

Databases schema management for lazybones: from chaos to heaven

Julien Riou

PostgreSQL Users Group Belgium

March 5, 2020

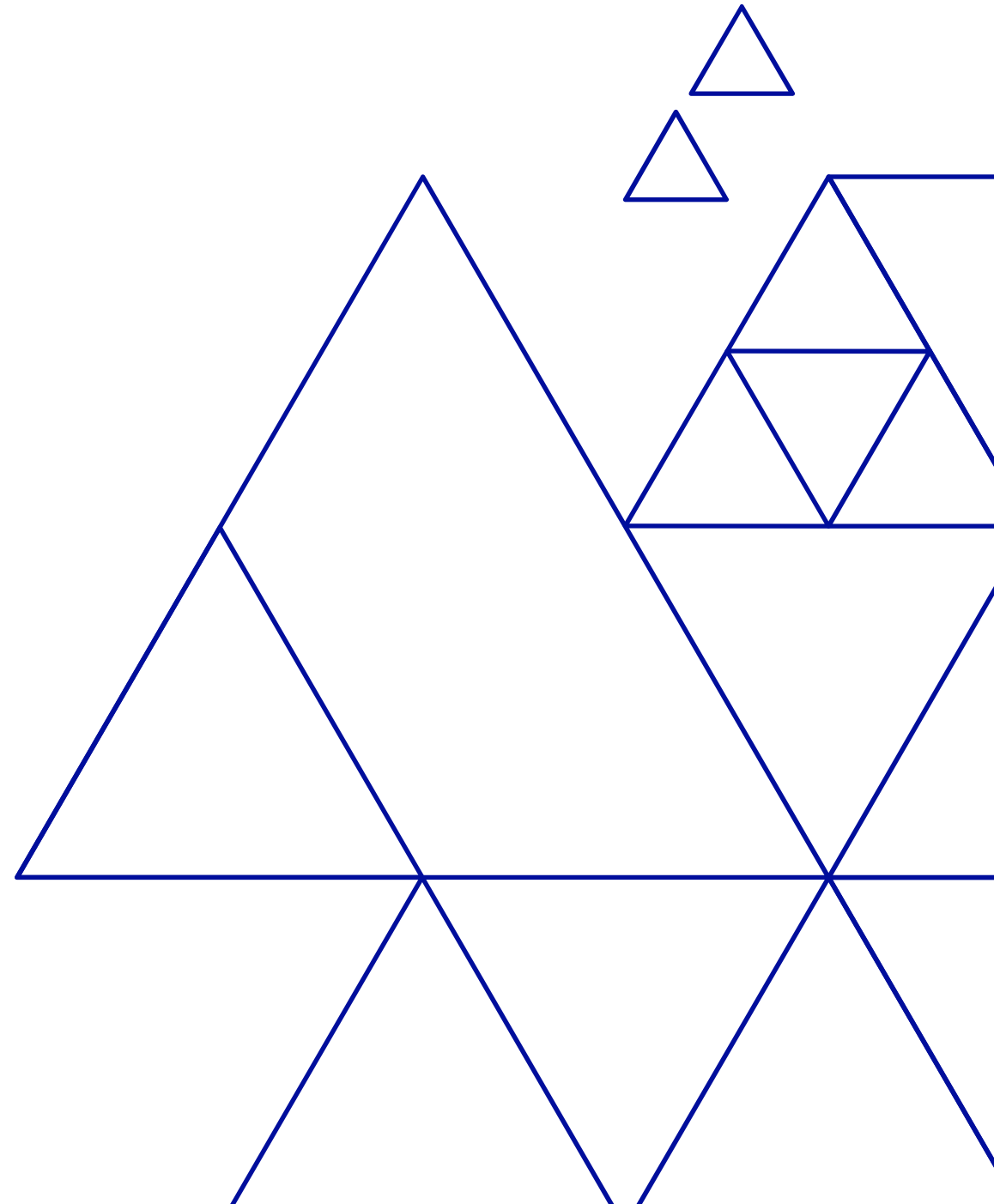
Speaker



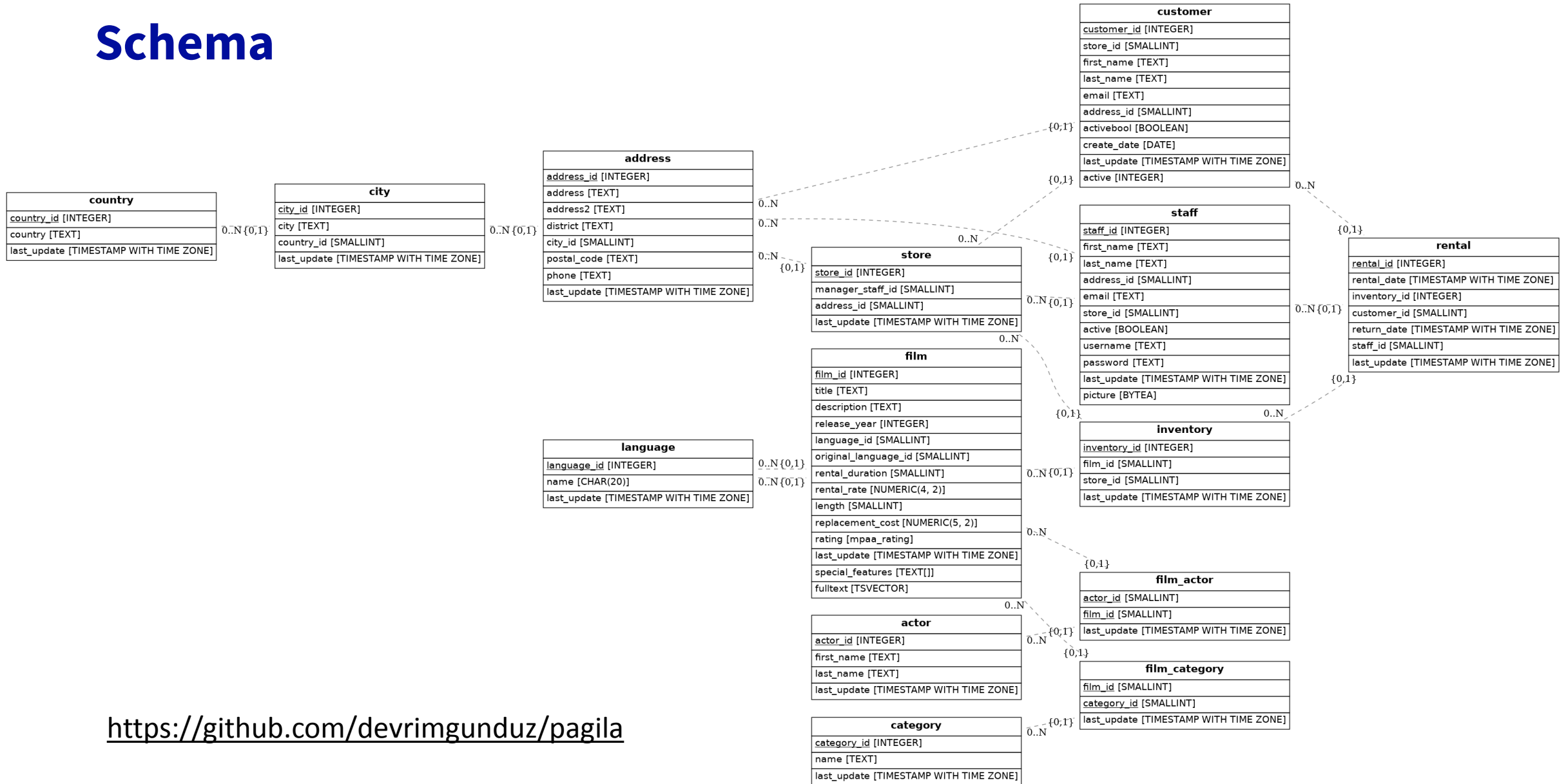
- ▶ Julien Riou
- ▶ DBA since 2012
- ▶ Tech lead in the databases team at OVHcloud since 2015
- ▶ Patroni contributor #10



Definitions



Schema

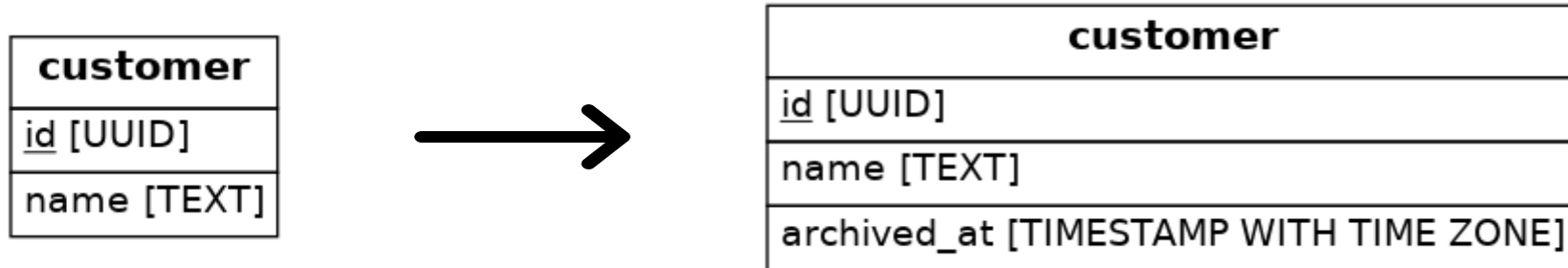


<https://github.com/devrimgunduz/pagila>

Schema changes over time

- ▶ New tables
- ▶ New columns
- ▶ New constraints
- ▶ New everything

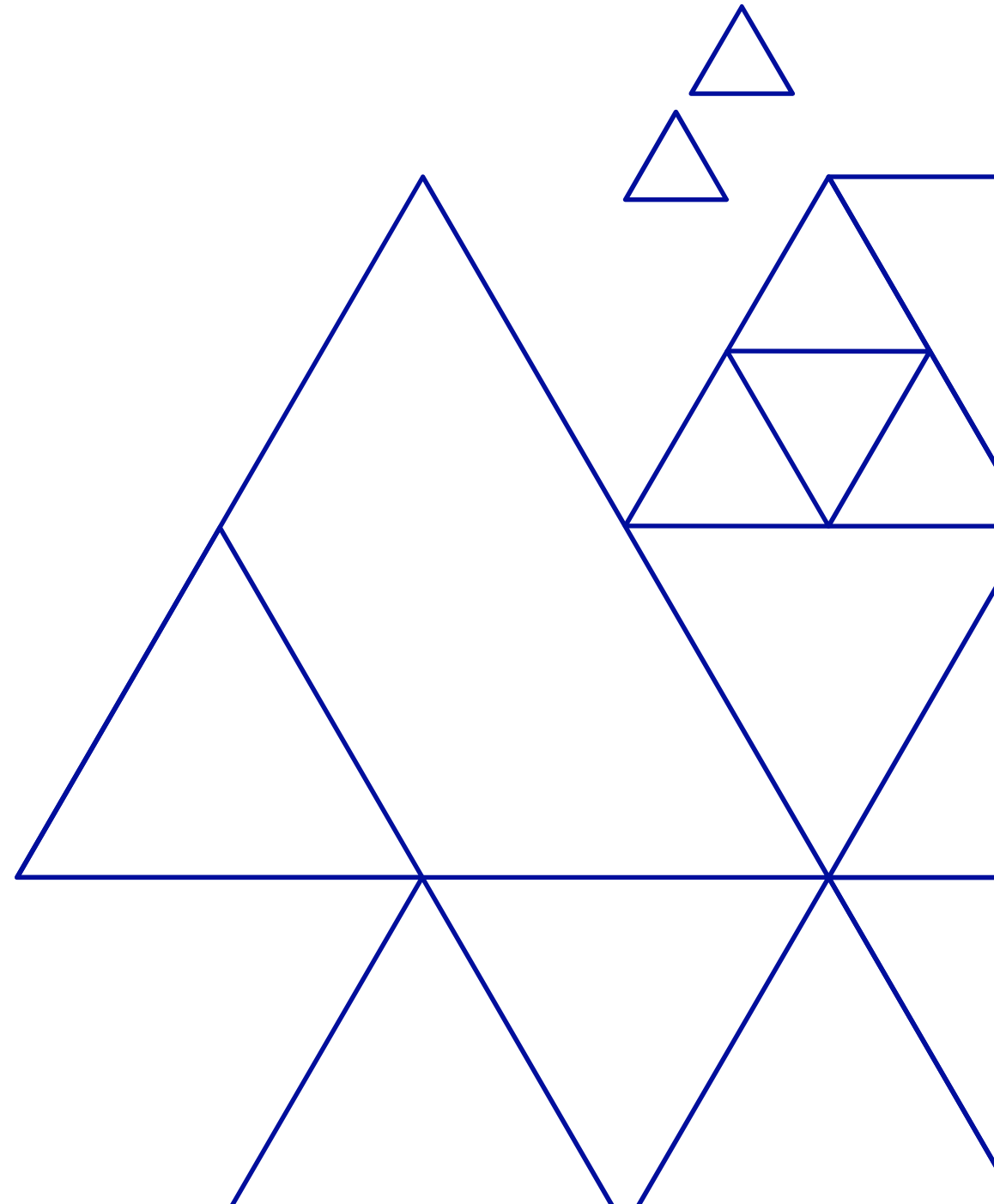
Schema migration



- ▶ Data definition language (DDL)

```
ALTER TABLE customer  
  ADD COLUMN archived_at TIMESTAMP WITH TIME ZONE;
```

Context





Products

▶ Baremetal

- Built at Croix, France
- Hosted around the World

▶ Cloud computing

- Public Cloud (Openstack)
- Private Cloud (VMware)

▶ Platform

- Kubernetes, databases, logs, metrics, Big data, AI, Machine Learning, ...

▶ Web hosting

- Domain names, websites, e-mail, SSL/CDN, ...

▶ Telecom

- xDSL, telephony, SMS, VDI, OverTheBox, ...

Perimeter

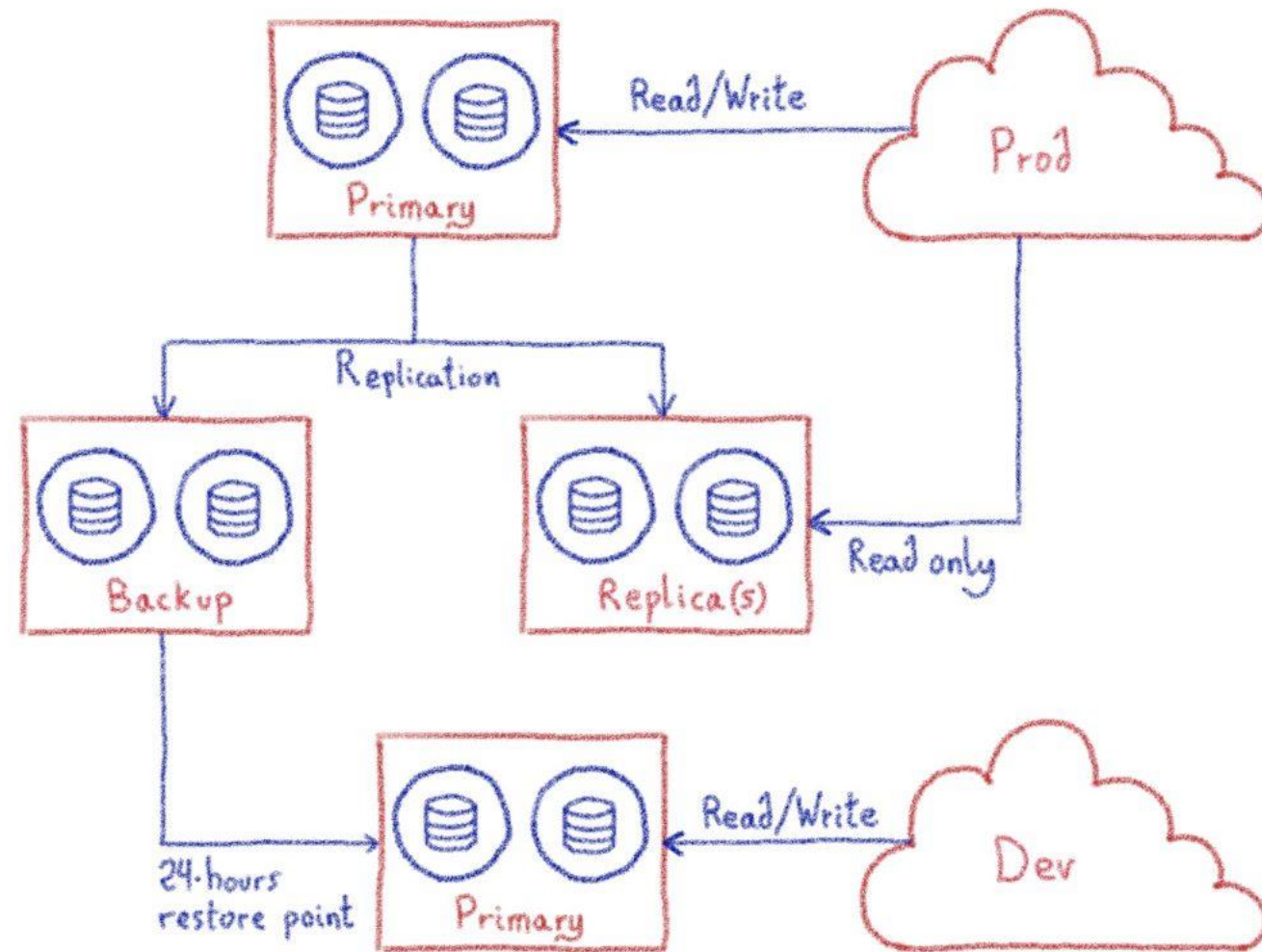
- ▶ 60 clusters
- ▶ 3000 applications
- ▶ 700 users
- ▶ 400 databases
- ▶ Worldwide

Internal databases

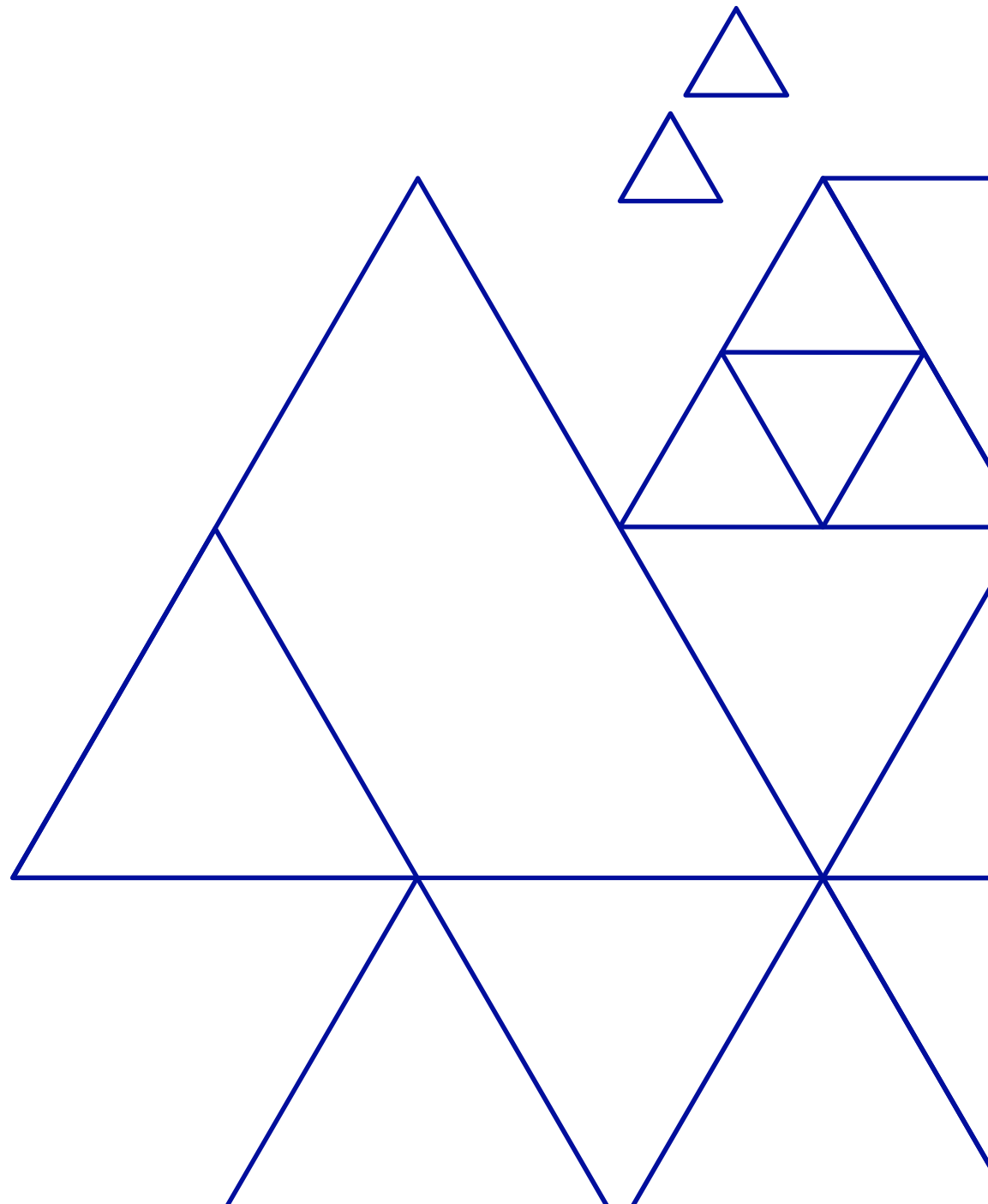


Internal cluster example

- ▶ MySQL
- ▶ PostgreSQL



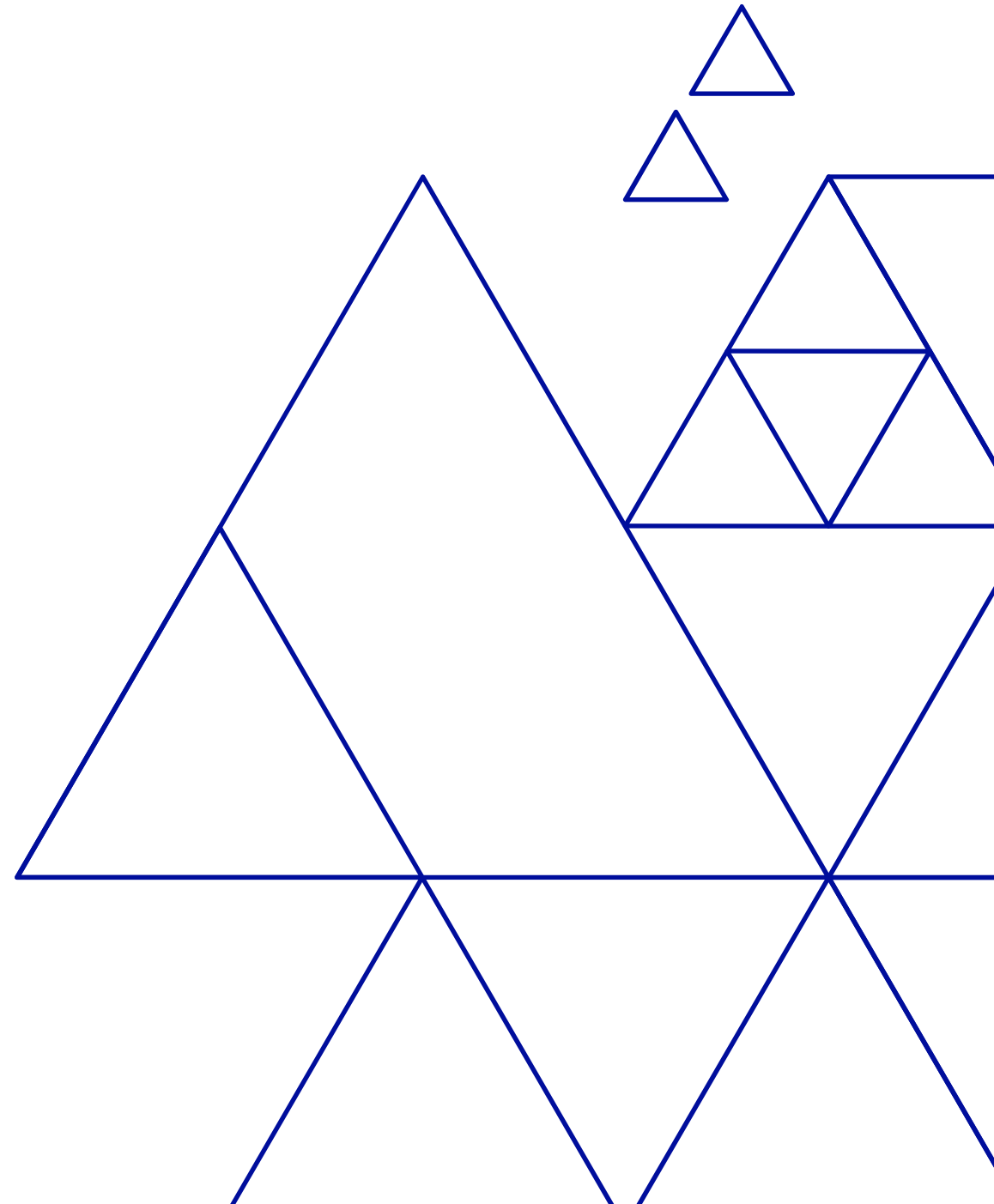
Overview



Overview

- ▶ Chaos
- ▶ Ticketing
- ▶ Schema as code
- ▶ CI/CD
- ▶ Inventory and automation
- ▶ Heaven
- ▶ What's next?

Chaos



Who can ALTER my database?



Startup mindset

- ▶ Employees are all in the same openspace
- ▶ Developer comes to the ops* desk
- ▶ *“Can you do X on my Y database, please?”*
- ▶ Ops connects to the production database host using SSH
- ▶ `psql Y`
- ▶ X
- ▶ Job done

**generally sysadmin, or DBA for lucky companies*

Startup mindset

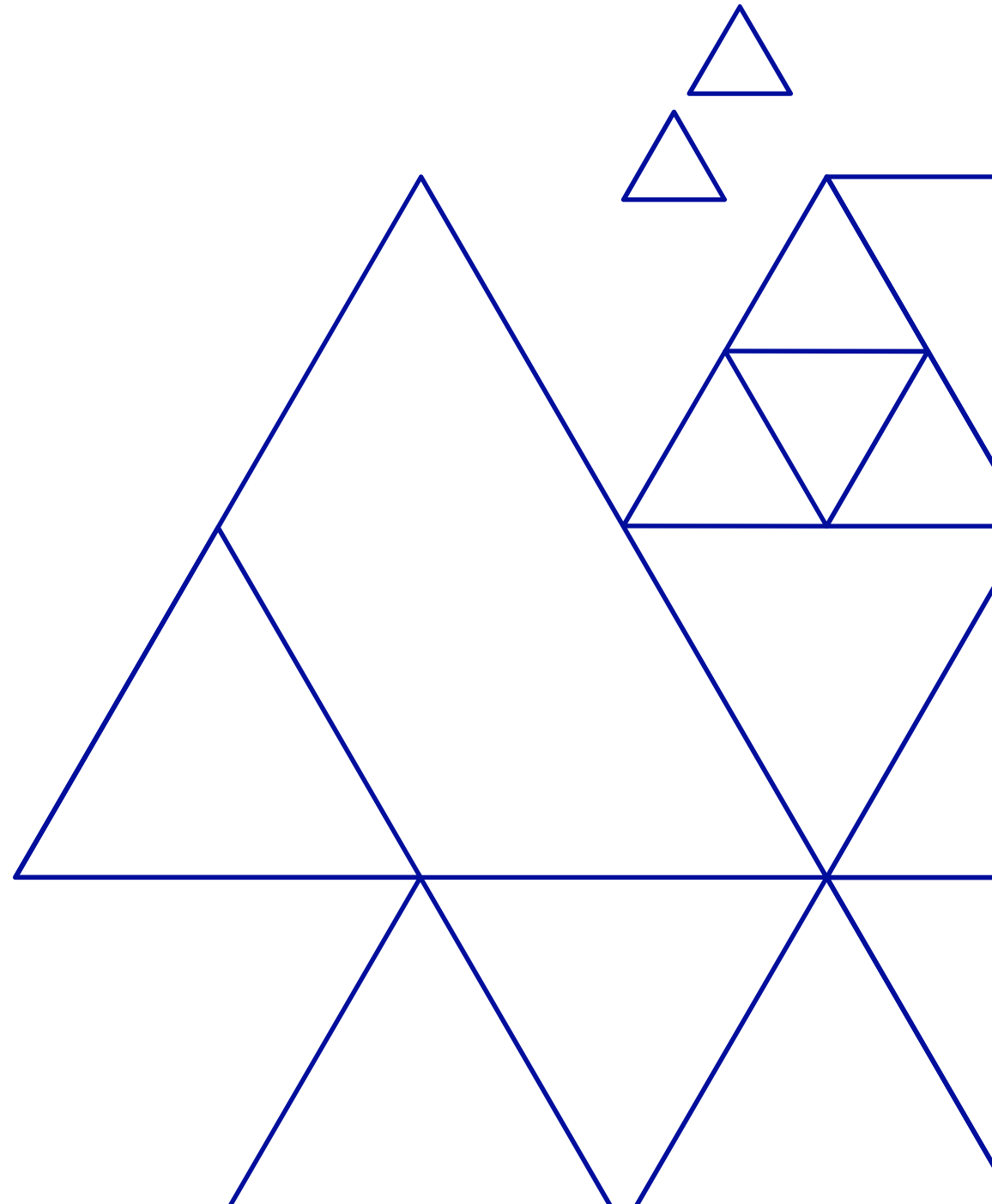


- ▶ Developer wanted to execute Z and not X
- ▶ Ops forgot where is the Y database
- ▶ Ops executed the change on the European database and forgot the Canadian one
- ▶ Ops executed statements outside of a transaction and it failed in the middle
- ▶ Ops started a transaction but forgot to commit and transaction is still opened
- ▶ The transaction is taking unexpected long time on production
- ▶ Ops didn't execute "SET ROLE" and now permissions are broken

Blame the ops!



Ticketing



Let's write migrations in a ticketing system

► JIRA

The screenshot shows a JIRA issue titled "New table for [redacted] on [redacted]". The issue is in a "CLOSED" state with a resolution of "Fixed". The details section shows it is an "Improvement" with "Minor" priority, related to the "DB" component. The description contains a request to create a table on PostgreSQL EU+CA, followed by a SQL migration script. The right sidebar shows the issue was created on 12/Oct/15 and updated on 27/Jun/16.

Details

Type:	<input checked="" type="checkbox"/> Improvement	Status:	CLOSED (View Workflow)
Priority:	↓ Minor	Resolution:	Fixed
Component/s:	DB		
Labels:	None		
Templates:			
Otrs linked Tickets number:	0		

Description

Hello,

Can you create this table on Postgres EU+CA.

DB schema

No need for archive table.

– MAIN TABLE

```
DROP TABLE IF EXISTS [redacted] ;
CREATE TABLE [redacted] (
  SERIAL PRIMARY KEY,
  INET NOT NULL,
  VARCHAR(64),
  VARCHAR(64),
  VARCHAR(255),
```

People

Assignee:

Reporter:

Votes: 0 [Vote for this issue](#)

Watchers: 2 [Start watching this issue](#)

Dates

Created: 12/Oct/15 14:35

Updated: 27/Jun/16 11:41

Resolved: 14/Oct/15 10:30

Development

[Create branch](#)

Agile

[View on Board](#)

Let's write migrations in a ticketing system

► OTRS

▼ Article #1 – provision POSTGRESQL db Created: 06/15/2016 17:56 (+1) by [redacted]

Mark | Print

From: [redacted]

Subject: provision POSTGRESQL db

To open links in the following article, you might need to press Ctrl or Cmd or Shift key while clicking the link (depending on your browser and OS). ✕

Hello, we need a new postgresql db named
Same characteristics as existing
With the following schema applied:
Thank you :)

```
BEGIN;

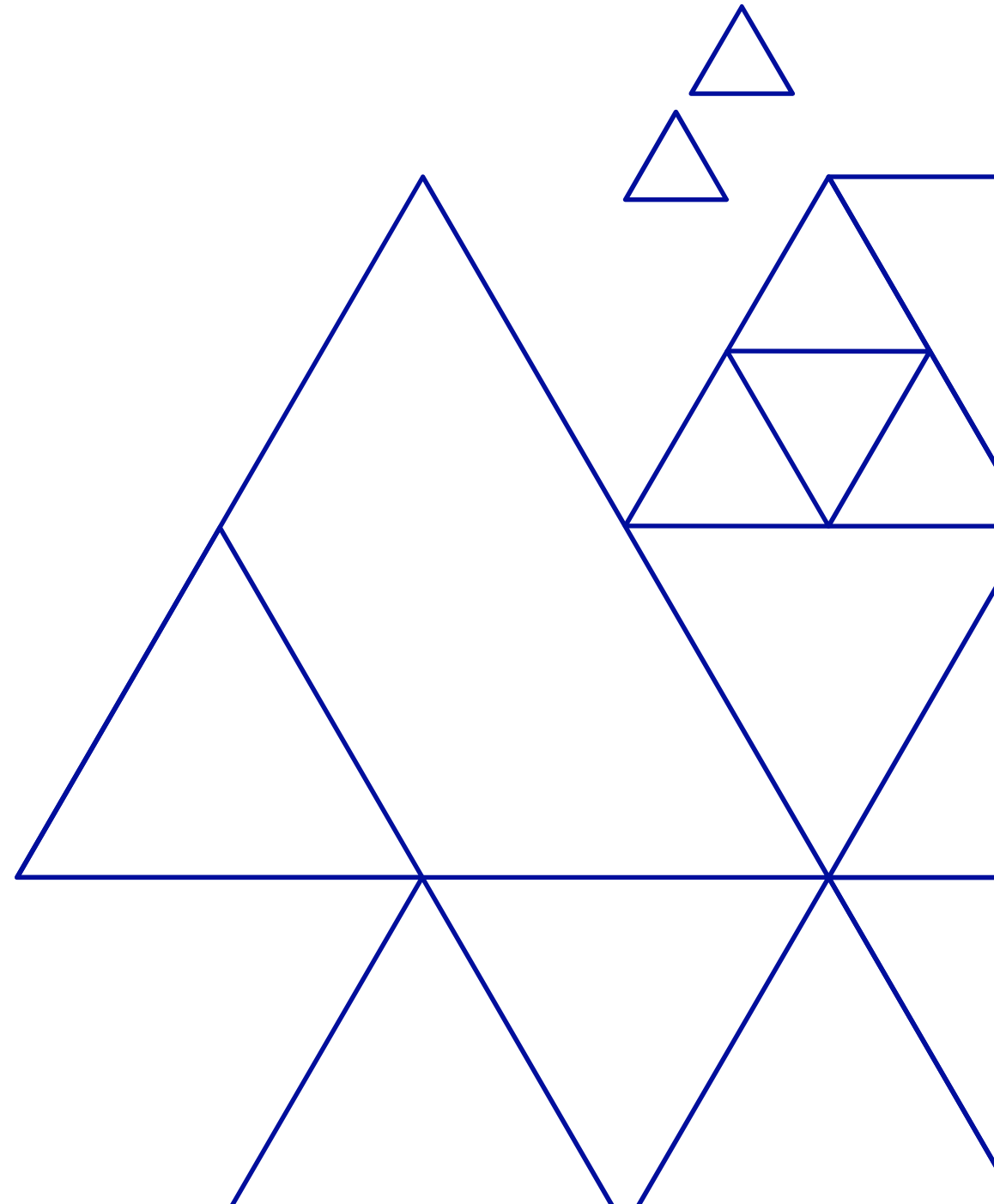
DROP TABLE IF EXISTS          CASCADE;
DROP TABLE IF EXISTS          CASCADE;
DROP TABLE IF EXISTS          CASCADE;
DROP TABLE IF EXISTS          CASCADE;
DROP TABLE IF EXISTS          CASCADE;
DROP TABLE IF EXISTS          CASCADE;

CREATE TABLE (
    BIGSERIAL PRIMARY KEY,
    TEXT UNIQUE NOT NULL,
    TEXT NOT NULL,
    JSON NOT NULL,
    JSON NOT NULL,
    JSON,
    JSON,
    BOOL NOT NULL DEFAULT false,
    BOOL NOT NULL DEFAULT false,
    TEXT NOT NULL
);
```

But

- ▶ Multiple ticketing systems for different teams
- ▶ SQL statements are not well formatted
- ▶ Doesn't prevent from bad copy pastes
- ▶ Poor reviewing system

Schema as code



Version control

- ▶ Git
- ▶ One repository for one or more databases
- ▶ Well known by developers

Schema migrations

sql-migrate

- ▶ “SQL schema migration tool for Go”
- ▶ <https://github.com/rubenv/sql-migrate>
- ▶ MIT license

Schema migrations

► Developers

```
-- +migrate Up  
  
create table x (...);  
  
-- +migrate Down  
  
drop table x;
```

Schema migrations

- ▶ Ops
- ▶ Wrapper created to handle “SET ROLE”
- ▶ Two migrations paths
 - “admin” (run as superuser) for create extensions
 - “normal” (run with DDL privileges) for the rest

```
hostname ~ $ sql-migrate-wrapper status -config database.yaml
INFO[2020-01-17T08:51:21+01:00] Processing 'database' database
INFO[2020-01-17T08:51:21+01:00] Processing admin migration path
+-----+-----+
| MIGRATION |                APPLIED                |
+-----+-----+
| v0001.sql | 2019-11-05 12:36:59.909699 +0100 CET |
+-----+-----+
INFO[2020-01-17T08:51:21+01:00] Processing normal migration path
+-----+-----+
| MIGRATION |                APPLIED                |
+-----+-----+
| v0001.sql | 2019-11-05 12:37:00.218529 +0100 CET |
| v0002.sql | 2019-11-19 15:04:22.464493 +0100 CET |
| v0003.sql | 2019-12-17 16:02:06.383822 +0100 CET |
+-----+-----+
```

Schema migrations

- ▶ One file for one migration
- ▶ One migration for one transaction

- ▶ Not transactional means two migration files
 - ALTER TYPE... ADD VALUE...
 - CREATE INDEX CONCURRENTLY...

- ▶ Focus on schema (DDL), not data (DML)
- ▶ ... except users, PostgreSQL schemas and databases

Code reviews



- ▶ Git based collaboration tool
- ▶ <https://bitbucket.org/>
- ▶ Not opensource

Code reviews with Bitbucket

The screenshot shows the Bitbucket web interface for a code review. The top navigation bar includes the Bitbucket logo, 'Projects', 'Repositories', and a search bar. The main content area is titled 'migration' and shows a diff view for a file named 'v0001.sql'. The diff is highlighted in green, indicating it is an addition. The code is a PostgreSQL migration script with the following content:

```
1 + -- Postgres schema
2 +
3 + -- +migrate Up
4 + CREATE TYPE [redacted] AS ENUM (
5 +
6 + CREATE TABLE [redacted] (
7 +     [redacted] UUID PRIMARY KEY,
8 +     [redacted] timestamp with time zone,
9 +     [redacted] timestamp with time zone,
10 + [redacted] timestamp with time zone,
11 + [redacted] timestamp with time zone,
12 + [redacted] bigint NOT NULL,
13 + [redacted] bigint NOT NULL,
14 + [redacted] bigint NOT NULL,
15 + [redacted] int NOT NULL,
16 + [redacted] int NOT NULL,
17 + [redacted] [redacted] NOT NULL DEFAULT [redacted],
18 + [redacted] jsonb NOT NULL,
19 + [redacted] bigint[] NOT NULL,
20 + [redacted] jsonb NOT NULL
21 + );
22 +
23 + CREATE INDEX [redacted] ON [redacted] ([redacted]);
24 + CREATE INDEX [redacted] ON [redacted] ([redacted]);
25 + CREATE INDEX [redacted] ON [redacted] ([redacted]);
26 +
27 + -- +migrate Down
28 + DROP INDEX
29 + [redacted];
30 + [redacted];
31 + [redacted];
32 + DROP TABLE [redacted];
33 + DROP TYPE [redacted];
```

Schema as code

- ▶ **Developer:** git clone, modifications, git commit, git push, create a pull request
- ▶ **Developer and ops:** code review
- ▶ **Ops:** merge
- ▶ **Ops:** connect to dev, git pull, sql-migrate
- ▶ **Ops:** connect to production, git pull, sql-migrate
- ▶ Job done

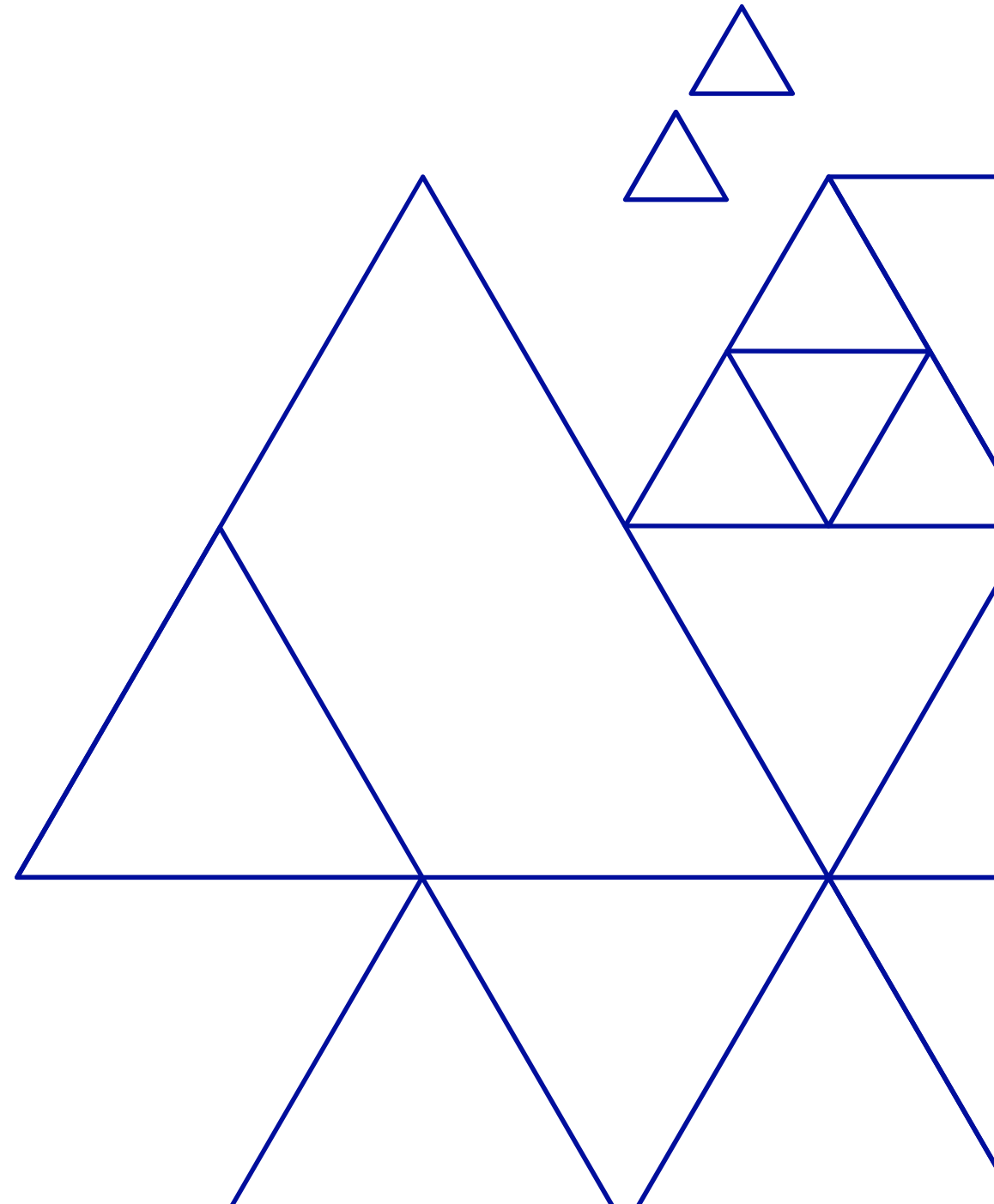
What's good

- ▶ No more copy/paste typos from the ops
- ▶ Developers are familiar with version control and code reviews
- ▶ Better change history
- ▶ Better reviewers system
- ▶ Better centralization (one tool)

What's bad

- ▶ Multiple migration paths are hard to manage
- ▶ Highly concurrent repositories conflicts a lot
- ▶ Focuses on “*how to go from version 1 to version 2?*” instead of “*what is the expected schema?*”
- ▶ Lots of errors seen at execution time

CI/CD



Continuous integration

- ▶ Test locally
- ▶ Ensure a software is deliverable
- ▶ Merge changes to the main branch

Continuous delivery

- ▶ Build release (without deployment)

Continuous deployment

- ▶ Build release and automatically deploy it
- ▶ No manual intervention

Continuous integration with CDS



- ▶ “CDS is an Enterprise-Grade Continuous Delivery & DevOps Automation Open Source Platform”
- ▶ <https://ovh.github.io/cds/>
- ▶ BSD 3-Clause License

Continuous integration with CDS

- ▶ **Project:** group all schema tests at the same place
- ▶ **Application:** one application per database schema
- ▶ **Pipeline:** set of sequential group of parallel jobs, one pipeline per DBMS version
- ▶ **Workflow:** link between applications and pipelines

Continuous integration with CDS



Continuous integration with CDS

▶ Offline tests

- Compare number of Up and Down migrations
- Detect patterns (DEFAULT NULL, IF EXISTS, timetz, OWNER TO)
- Detect groupable instructions

▶ Online tests

- Detect differences between migrations
- Detect patterns (missing primary keys, duplicate indexes and constraints)

▶ Artifacts

- Schema diagram with eralchemy

Continuous integration with CDS

The screenshot displays the CDS interface for a pipeline named 'sql-migrate-postgresql-9.6'. The pipeline is highlighted in green, indicating it is successful. The interface includes a sidebar with pipeline details, a main view showing the pipeline's progress through stages, and a terminal window at the bottom displaying job logs.

Pipeline Details (Sidebar):

- Project: 607
- Pipeline: sql-migrate-postgresql-9.6
- 4 Artifacts
- 1m22s
- 10/01/2020 12:40
- Stage 1: ✓
- Stage 2: ✓ ⚠️

Pipeline Progress (Main View):


- Stage 1: offline (6s)
- Stage 2: eralchemy (29s)
- Stage 2: online (1m1s)

Terminal Log (Information):

```
[2020-01-10T11:40:10] ✓ Job has been queued
[2020-01-10T11:40:12] Hatchery p191-prod-swarm-multiple starts spawn worker with model 16357...(EXTRA string=shared.infra/sql-
[2020-01-10T11:40:12] Hatchery p191-prod-swarm-multiple starts docker pull 16357...(EXTRA string=
[2020-01-10T11:40:13] Hatchery p191-prod-swarm-multiple docker pull 16357 done...(EXTRA string=
[2020-01-10T11:40:17] Hatchery p191-prod-swarm-multiple spawn worker 16357 successfully in swarmy-sql-migrate-wrapper-happy-
[2020-01-10T11:40:20] This worker swarmy-sql-migrate-wrapper-happy-and-loving-goodall was created to take this action
[2020-01-10T11:40:20] Job 7421868 was taken by worker swarmy-sql-migrate-wrapper-happy-and-loving-goodall
[2020-01-10T11:40:20] Worker swarmy-sql-migrate-wrapper-happy-and-loving-goodall version:0.43.1-l+sha.geafccf1.cds.11922 os:
[2020-01-10T11:40:25] ✓ Worker swarmy-sql-migrate-wrapper-happy-and-loving-goodall finished working on this job and took 5s
```

Continuous integration with CDS



🔄 1 build 



🔄 1 build failed 



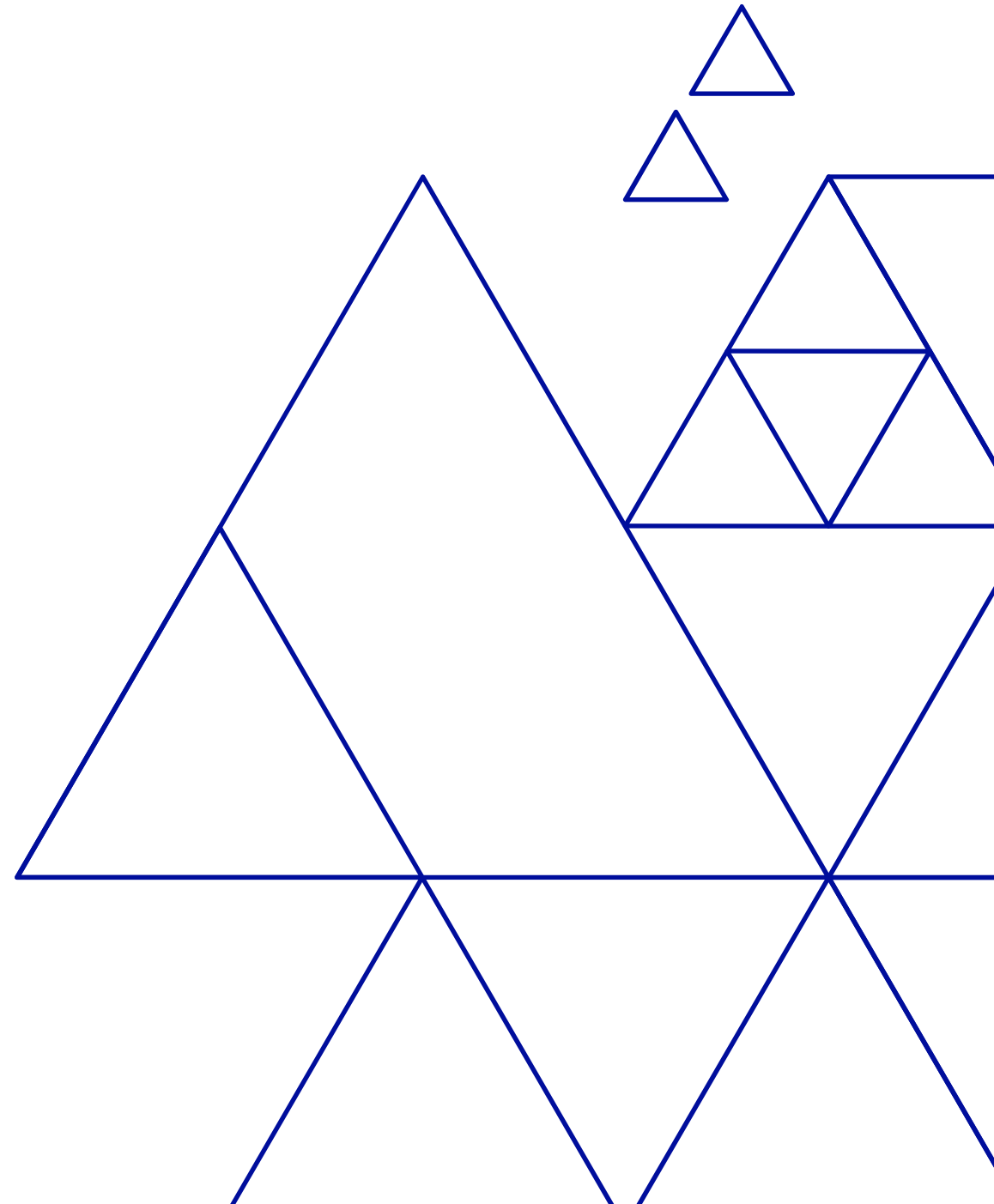
What's good

- ▶ Early testing
- ▶ Time and efforts saved

What's bad

- ▶ Under very active development
- ▶ Not enough trust to use continuous deployment feature on production databases

Inventory and automation



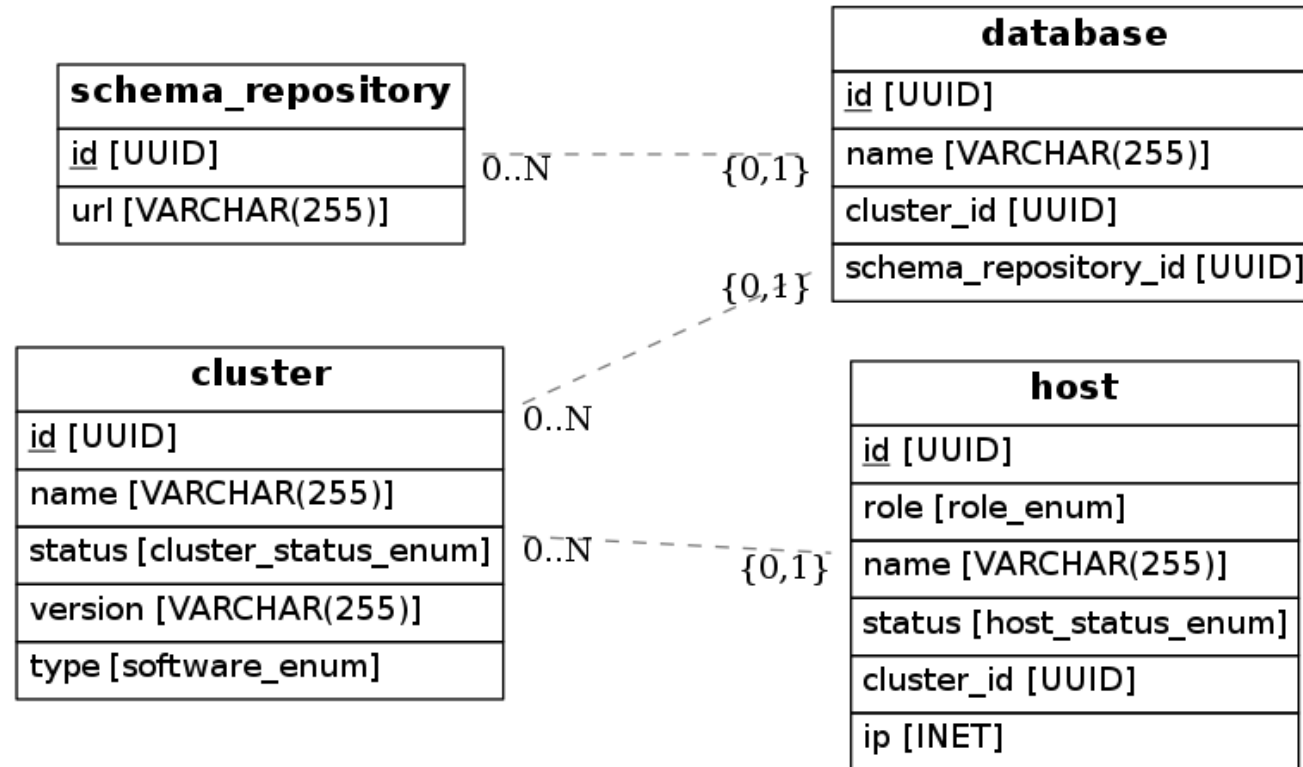
Inventory

- ▶ What databases depend on this git repository?
- ▶ Where is my database?



Inventory

► In a database of databases!



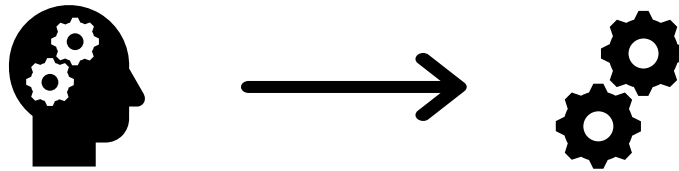
Inventory

► With an HTTP API

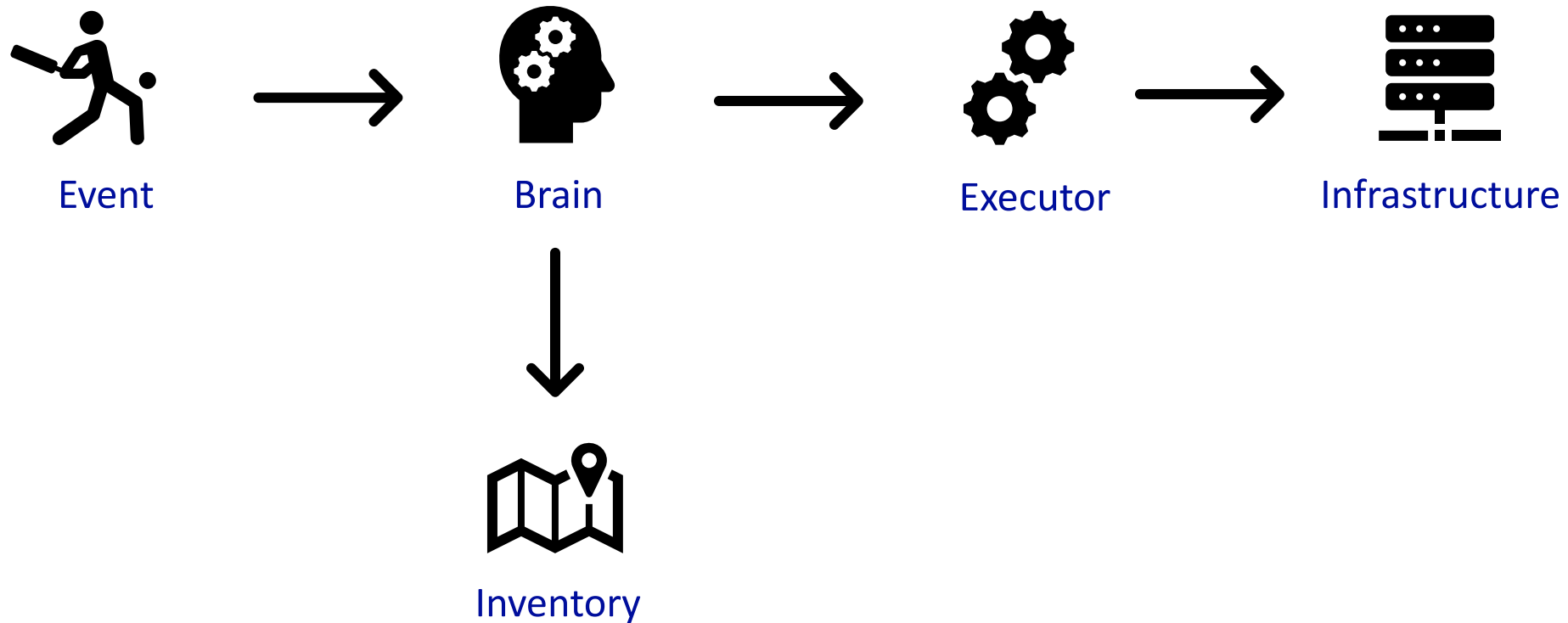
```
curl https://<api>/database?name=test
{
  "code": 200,
  "data": [
    {
      "schemarepository": "<uuid>",
      "name": "test",
      "id": "<uuid>",
      "cluster": "<uuid>"
    }
  ]
}
```

Automation

- ▶ How migrations can be automatically applied?



Automation



Brain

► Business logic

- Look for databases, clusters, hosts for IP addresses
- Ask “**Executor**” to run the schema migration

► Written in Python

- HTTP API with **Flask**
- Job scheduling with **Celery**
- UI with **Celery Flower**

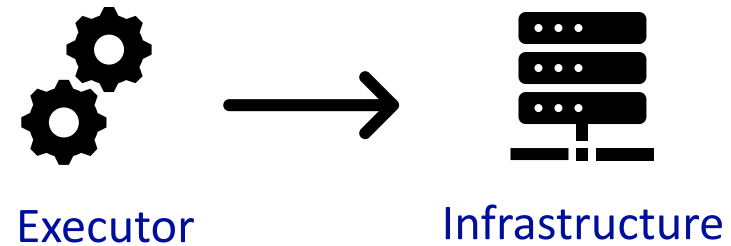
The screenshot shows the Celery Flower dashboard. At the top, there is a navigation bar with 'Celery Flower' and links for 'Dashboard', 'Tasks', 'Broker', and 'Monitor'. On the right side of the navigation bar are links for 'Logout', 'Docs', and 'Code'. Below the navigation bar, there are five summary boxes: 'Active: 0', 'Processed: 91', 'Failed: 1', 'Succeeded: 90', and 'Retried: 0'. Below these boxes is a 'Shut Down' button and a search input field. The main content area contains a table with the following data:

Worker Name	Status	Active	Processed	Failed	Succeeded	Retried	Load Average
celery@390d4e594c1d	Online	0	91	1	90	0	0.74, 0.63, 0.57

Showing 1 to 1 of 1 entries

Executor

- ▶ Next to the infrastructure
- ▶ Same stack as “**Brain**”
- ▶ Execute Ansible playbooks



Ansible playbook example

```
---
- name: update database {{ database }} schema
  hosts: all
  gather_facts: false
  pre_tasks:
    - name: "import gather_host"
      import_tasks: ../common/tasks/gather_host.yml
  tasks:
    - name: update schema
      when: role == "master" and branch == "master" or role == "dev"
      block:
        - name: create configuration
          template:
            src: ./database.yml.j2
            dest: "/etc/sqlmigrate/{{ database }}.yaml"
            mode: '0644'
        - name: clean previous migrations
          file:
            path: /var/lib/sqlmigrate/{{ database }}
            state: absent
        - name: git clone
          command: /usr/bin/git clone "{{ url }}" -b {{ branch }} /var/lib/sqlmigrate/{{ database }}
          tags:
            # Ansible is not able to git clone when using noexec on /tmp
            # https://github.com/ansible/ansible/issues/30064
            - skip_ansible_lint
        - name: run sql-migrate-wrapper
          command: /usr/bin/sql-migrate-wrapper up -config /etc/sqlmigrate/{{ database }}.yaml
```

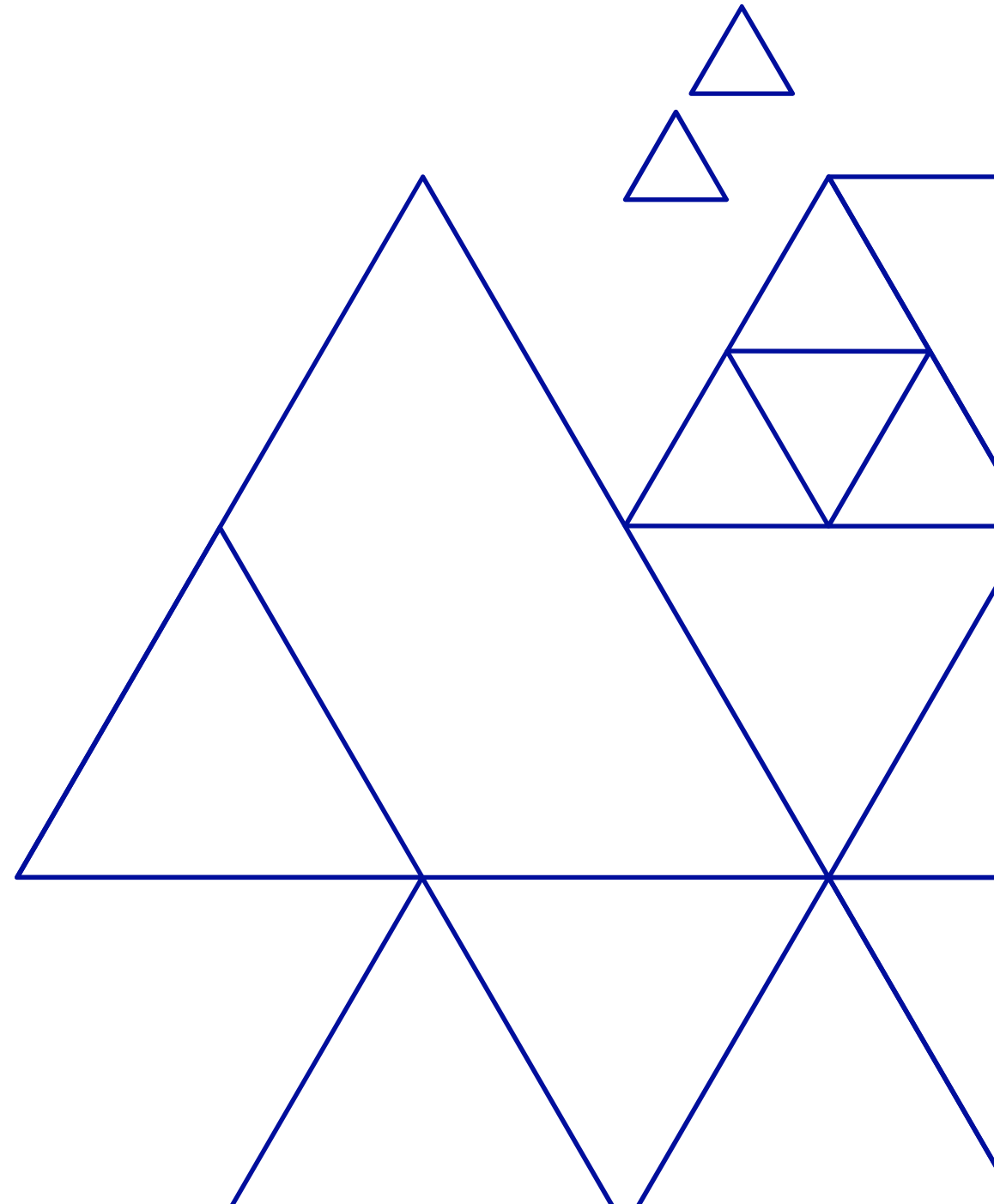
What's good

- ▶ Asynchronous
- ▶ Scalable
- ▶ Fully automated

What's bad

- ▶ Lots of components
- ▶ Home made
- ▶ Ansible can be hard to secure (sudo)

Heaven

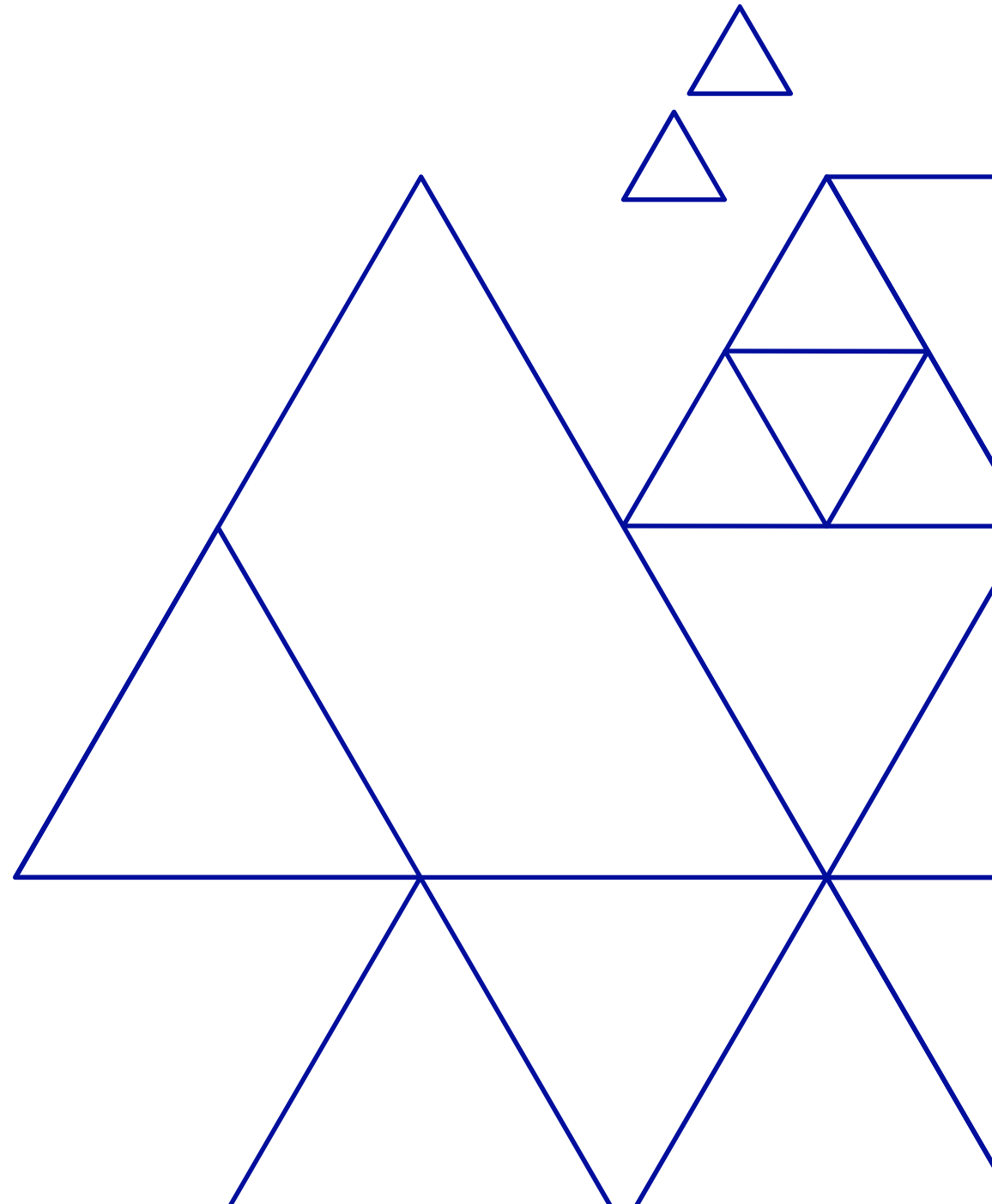


Heaven

- ▶ Code review
- ▶ Click on “merge”
- ▶ Job done



What's next?



The future

- ▶ Use standard products
 - Service discovery with Consul instead of “Inventory”
 - Ansible Tower or AWX instead of “Executor”

- ▶ Migrate legacy MySQL databases to PostgreSQL

In other communities



skeema

- ▶ **Database schema management**
- ▶ Designed for MySQL
- ▶ Like “git”

In other communities



gh-ost



PERCONA
Toolkit

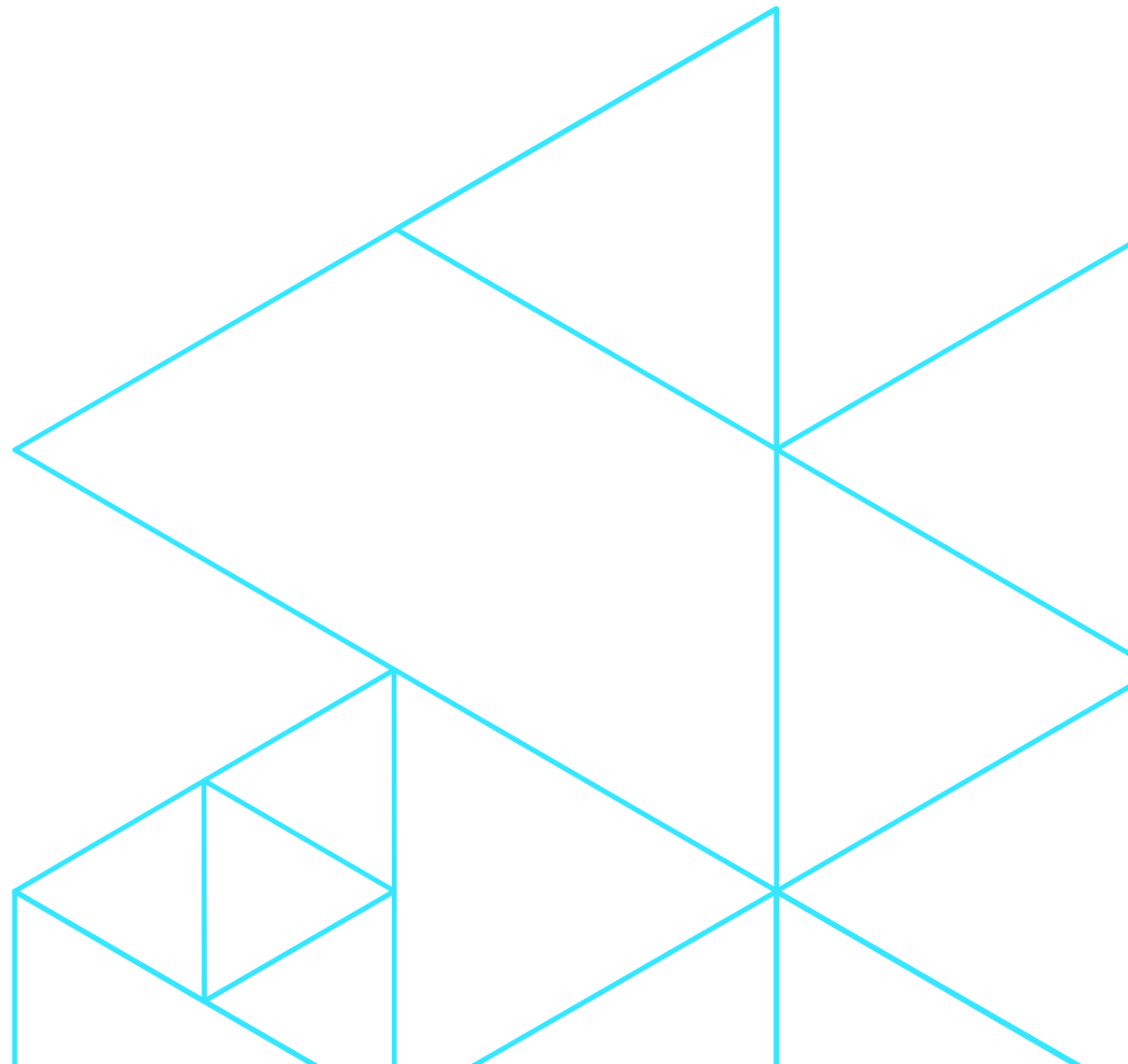
pt-online-schema-change

- ▶ **Online schema change**
- ▶ Designed for MySQL
- ▶ Controllable
- ▶ Logical replication or triggers
- ▶ Focus on tables

Useful resources

- ▶ “Automating schema migration flow with GitHub Actions, skeema & gh-ost”
https://fosdem.org/2020/schedule/event/mysql_github_schema/
- ▶ “Challenges of Concurrent DDL - Robert Haas” <https://www.youtube.com/watch?v=kbtkKh9B7eo>
- ▶ “Transparent DDL replication using Pglogical” https://github.com/enova/pgl_ddl_deploy
- ▶ Schema comparison tool “pgquarrel”: <https://github.com/eulerto/pgquarrel>

Thank you



Questions?

