Postgres Present and Future

@craigkerstiens

@craigkerstiens

Head of **Citus Cloud**

Curate **Postgres Weekly**

Launched Python support on Heroku 4.5 years ago, ran product for a number of areas at Heroku, primarily Heroku Postgres and our core languages

Pronunciation

> It might help to explain that the pronunciation is "**post-gres**" or "post-gres-cue-ell", not "post-gray-something".
>
> I heard people making this same mistake in presentations at this past weekend's Postgres Anniversary Conference :-(Arguably, the 1996 decision to call it PostgreSQL instead of reverting to plain Postgres was the single **worst mistake** this project ever made.
> It seems far too late to change now, though.
>
> regards, tom lane

Postgres – TLDR;

Datatypes

Conditional indexes

Transactional DDL

Foreign Data Wrappers

Extensions

Common Table Expressions

Geospatial capabilities

Full text search

Fast Column Addition

Listen/Notify

Table Inheritance

Per Transaction Sync. Replication

Window Functions

JSONB

Momentum

“Postgres - it’s the emacs of databases”

Rough outline

Postgres 9.5 – released 3 months ago

Postgres 9.6 – feature frozen 1 month ago

Extensions – a small tour of a few

Postgres – 9.5

Insert… on conflict do…

BRIN Indexes

Foreign schema

Grouping sets

New JSONB Operators

Insert… on conflict do…

~~Insert… on conflict do…~~

Upsert

Try to insert a record

If some key identifier is already there, simply update the record

Each user can have only one, if you try to save mine and it already exists, just update the data.

Tweets Tweets & replies Media

Pinned Tweet

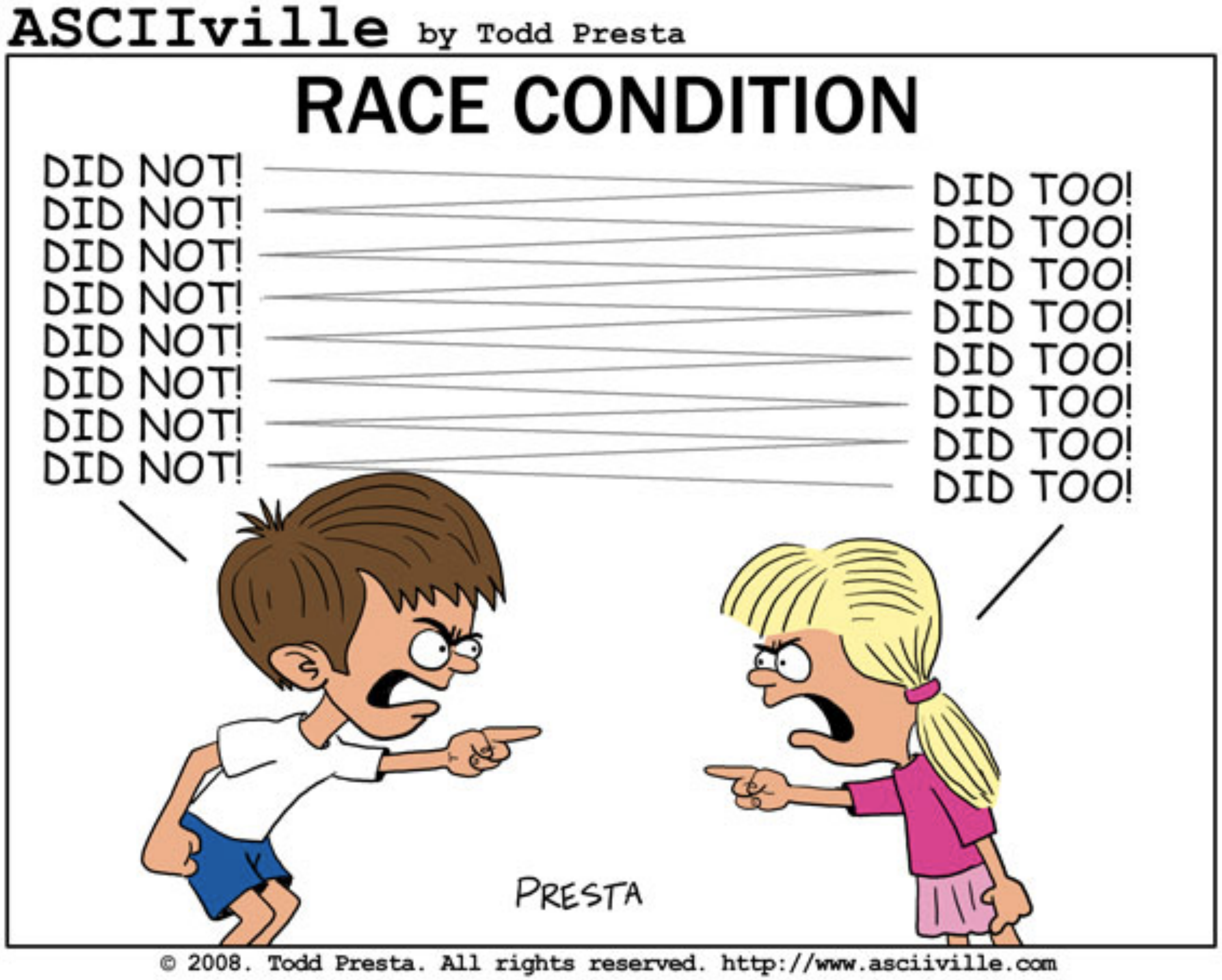
Craig Kerstiens @craigkerstiens · 22 Jun 2015

Was reminded by someone earlier today that Home Game by Michael Lewis is the best ever book on fatherhood -

Before upsert

```
WITH upsert AS (

    UPDATE pinned_tweet
    SET tally=tally+1
    WHERE user_id=1
      AND tweet_id=2 RETURNING *

)


INSERT INTO pinned_tweet (user_id, tweet_id)

SELECT 1, 3 WHERE NOT EXISTS (SELECT * FROM upsert)
```

Race conditions


ASCIIville by Todd Presta
RACE CONDITION
DID NOT! DID NOT! DID NOT! DID NOT! DID NOT! DID NOT! DID NOT! DID NOT!
DID TOO! DID TOO! DID TOO! DID TOO! DID TOO! DID TOO! DID TOO! DID TOO!
PRESTA
© 2008. Todd Presta. All rights reserved. http://www.asciiville.com

Now

Transactionally safe upsert:

```
INSERT INTO pinned_tweets (user_id, tweet_id)
VALUES (1, 3)
ON CONFLICT
DO UPDATE SET tweet_id = 5;
```

Now

Transactionally safe upsert:

```
INSERT INTO pinned_tweets (user_id, tweet_id)
VALUES (1, 3)
ON CONFLICT
DO UPDATE SET tweet_id = 5;
```

Indexes

B-Tree

GIN

GiST

KNN

SP-GiST

BRIN

Indexes - which to use

B-Tree

GIN

GiST

KNN

SP-GiST

BRIN

Indexes - which to use

B-Tree

- Default
- What you usually want

Indexes - which to use

- Gin
    - Use with multiple values in a single column
    - hstore/array/JSONB

Indexes - which to use

GiST

- Values between columns overlap
- Full text search, shapes (GIS)

Indexes

- B-Tree
- GIN
- GiST
- KNN
- SP-GiST
- BRIN

Indexes

- B-Tree
- GIN
- GiST
- **KNN**
- SP-GiST
- BRIN

Indexes

- B-Tree
- GIN
- GiST
- KNN
- **SP-GiST**
- BRIN

Indexes

B-Tree

GIN

GiST

KNN

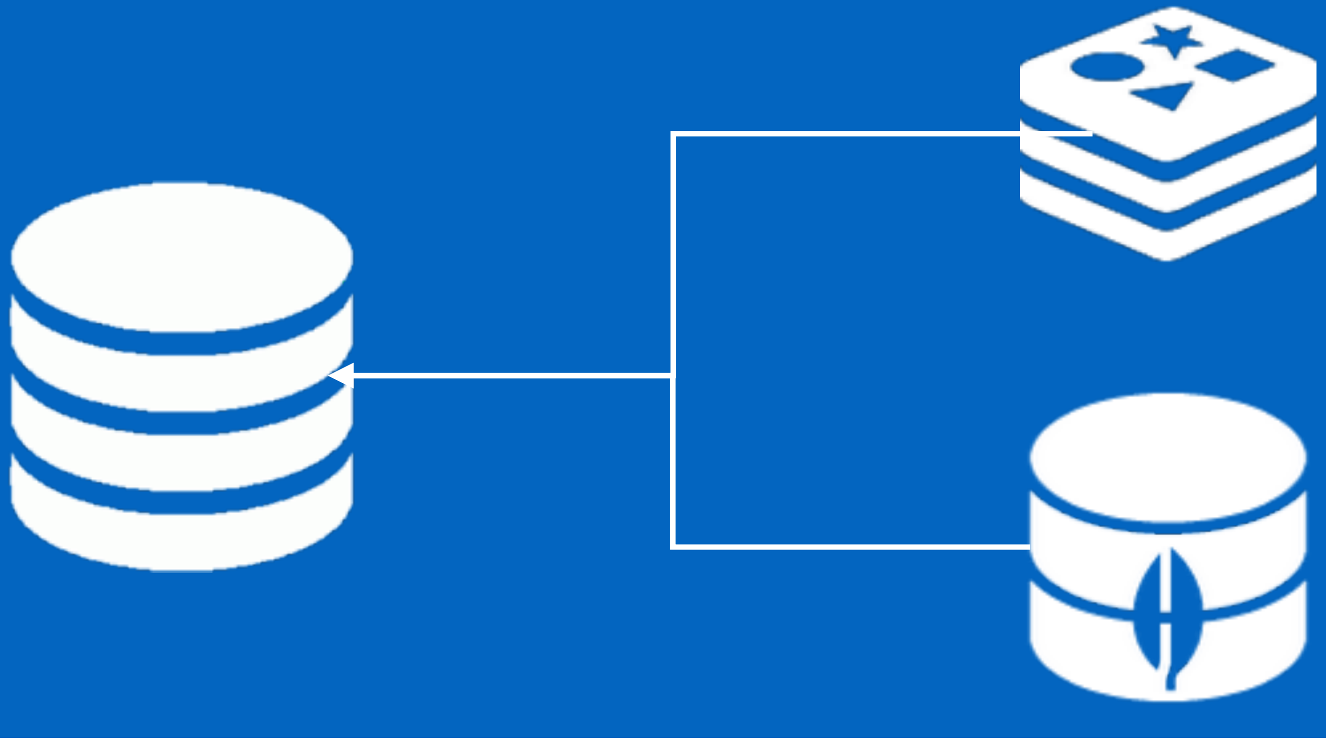
SP-GiST

BRIN

Import Foreign Schema

But first, foreign data wrappers

Connect from inside Postgres to some other data source and query directly in Postgres



Import Foreign Schema

```
CREATE EXTENSION mongo_fdw;
```

Import Foreign Schema

```
CREATE EXTENSION mongo_fdw;
CREATE SERVER foo…
```

Import Foreign Schema

```
CREATE EXTENSION mongo_fdw;
CREATE SERVER foo…
CREATE FOREIGN TABLE my_mongo_table (
    id int,
    title varchar(255),
    description text
)
```

Import Foreign Schema

```
CREATE SERVER foo…



IMPORT FOREIGN SCHEMA mongo_schema
FROM SERVER foo
INTO mongo_locally;
```

Grouping Sets

Handy for analytics that otherwise took a lot of case statements

Essentially new types of grouping that lets you easily cube data.

Grouping Sets

```
SELECT department, role, age, count(*)
FROM employees
GROUP BY your_grouping_type_here;
```

Options include: Grouping Sets, Cube, Rollup

Grouping Sets

```
GROUPING SETS (department, role, age, ());
 department |       role       |    age    | count
------------+-----------------+-----------+-------
 Finance    |                 |           |     3
 IT         |                 |           |     2
 Sales      |                 |           |     2
            |                 |           |     7
            |                 |    30     |     3
            |                 |    40     |     4
            | Accountant      |           |     1
            | Manager         |           |     3
            | Project Manager |           |     3
(9 rows)
```

Cube

```
CUBE (department, role, age);

  department |      role         |    age     | count
 ------------+-----------------+-----------+-------
  Finance    | Accountant      |    40     |     1
  Finance    | Accountant      |           |     1
  Finance    | Manager         |    40     |     1
  Finance    | Manager         |           |     1
  Finance    | Project Manager |    30     |     1
  Finance    | Project Manager |           |     1
  Finance    |                 |           |     3
  IT         | Manager         |    40     |     1
  IT         | Manager         |           |     1
…
```

JSONB

Came in Postgres 9.4

Binary JSON, more or less what mongo does

Can index it–indexes an entire document

JSONB - Functions

Concatenation

Remove specific keys

Pretty print

```
SELECT ‘{
         "name": "Craig",
         "city": “Albany”
        }'::jsonb ||
       ‘{
         "talk": “Postgres”
        }'::jsonb;

          ?column?
 ------------------------------
  {“city”: “Albany”, "name":
"Craig", "talk": "Postgres"}
 (1 row)
```

JSONB - Functions

Concatenation

Remove specific keys

Pretty print

```
SELECT ‘{
         "name": "Craig",
         "city": “Albany”
        }’::jsonb
       - ‘city’

          ?column?
 ------------------------------
  {"name": "Craig"}
 (1 row)
```

JSONB - Functions

Concatenation

Remove specific keys

Pretty print

```
SELECT jsonb_pretty(’{
         "name": "Craig",
         "city": “Albany”,
         "talk": "Postgres"
        }’::jsonb)

          ?column?
 -----------------------------
 {                      +
     "city": “Albany”, +
     "name": "Craig",  +
     "talk": "Postgres"+
 }
 (1 row)
```

Postgres – 9.5

Upsert

BRIN Indexes

Foreign schema

Grouping sets

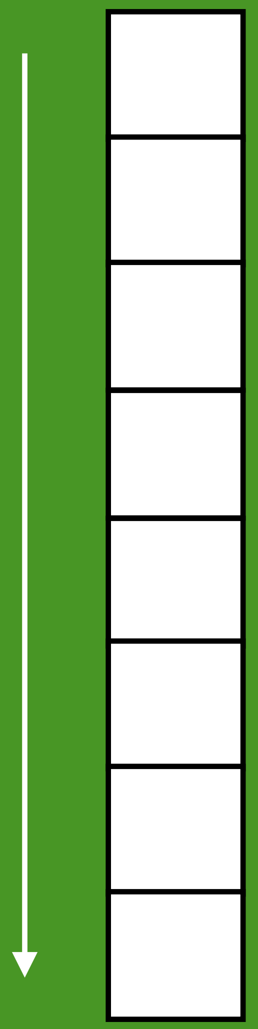
New JSONB Operators

Postgres – 9.6

- Parallelism
  - Parallel sequential scans
  - Parallel joins
- No more full-table vacuums
- Bloom filter
- Postgres FDW improvements

Parallel scans

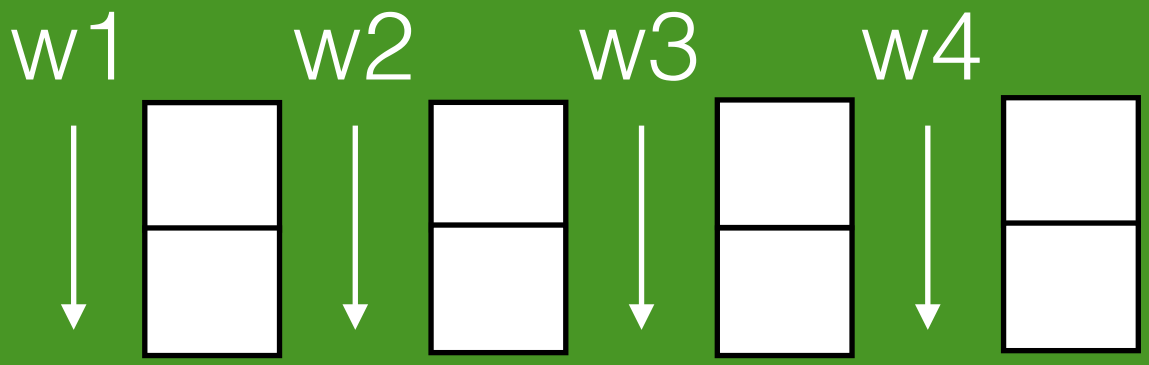
Previously on a sequential scan 1 worker process scans all rows



Parallel scans

Now:

```
set max_parallel_degree = 4;
```



Rough benchmarking shows an example of 743 ms to 213 ms -

per http://rhaas.blogspot.co.nz/2015/11/parallel-sequential-scan-is-committed.html

No more full-table vacuums

Behind the scenes every 2 billion write transactions Postgres has to check for frozen tuples

This is a problem for large databases, think terabytes without always frequent updates

Bloom filter

Space efficient probabilistic data structure

Example schema:

columns a, b, c, d, e

You want to search for WHERE a = 23 and b = 785

Indexing

```
CREATE INDEX idx_ab ON table (a, b)
```

This misses:

```
WHERE a = foo and c = bar
```

Bloom filter

```
CREATE extension bloom;

CREATE INDEX bloomidx ON table using
bloom (a, b, c, d, e);
```

Might return false positives, but Postgres will then filter those.

Postgres – 9.6

- Parallel
  - Parallel sequential scans
  - Parallel joins
- No more full-table vacuums
- Bloom filter
- Postgres FDW improvements

Extensions

Citus

Hyperloglog

Citus

Postgres works best when data is in memory

Most commonly it’s one table you need to scale out

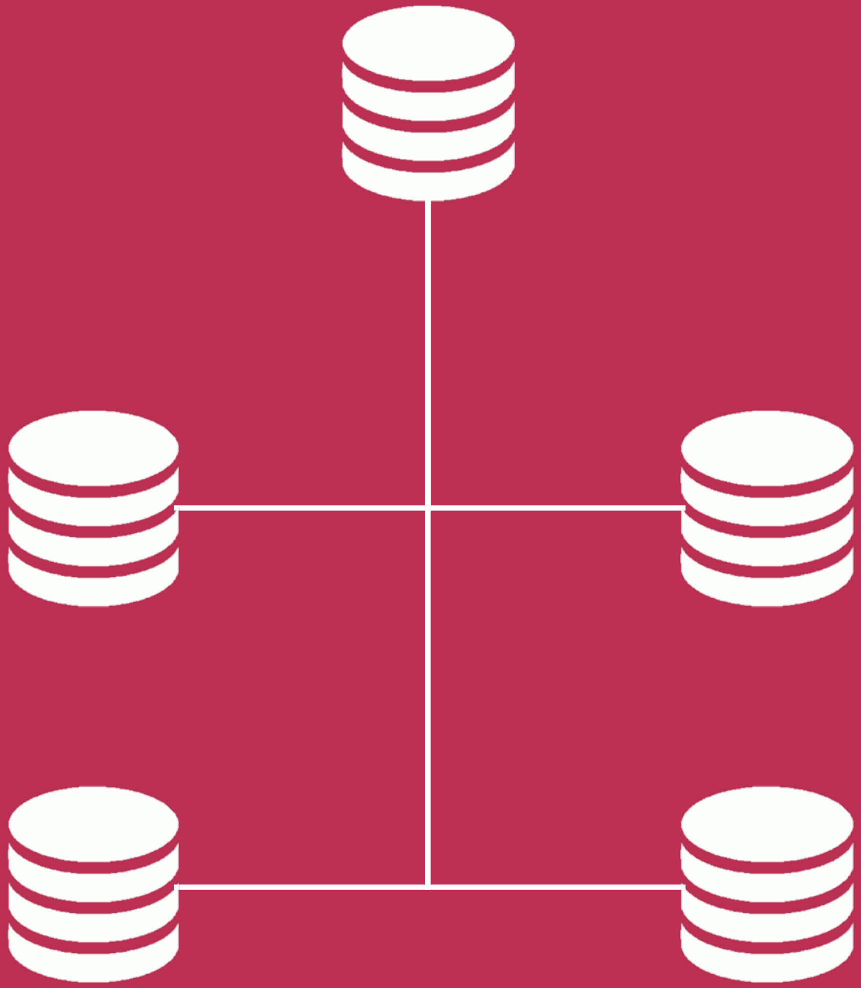
You might need to shard if you have a table called: events, logs, messages

Citus

Postgres extension that allows you to turn Postgres into a distributed database.

You go from one database, to spreading your data set across a single node

Think parallel, but across multiple instances not just scaled up



Citus

```
CREATE EXTENSION citus;

CREATE TABLE tweets (id uuid,
                     user_id uuid,
                     tweet varchar(140));

SELECT master_create_distributed_table(
       'tweets',
       'id',
       'hash');

SELECT master_create_worker_shards('tweets', 16, 1);

INSERT INTO tweets (id, user_id, tweet) VALUES (1, 2,
```

Hyper log log

$$DV_{HLL} = \text{constant} * m^2 * \left(\sum_{j=1}^{m} 2^{-R_j} \right)^{-1}$$

Hyperloglog

KMV - K Minimum Value

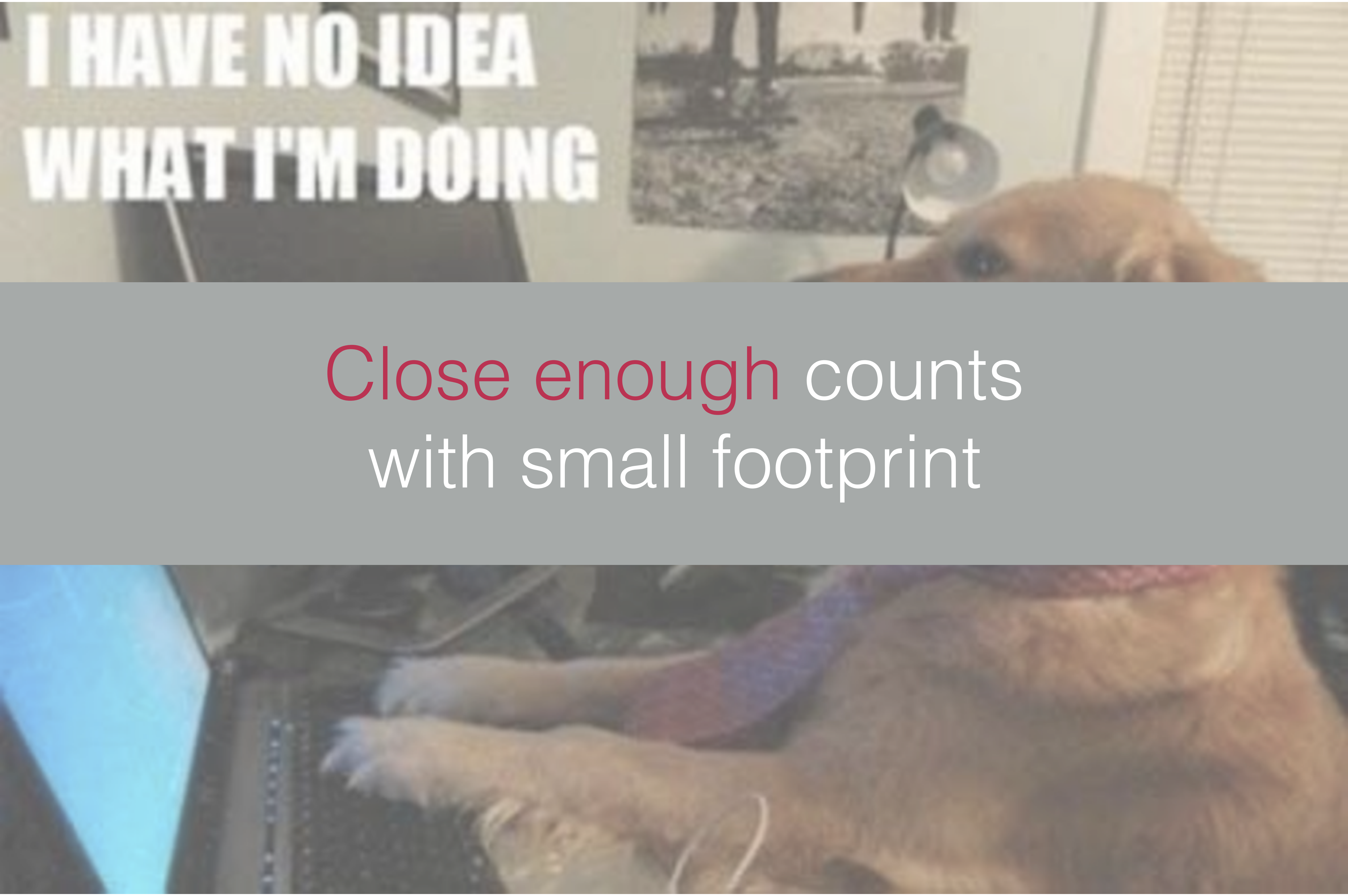
Bit observable patterns

Stochastic averaging

Harmonic averaging

I HAVE NO IDEA
WHAT I'M DOING

Probabilistic uniques
with small footprint



Close enough counts
with small footprint

Hyperloglog

```
CREATE EXTENSION hll;
CREATE TABLE daily_uniques (
    date    date unique,
    users   hll
);
```

Hyperloglog

```
CREATE EXTENSION hll;
CREATE TABLE daily_uniques (
    date    date unique,
    users   hll
);
```

Hyperloglog

```
INSERT INTO daily_uniques(date, users)
SELECT date, hll_add_agg(hll_hash_integer(user_id))
FROM users
GROUP BY 1;
```

Hyperloglog

```
SELECT EXTRACT (month from date) AS MONTH,
       hll_cardinality(hll_union_agg(users))
FROM daily_uniques
WHERE date >= ‘2016-01-01’
  AND date < ‘2016-02-01’
GROUP BY 1;
```

Hyperloglog

```
SELECT EXTRACT (month from date) AS MONTH,
       hll_cardinality(hll_union_agg(users))
FROM daily_uniques
WHERE date >= ‘2016-01-01’
  AND date < ‘2016-02-01’
GROUP BY 1;
```

Extensions

Citus

Hyperloglog

HypoPG

PLV8

Multicorn

pg_partman

pg_repack

range_partitioning

orafce

mysql_fdw

mongo_fdw

cstore_fdw

sqlite_fdw

oracle_fdw

ldap_fdw

postgres_fdw

Extensions

PGXN.org

Github

Postgres Weekly

Rough recap

- Postgres 9.5
  - released 5 months ago
  - Highlights include Upsert, lots of small usability improvements
- Postgres 9.6
  - Feature frozen 1 month ago
  - Mostly performance improvements
- Extensions
  - Citus - Horizontally scalable Postgres
  - HLL - Almost exact uniques across large datasets

Postgres – TLDR;

Datatypes

Conditional indexes

Transactional DDL

Foreign Data Wrappers

Extensions

Common Table Expressions

Geospatial capabilities

Full text search

Fast Column Addition

Listen/Notify

Table Inheritance

Per Transaction Sync. Replication

Window Functions

JSONB

Momentum