Postgres schema migrations using the expand/contract pattern

October 24, 2024

xata

Speaker

Xata

Andrew Farries
Staff Software Engineer
andrew.farries@xata.io

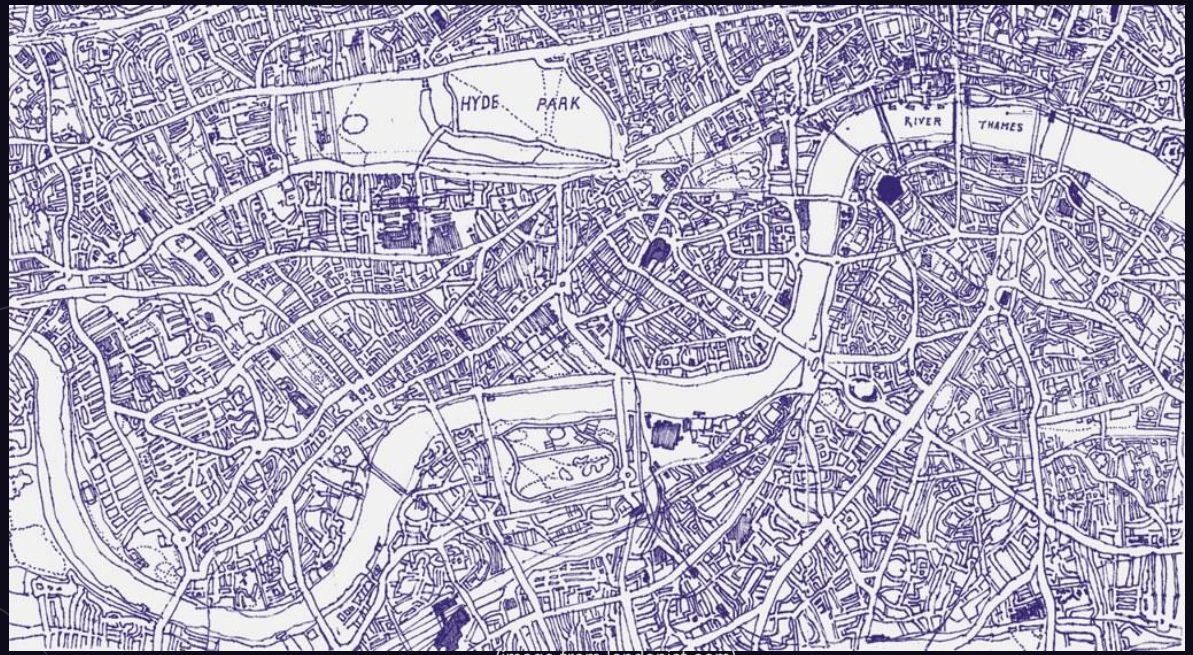
The plan

01 Common pitfalls

02 Tools and techniques

03 Expand contract migrations

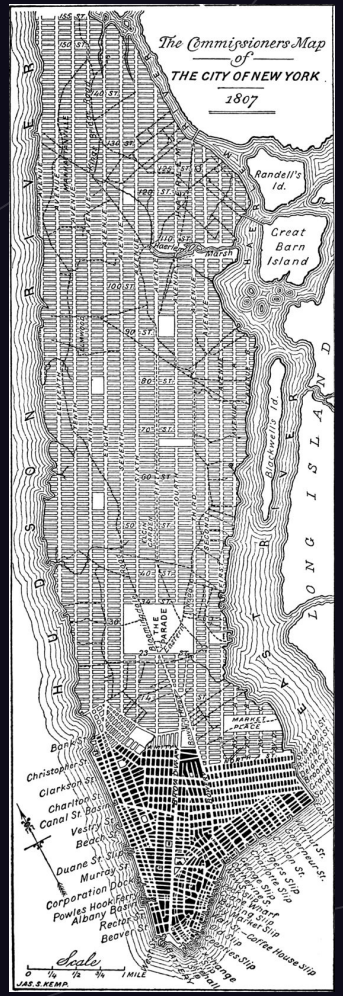
04 Expand/contract with pgroll



(image from londonist.com)

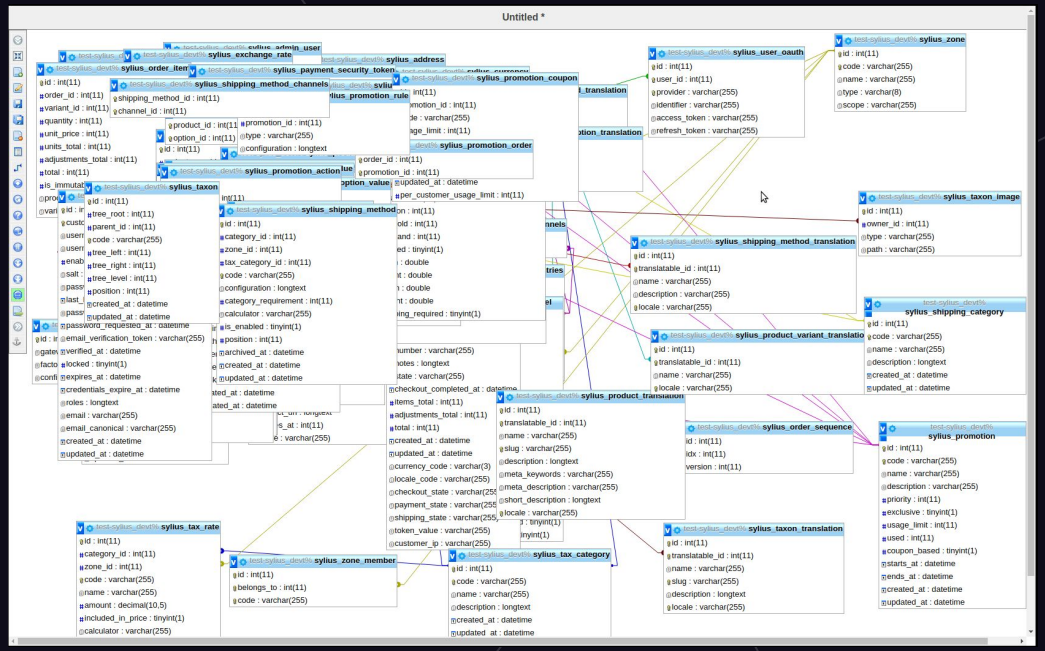
xata


HOT DOGES

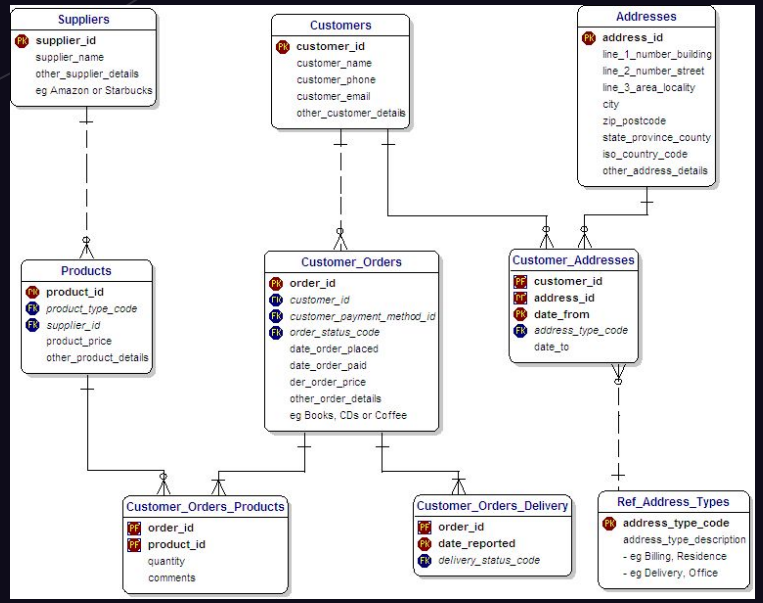

The Commissioners Map
of
THE CITY OF NEW YORK
1807
Randell's Id.
Great Barn Island
Blackwell's Id.
HUDSON RIVER
EAST RIVER
LONG ISLAND
THE BOWERY
MARKET PLACE
Bank St.
Christopher St.
Clarkson St.
Charlton St.
Canal St.
Vestry St.
Beach St.
Duane St.
Murray St.
Corporation Dock
Powles Hook Ferry
Albany Basin
Rector St.
Beaver St.
Cortlandt St.
Cortlandt Slip
Carter House Slip
Scale
0 1/4 1/2 3/4 1 MILE
JAS. S. KEMP.

xata

London



New York



“Database schemas are notoriously volatile, extremely concrete, and highly depended on. This is one reason why the interface between OO applications and databases is so difficult to manage, and why schema updates are generally painful.”

Robert C. Martin, Clean Architecture

Common pitfalls

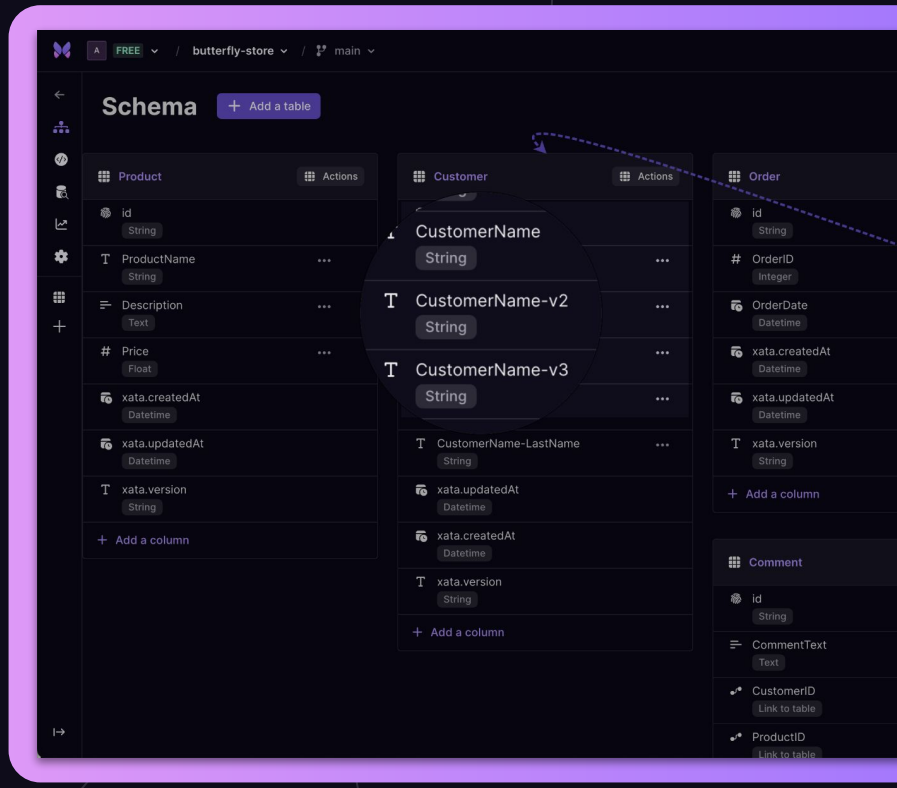
Additive-only changes and schema debt

Wait, what?

The act of never modifying or removing columns, only adding new ones

Why is this bad?

- Bugs and performance implications
- General confusion
- Long-living compatibility code



The locking minefield

Wait, what?

PostgreSQL offers good ways to control locking your database, but you need to know what you're doing

Why is this bad?

- Tables are inaccessible
- Query queuing
- Testing is hard

Non-conflicting lock modes can be held concurrently by many transactions. Notice i t some lock modes are self-conflicting (for exa
one transaction at a time) while others are not self-conflicting (for example, an ACCE can be held by multiple transactions).

Table-Level Lock Modes

`ACCESS SHARE (AccessShareLock)`

Conflicts with the `ACCESS EXCLUSIVE` lock mode only.

The `SELECT` command acquires a lock of this mode on referenced tables. In general, any query that only *reads* a table and does not mo

`ROW SHARE (RowShareLock)`

Conflicts with the `EXCLUSIVE` and `ACCESS EXCLUSIVE` lock modes.

The `SELECT` command acquires a lock of this mode on all tables on which one of the `FOR UPDATE`, `FOR NO KEY UPDATE`, `FOR SHARE`, `SHARE` locks on any other tables that are referenced without any explicit `FOR ...` locking option).

`ROW EXCLUSIVE (RowExclusiveLock)`

Conflicts with the `SHARE`, `SHARE ROW EXCLUSIVE`, `EXCLUSIVE`, and `ACCESS EXCLUSIVE` lock modes.

The commands `UPDATE`, `DELETE`, `INSERT`, and `MERGE` acquire this lock mode on the target table (in addition to `ACCESS SHARE` locks on acquired by any command that *modifies data* in a table.

`SHARE UPDATE EXCLUSIVE (ShareUpdateExclusiveLock)`

Conflicts with the `SHARE UPDATE EXCLUSIVE`, `SHARE`, `SHARE ROW EXCLUSIVE`, `EXCLUSIVE`, and `ACCESS EXCLUSIVE` lock modes. Thi `VACUUM` runs.

Acquired by `VACUUM` (without `FULL`), `ANALYZE`, `CREATE INDEX CONCURRENTLY`, `CREATE STATISTICS`, `COMMENT ON`, `REINDEX CONCUR` (for full details see the documentation of these commands).

`SHARE (ShareLock)`

Conflicts with the `ROW EXCLUSIVE`, `SHARE UPDATE EXCLUSIVE`, `SHARE ROW EXCLUSIVE`, `EXCLUSIVE`, and `ACCESS EXCLUSIVE` lock changes.

Acquired by `CREATE INDEX` (without `CONCURRENTLY`).

`SHARE ROW EXCLUSIVE (ShareRowExclusiveLock)`

Conflicts with the `ROW EXCLUSIVE`, `SHARE UPDATE EXCLUSIVE`, `SHARE`, `SHARE ROW EXCLUSIVE`, `EXCLUSIVE`, and `ACCESS EXCLUSIV` data changes, and is self-exclusive so that only one session can hold it at a time.

Acquired by `CREATE TRIGGER` and some forms of `ALTER TABLE`.

Testing schema migrations

Wait, what?

Confidence that a migration will succeed in production is hard to obtain with limited data available in lower environments.

Why is this bad?

- Failures only detected in production
- Problems provisioning realistic data-sets
- Lack of confidence in migrations

Rolling back your changes

Wait, what?

We're all human, mistakes occur. Having to roll back the changes you made in production can be painful

Why is this bad?

- Unplanned maintenance
- Likely untested
- Time consuming

Production rollout strategies

Version controlled SQL scripts

New migration scri...Works2012 (sa (55))* | SQLQuery1.sql - SE...Works2012 (sa (54))* | SQL Source Control

Name: Covers changes to: SelectAllProducts. Script created at 2017-01-06 06:41.

Save and close

```
/*
This migration script replaces uncommitted changes made to these objects:
SelectAllProducts

Use this script to make necessary schema and data changes for these objects only. Schema changes to any other objects won't be deplo

Schema changes and migration scripts are deployed in the order they're committed.
*/

SET NUMERIC_ROUNDABORT OFF
GO
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, ARITHABORT, QUOTED_IDENTIFIER, ANSI_NULLS ON
GO
PRINT N'Disabling DDL triggers'
GO
DISABLE TRIGGER ALL ON DATABASE
GO
PRINT N'Altering [dbo].[SelectAllProducts]'
GO
ALTER procedure [dbo].[SelectAllProducts]
AS
SELECT * FROM Production.Products

PRINT 'This is change #4'
GO
PRINT N'Re-enabling DDL triggers'
GO
```

Frameworks & ORMs


```
python manage.py makemigrations
```

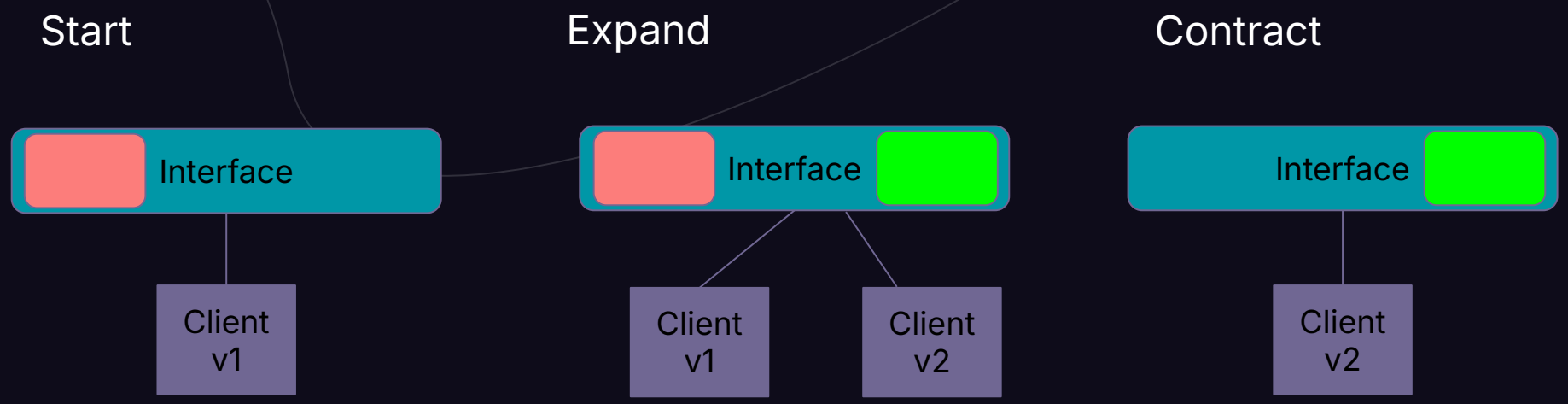

```
rails generate migration AddPartNumberToProduct
```


```
prisma migrate dev --name init
```


```
pnpm drizzle-kit generate:mysql
```

The expand / contract pattern

Expand & contract



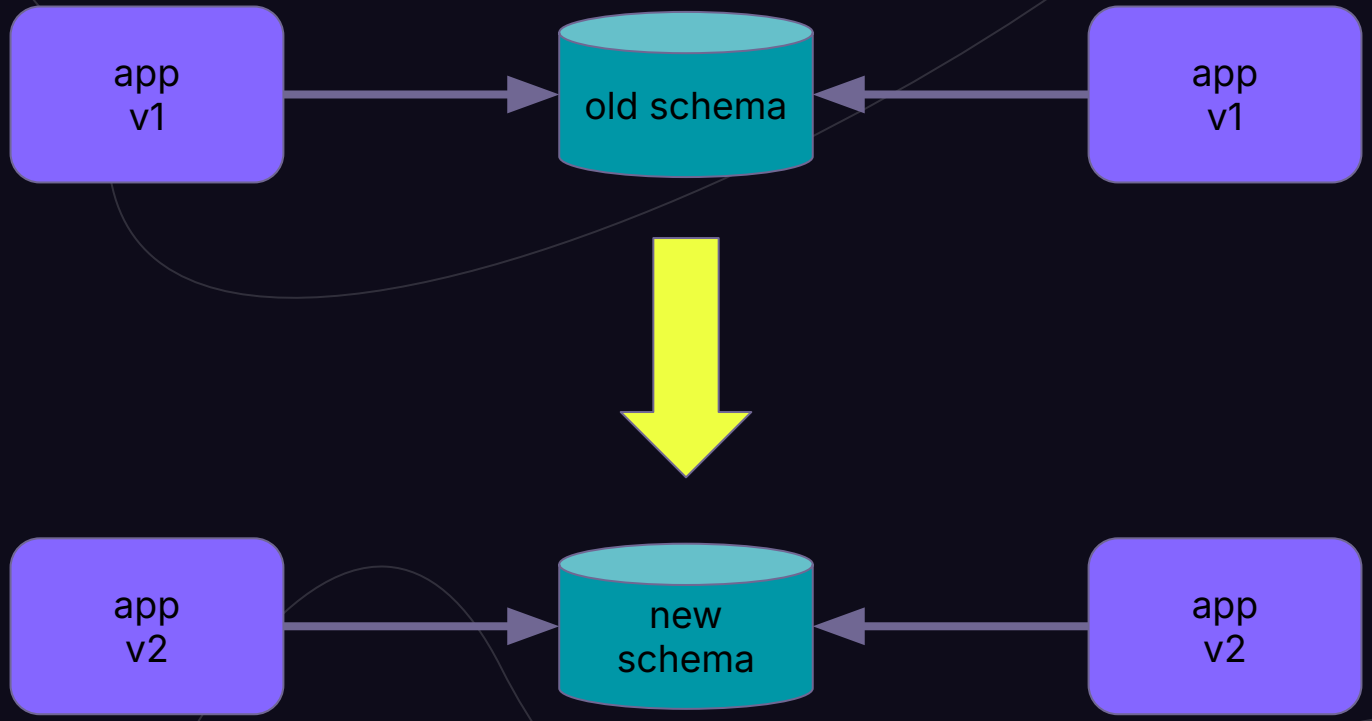
Expand & contract

orders	
order_id	varchar(255)
customer_id	varchar(255)
billing_address	text
shipped	boolean

orders	
order_id	varchar(255)
customer_id	varchar(255)
billing_address	text
status	order_status E

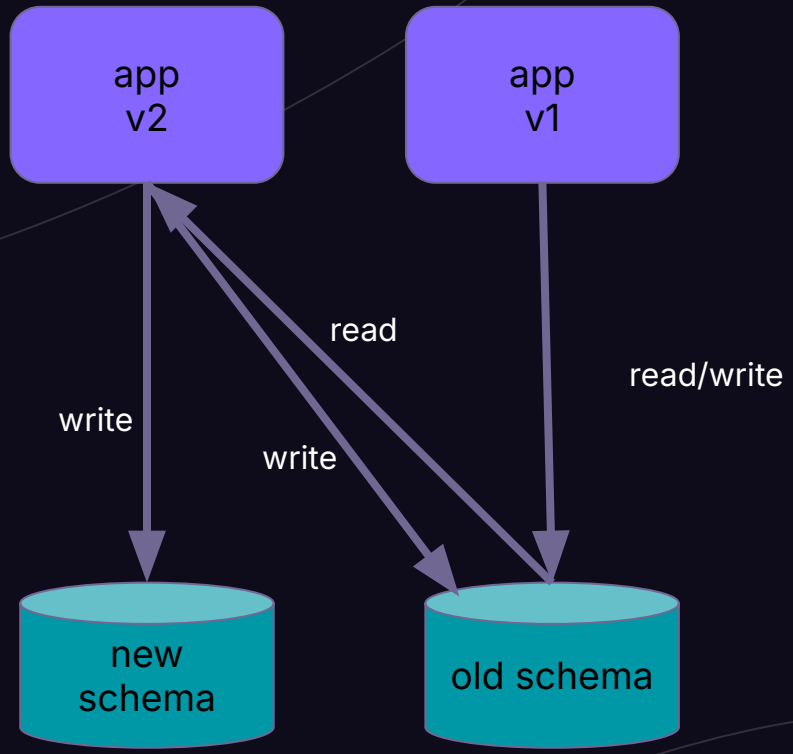
```
status order_status

ENUM order_status:
  pending
  shipped
  delivered
  cancelled
```

Big bang migration



expand/contract - dual write

id	customer_id	billing_address	shipped	status
3	5678	456 Somewhere Lane	False	<null>
4	1234	123 Somewhere Street	True	<null>

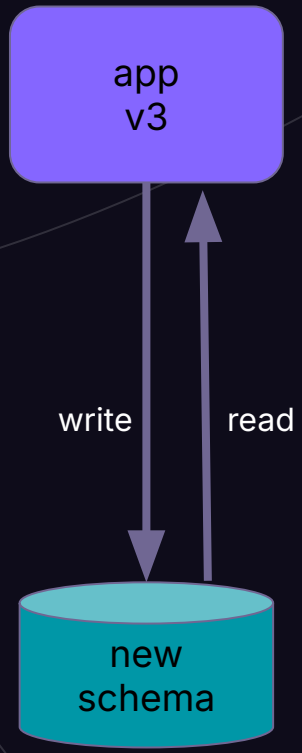


expand/contract - migrate

- Wait for the rollout of v2 to complete
- Run a data migration to backfill the `status` field

id	customer_id	billing_address	shipped	status
1	1234	123 Somewhere Street	True	shipped
2	5678	456 Somewhere Lane	False	pending

expand/contract - read new

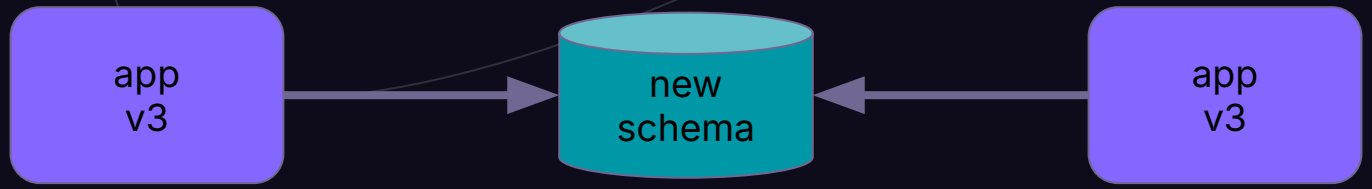


expand/contract - contract

- Once the rollout of v3 is complete, drop the `shipped` field
- The migration is complete

id	customer_id	billing_address	~~shipped~~	status
1	1234	123 Somewhere Street	~~True~~	shipped
2	5678	456 Somewhere Lane	~~False~~	pending

Expand / contract - complete



3 migrations required:

- Add the new field
- Backfill the new field
- Remove the old field

2 new application versions:

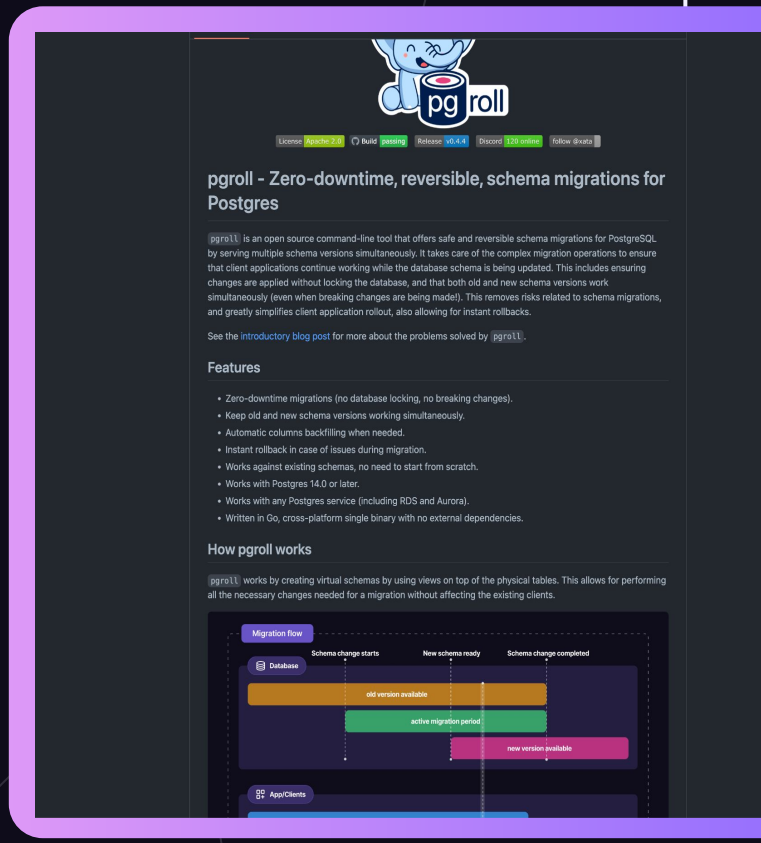
- Dual write
- Final version

Zero-downtime, reversible, schema migrations for Postgres

xata

pgroll - design goals

- Build around the expand/contract pattern
- Keep migration logic out of the application layer
- Easy rollbacks
- No nasty surprises around locking behaviour
- Postgres only
- Open source



License Apache 2.0 | Build passing | Release v0.4.4 | Discord 120 online | follow @xata

pgroll - Zero-downtime, reversible, schema migrations for Postgres

`pgroll` is an open source command-line tool that offers safe and reversible schema migrations for PostgreSQL by serving multiple schema versions simultaneously. It takes care of the complex migration operations to ensure that client applications continue working while the database schema is being updated. This includes ensuring changes are applied without locking the database, and that both old and new schema versions work simultaneously (even when breaking changes are being made!). This removes risks related to schema migrations, and greatly simplifies client application rollout, also allowing for instant rollbacks.

See the introductory blog post for more about the problems solved by `pgroll`.

Features

- Zero-downtime migrations (no database locking, no breaking changes).
- Keep old and new schema versions working simultaneously.
- Automatic columns backfilling when needed.
- Instant rollback in case of issues during migration.
- Works against existing schemas, no need to start from scratch.
- Works with Postgres 14.0 or later.
- Works with any Postgres service (including RDS and Aurora).
- Written in Go, cross-platform single binary with no external dependencies.

How pgroll works

`pgroll` works by creating virtual schemas by using views on top of the physical tables. This allows for performing all the necessary changes needed for a migration without affecting the existing clients.

Demo

Lesson learned

- Expand contract is a powerful technique for schema change
- Migration tools should operate at a higher level than raw SQL
- Migrations are long-lived processes and migration tools should manage them end to end
- Data migrations should be handled by migration tools, not at the application level

Thank you!

@xata

xataio/pgroll

@xata.io

xata.io/discord