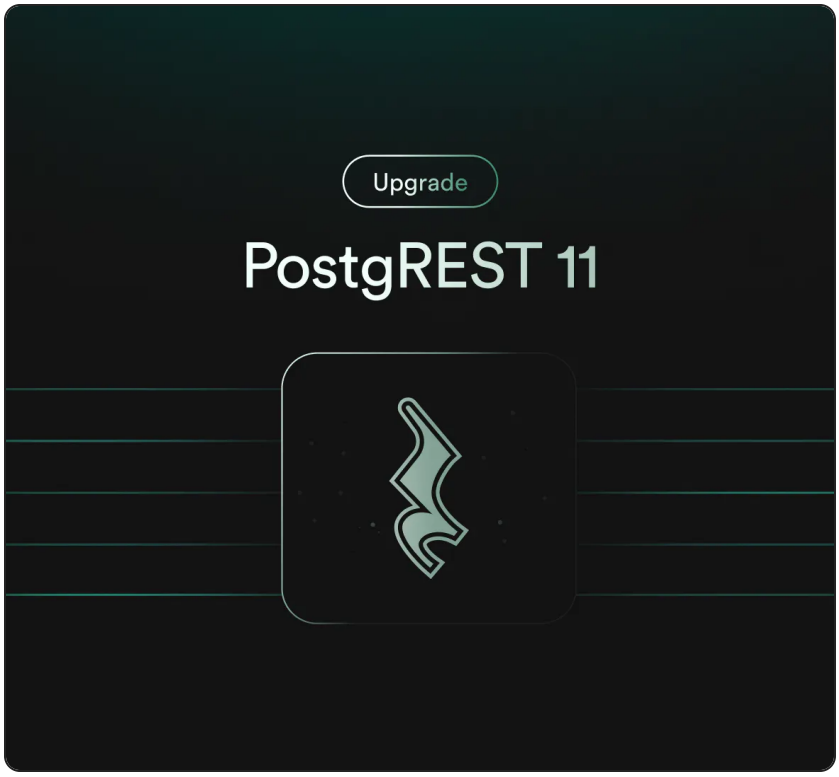
Blog post

PostgREST 11 pre-release

2022-12-16 • 5 minute read

Steve Chavez
Engineering & PostgREST maintainer



PostgREST 11 is not wrapped up yet, however a pre-release with the latest features and fixes is available on the Supabase CLI.

In this blog post we'll cover some of the improved querying capabilities: spreading related tables, related orders and anti-joins.

Spreading related tables

Very often the way we structure a database is not the way we want to present it to the frontend application. For example, let's assume we have a `films` and `technical_specs` tables and they form a one-to-one relationship.

Using PostgREST resource embedding, we can query them in one request like so

From HTTP:

```
GET /films?select=title,technical_specs(camera,
```

or JavaScript:

```
const { data, error } = await supabase.from('fi
    title,
    technical_specs (
      camera, laboratory, duration
    )
  `)
```

Response:

```
[
  {
    "title": "Pulp Fiction",
    "technical_specs": {
      "camera": "Arriflex 35-III",
      "laboratory": "DeLuxe, Hollywood (CA), US
      "duration": "02:34:00"
    }
  },
  "..."
]
```

But we'd like to present a “flattened” result to the frontend, without the `technical_specs` object. For this we could create a new database view or function that shapes the json the way we want, but creating extra database objects is not always convenient.

Using the new “spread” operator(syntax borrowed from JS), we can expand a related table columns and remove the nested object.

From HTTP:

```
GET /films?select=title,...technical_specs(came
```

or JavaScript:

```
const { data, error } = await supabase.from('fi
    title,
    ...technical_specs (
      camera, laboratory, duration
    )
  `)
```

Response:

```
[
  {
    "title": "Pulp Fiction",
    "camera": "Arriflex 35-III",
    "laboratory": "DeLuxe, Hollywood (CA), USA
    "duration": "02:34:00"
  },
  "..."
]
```

This only works for one-to-one and many-to-one relationships for now but we're looking at ways to remove this restriction.

Order by related tables

It's also a common use case to order a table by a related table column. For example, suppose you'd like to order `films` based on the `technical_specs.duration` column.

You can now do it like so:

From HTTP:

```
GET /films?select=title,...technical_specs(dura
```

or JavaScript:

```
const { data, error } = await supabase
  .from('films')
  .select(`
    title,
    ...technical_specs (
      duration
    )
  `)
   .order('technical_specs(duration)', { descer
```

Response:

```
[
  {
    "title": "Amra Ekta Cinema Banabo",
    "duration": "21:05:00"
  },
  {
    "title": "Resan",
    "duration": "14:33:00"
  },
  "..."
]
```

Similarly to spreading related tables, this only works for one-to-one and many-to-one relationships.

Anti-Joins

To do the equivalent of a left anti-join, you can now filter the rows where the related table is `null`.

From HTTP:

```
GET /films?select=title,nominations()&nominatic
```

or JavaScript:

```
const { data, error } = await supabase
  .from('films')
  .select(`
```

```
    title,
    nominations()
  `)
  .is('nominations', null))
```

Response:

```
[
  {
    "title": "Memories of Murder"
  },
  {
    "title": "Rush"
  },
  {
    "title": "Groundhog Day"
  },
  "..."
]
```

Note that `nominations` doesn't select any columns so they don't show on the resulting response.

The equivalent of an inner join can be done by filtering the rows where the related table is `not null`.

```
GET /films?select=title,nominations(rank,...com
```

```
const { data, error } = await supabase
  .from('films')
  .select(
    `
    title,
    nominations(rank,...competitions(name))
  `
  )
  .not('nominations', 'is', null)
```

Response:

```
[
  {
    "title": "Pulp Fiction"
    "nominations": [
      {"rank": 1, "name": "Palme d'Or"},
      {"rank": 1, "name": "BAFTA Film Award"},
      {"..."}
```

```
    ]
  },
  "..."
]
```

This was already possible with the `!inner` modifier(introduced on PostgREST 9) but the `not null` filter is more flexible and can be used with an or filter to combine related tables' conditions.

Try it out

This pre-release is not deployed to Supabase cloud but you can try it out locally with the Supabase CLI.

```
$ supabase start
```

Please try it and report any bugs, suggestions or ideas!

More Launch Week 6

- Day 1: New Supabase Docs, built with Next.js
- Day 2: Supabase Storage v2: Image resizing and Smart CDN
- Day 3: Multi-factor Authentication via Row Level Security Enforcement
- Day 4: Supabase Wrappers, a Postgres FDW framework written in Rust
- Day 5: Supabase Vault is now in Beta
- Community Day
- Point in Time Recovery is now available
- Custom Domain Names are now available
- Wrap Up: everything we shipped

Share this article

Last post

Supabase Vault is now in Beta
16 December 2022

Next post

Point in Time Recovery is now available for Pro projects
16 December 2022

Related articles

- Supabase Vault is now in Beta
- PostgREST 11 pre-release
- Point in Time Recovery is now available for Pro projects
- pg_graphql v1.0
- What's new in Postgres 15?

View all posts

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
- SupaSquad
- DevTo
- RSS

Company

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

Security.txt