# Just use Postgres

Victor Adossi

June, 2019

# Roadmap

- What is Postgres?
- Why Postgres?
- Why not Postgres?
- SQL vs NOSQL
- Relational Data
- Key/Value Data
- Document Storage
- Graph Data
- Geospatial Data
- Log Storage
- Message Queue
- Time Series Data

# What is Postgres?

**Postgres is the most advanced open source database that's ever existed**. It's developed in the open, driven and maintained by the community.

There are a few large contributors in the space:

- 2nd Quadrant
- Citus Data (acquired by MS in January)

A few features set postgres apart:

- Multi Version Concurrency Control (MVCC)
- Plugin system (indices, functionality)
- Process-per-connection model
- Elephant mascot

# Why Postgres?

There are lots of reasons to use Postgres:

**Reliability** - Postgres is rock solid

**Performance** - Postgres is generally "fast enough", and can even be *really fast*

**Cloud vendor support** - AWS RDS, Azure Database, GCP Cloud SQL

**Open Source** - There are a lot of resources and other people pushing the boundaries

Reddit got to a billion users on a master-slave Postgres (scaling up rather than out, mostly)[1]

# Why Not Postgres?

There are many reasons Postgres might *not* be a fit for your next project, here are a few:

No vendor, so no vendor to pay to help[2]

No official scale out story[3]

Structured Query Language (SQL) can be difficult

Transactional guarantees can eat into performance

[2] Lots of consultancies though (like 2nd Quadrant) which can help out

[3] PostgresXL does exist

# NOSQL vs SQL

What people *normally* mean when they say "NOSQL" is a rejection of the structure, and transactional guarantees normally included with a Relational DataBase Management Systems (RDBMS)

**Relational Structure** as in Relational Algebra (sets, projection, unions, intersections, joins)

**Transactional Guarantees** as in ACID (Atomicity, Consistency, Isolation, Durability)

Examples of NOSQL databases:

- RethinkDB / MongoDB (documents, usually JSON)
- Redis (key/value)
- Neo4J (graphs)
- Postgres (more on this later)

# Relational Data

99% of the data your organization needs to deal with is going to be relational – most data isn't very useful without context.

```
SELECT company_name,amount,payment_status
FROM customers
JOIN invoices ON customers.id = invoices.customer_id
WHERE payment_status=='not-paid';
```

Want to go deeper? Postgres has `ENUM`s and custom `TYPE`s, and most tools you'd need to make sure the only data is *valid*, *correct* data.

# Key/Value Data

Postgres makes a surprisingly good simple key value store. You're not going to beat Redis, but it's *probably* going to be fast enough!

```
CREATE UNLOGGED TABLE kv (
  id serial GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  key text NOT NULL,
  value jsonb,
  created_at timestamptz NOT NULL DEFAULT NOW()
);
```

Want to go deeper? Pluggable storage engines (the table access interface)[4] has landed, you could *actually* put Redis in your Postgres

[4] https://www.postgresql.org/docs/devel/tableam.html

# Document Storage

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE TABLE docs (
  id uuid PRIMARY KEY DEFAULT uuid_generate_v4(),
  data jsonb,
  updated_at timestamptz NOT NULL DEFAULT NOW(),
  created_at timestamptz NOT NULL DEFAULT NOW()
);

-- GIN indexes massively speed up searches like:
-- SELECT * FROM docs WHERE data @> {"some_key": "some_value"}
CREATE INDEX docs_data_idx ON docs USING GIN (data);
```

Want to go deeper? Look into Postgres's full range of JSON operators[5]. SQL/JSON (JSONPath for SQL) is coming in 12[6].

[5] https://www.postgresql.org/docs/current/functions-json.html

[6] https://www.postgresql.org/docs/12/functions-json.html#FUNCTIONS-SQLJSON-PATH

# Graph Data

Recursive Common Table Expressions (CTEs) are here to help:

```
WITH RECURSIVE search_graph(id, link, data, depth, path, cycle) AS (
    SELECT g.id, g.link, g.data, 1,
      ARRAY[g.id],
      false
    FROM graph g
  UNION ALL
    SELECT g.id, g.link, g.data, sg.depth + 1,
      path || g.id,
      g.id = ANY(path)
    FROM graph g, search_graph sg
    WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;
```

OK, *maybe* don't do this, but maybe *do* use AgensGraph[7], a graphing solution built on Postgres.

Expect more graphing solutions to be build into Postgres once pluggable storage engines pick up steam.

[7] https://github.com/bitnine-oss/agensgraph

# Geospatial Data

Geographic Information System (GIS) data is the bread and butter of PostGIS[8]:

```
SELECT superhero.name
FROM city, superhero
WHERE ST_Contains(city.geom, superhero.geom)
AND city.name = 'Gotham';
```

Want to go deeper? Actually read and understand the feature set and documentation for PostGIS – there's a lot to master

[8] https://postgis.net

# Log Storage

Declarative partitioning means Postgres can take your gobs of structured logs (they *are* structured right?)

```
-- The partitioned table
CREATE TABLE logs (
  data   jsonb NOT NULL,
  logged_at timestamptz NOT NULL DEFAULT NOW()
) PARTITION BY RANGE (logged_at);

-- A partition for the month of June
SET TIME ZONE 'Asia/Tokyo';

CREATE TABLE logs_2019_06
PARTITION OF logs
FOR VALUES FROM ('2019-06') TO ('2019-07');
```

Some assembly/maintenance *is* required, but faster queries on smaller data sets (constraint exclusion) has never been cheaper.

# Message Queues

If all your application instances are connected to the database, why not have them communicate?

```
-- Create a channel named "virtual"
LISTEN virtual;

-- Notify with no payload
NOTIFY virtual;

-- notify with payload
NOTIFY virtual, 'This is the payload';
```

Maybe you don't need a NATS/RabbitMQ/NSQ/Kafka cluster *just* yet.

Want to go deeper? Try combining this feature with some `UNLOGGED` and `TEMPORARY` tables and build some data pipelines.

# Time Series Data

You could build your own solution by using `PARTITIONs`, `UNLOGGED` tables, some `TRIGGERs`, but don't bother. Just use TimescaleDB[9].

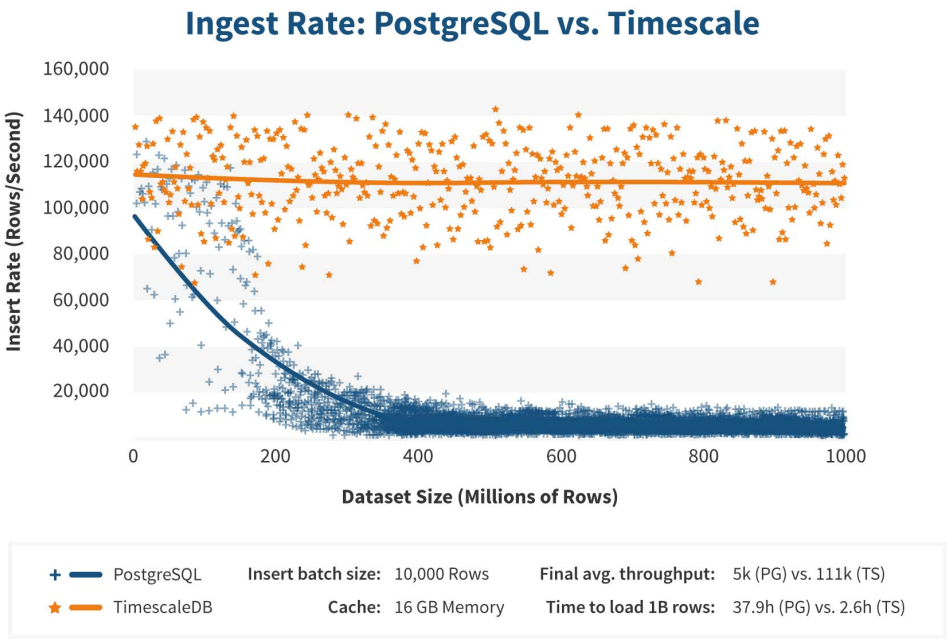


Figure 1: TimescaleDB insert performance on 1B inserts

[9] https://docs.timescale.com/v1.3/introduction

# Time Series Data (continued)

TimescaleDB also has an excellent, reasoned technical dives on where and why they can beat databases like MongoDB[10] and even purpose-built DBs like InfluxDB[11].

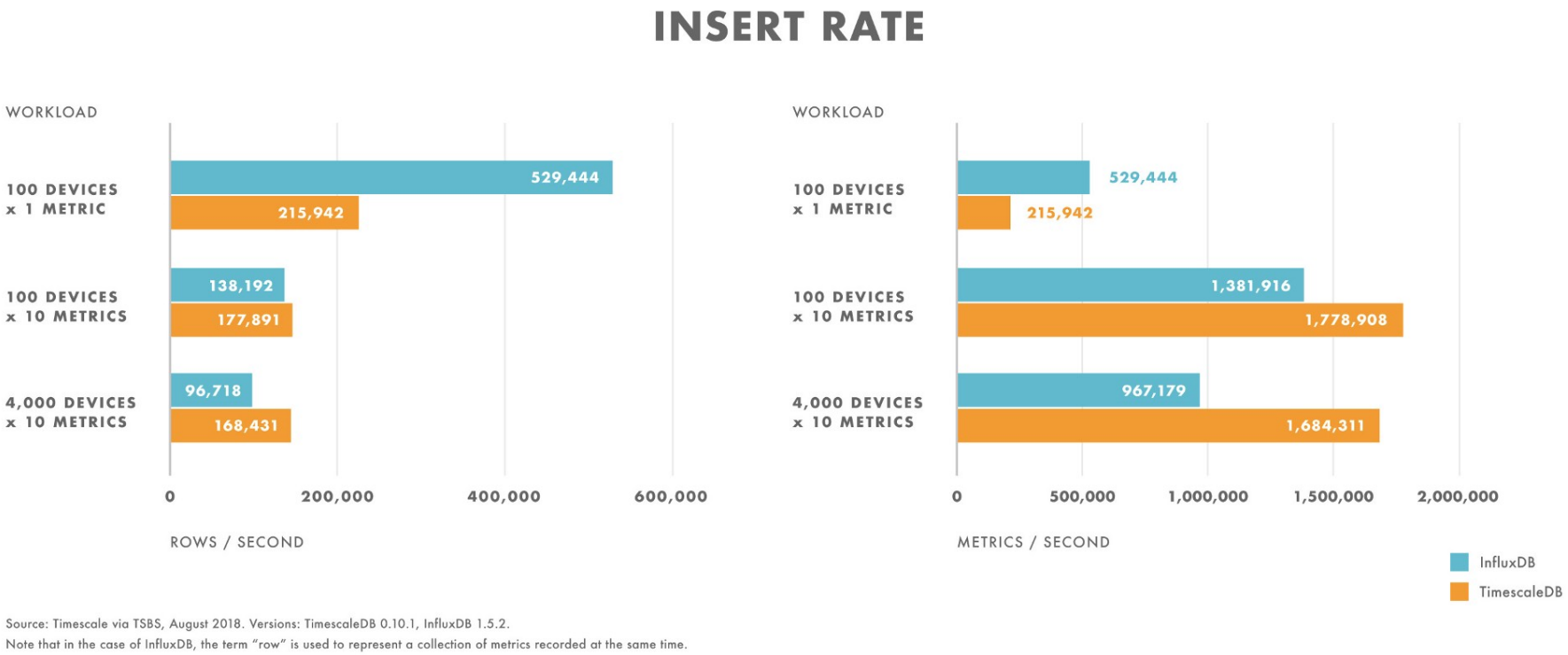


Figure 2: TimescaleDB vs influx

[10]

# The End

Thanks for listening

# whoami

If you've got any corrections, complaints, or comments, feel free to reach me using the information below:

Victor Adossi (vados@vadosware.io, vados@gaisma.co.jp)

GPG: ED874DE957CFB552

I run a couple very small consultancies to support businesses in Japan and the USA:

- GAISMA G.K. (https://gaisma.co.jp)
- VADOSWARE LLC (https://vadosware.io)

Need help getting your organization to the present/future? I can help with that.

Need help getting your organization to the past? I probably can't help with that.

# Bloopers: Hot takes and tips

A bunch of things I think that are probably right:

- Use Gitlab
- Don't write ECMAscript (AKA Javascript) without Typescript
- Try Lisp & Haskell (separately?) at least once
- Try Rust more than once
- Never price by project*
- Don't build & deploy VMs on a greenfield project in 2019**

* Unless you've built the thing already and you are literally going to reskin it and the client has absolutely *no* new feature requests.

** Unless your VM in production is basically Container Linux