs, such as GraphiQL render those descriptions, allowing developers to provide clearer API documentation.

Roadmap

The headline features we aim to launch in coming releases of `pg_graphql` include:

1. Support for user-defined functions: GitHub issue
2. Support for nested inserts: GitHub issue
3. An alternative approach to computed relationships based on SQL functions returning `SET OF` rather than comment directives (compatible with PostgREST)

More pg_graphql

- Introducing pg_graphql: A GraphQL extension for PostgreSQL
- GraphQL is now available in Supabase
- pg_graphql v1.0

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

Launch Week 7 Hackathon Winners

24 April 2023

Next post

Supabase Studio 2.0: help when you need it most

14 April 2023

Related articles

Supabase Beta April 2023

Securing your Flutter apps with Multi-Factor Authentication

Next steps for Postgres pluggable storage

Launch Week 7 Hackathon Winners

What's New in pg_graphql v1.2

View all posts

Build in a weekend, scale to millions

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