Postgres Demystified

@craigkerstiens

heroku

PSA: Macs

Postgres.app

PSA #2

Postgres Weekly

http://postgresweekly.com

PSA #3

CVE 2013-1899

UPGRADE

Agenda

Brief History

Developing w/ Postgres

Postgres Performance

Querying

Postgres History

Postgres
PostgreSQL

Post Ingres
Around since 1989/1995
Community Driven/Owned

Postgres

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 History

Postgres
PostgreSQL

Post Ingres
Around since 1989/1995
Community Driven/Owned

MVCC

Each query sees transactions committed before it

Locks for writing don't conflict with reading

Why Postgres

"Its the emacs of databases"

http://www.craigkerstiens.com/2012/04/30/why-postgres/

TLDR

Datatypes
Conditional Indexes
Transactional DDL
Foreign Data Wrappers
Concurrent Index Creation
Extensions
Common Table Expressions

Fast Column Addition
Listen/Notify
Table Inheritance
Per Transaction sync replication
Window functions
NoSQL inside SQL
Momentum

Developing

psql

```
its your friend

# \dt
# \d
# \d tablename
# \x
# \e
```

Datatypes

Datatypes

UUID

date

boolean

timestamptz

array

interval

integer

bigint

smallint

line

XML

enum

char

money

serial

bytea

point

float

inet

polygon

numeric

circle

varchar

cidr

tsquery

timetz

text

path

time

timestamp

box

macaddr

tsvector

Datatypes

UUID
boolean
date
timestamptz
array
interval
integer
bigint
smallint
line
XML
enum
char
money
serial
bytea
point
float
inet
polygon
numeric
circle
varchar
cidr
tsquery
timetz
text
path
time
timestamp
box
macaddr
tsvector

Datatypes

UUID
date
boolean
timestamptz
array
interval
integer
bigint
smallint
line
XML
enum
char
money
serial
bytea
point
float
inet
polygon
numeric
circle
varchar
cidr
tsquery
timetz
text
path
time
timestamp
box
macaddr
tsvector

Datatypes

UUID
date
boolean
timestamptz
array
interval
integer
bigint
smallint
line
XML
enum
char
money
serial
bytea
point
float
inet
polygon
numeric
circle
varchar
cidr
tsquery
timetz
text
path
time
timestamp
box
macaddr
tsvector

Arrays

```
CREATE TABLE item (
    id serial NOT NULL,
    name varchar (255),
    tags varchar(255) [],
    created_at timestamp
);
```

Arrays

```
CREATE TABLE item (
    id serial NOT NULL,
    name varchar (255),
    tags varchar(255) [],
    created_at timestamp
);
```

Arrays

```
INSERT INTO item
VALUES (1, 'Django Pony',
'{“Programming”,”Animal”}', now());
```

```
INSERT INTO item
VALUES (2, 'Ruby Gem',
'{“Programming”,”Jewelry”}', now());
```

Arrays

```
INSERT INTO item
VALUES (1, 'Django Pony',
'{“Programming”,”Animal”}', now());
```

```
INSERT INTO item
VALUES (2, 'Ruby Gem',
'{“Programming”,”Jewelry”}', now());
```

Range Types

```
CREATE TABLE talks
(
   room int,
   during tsrange
);

INSERT INTO talks VALUES
(
    3,
    '[2013-04-04 13:00, 2013-04-04 13:50)'
);
```

Range Types

```
CREATE TABLE talks
(
   room int,
   during tsrange
);

INSERT INTO talks VALUES
(
    3,
    '[2013-04-04 13:00, 2013-04-04 13:50)'
);
```

Range Types

```
CREATE TABLE talks
(
   room int,
   during tsrange
);

INSERT INTO talks VALUES
(
    3,
    '[2013-04-04 13:00, 2013-04-04 13:50)'
);
```

Range Types

```
ALTER TABLE talks
ADD EXCLUDE USING
    gist (during WITH &&);

INSERT INTO talks VALUES
(
   3,
   '[2013-04-04 13:30, 2013-04-04 14:00)'
);
ERROR:  conflicting key value violates
exclusion constraint "talks_during_excl"
```

Range Types

```
ALTER TABLE talks
ADD EXCLUDE USING
    gist (during WITH &&);

INSERT INTO talks VALUES
(
  3,
  '[2013-04-04 13:30, 2013-04-04 14:00)'
);
ERROR:  conflicting key value violates
exclusion constraint "talks_during_excl"
```

Range Types

```
ALTER TABLE talks
ADD EXCLUDE USING
    gist (during WITH &&);

INSERT INTO talks VALUES
(
   3,
   '[2013-04-04 13:30, 2013-04-04 14:00)'
);
ERROR:  conflicting key value violates
exclusion constraint "talks_during_excl"
```

Range Types

```
ALTER TABLE talks
ADD EXCLUDE USING
    gist (during WITH &&);

INSERT INTO talks VALUES
(
   3,
   '[2013-04-04 13:30, 2013-04-04 14:00)'
);
ERROR:  conflicting key value violates
exclusion constraint "talks_during_excl"
```

Extensions

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

pgrowlocks

isn

ltree

fuzzystrmatch

cube

earthdistance

dict_int

tablefunc

unaccent

btree_gist

dict_xsyn

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

pgrowlocks

isn

ltree

fuzzystrmatch

cube

earthdistance

dict_int

tablefunc

unaccent

btree_gist

dict_xsyn

NoSQL in your SQL

NoSQL in your SQL

```
CREATE EXTENSION hstore;
CREATE TABLE users (
    id integer NOT NULL,
    email character varying(255),
    data hstore,
    created_at timestamp without time zone,
    last_login timestamp without time zone
);
```

NoSQL in your SQL

```
CREATE EXTENSION hstore;
CREATE TABLE users (
    id integer NOT NULL,
    email character varying(255),
    data hstore,
    created_at timestamp without time zone,
    last_login timestamp without time zone
);
```

hStore

```
INSERT INTO users
VALUES (
 1,
 'craig.kerstiens@gmail.com',
 'sex => "M", state => “California”',
 now(),
 now()
);
```

hStore

```
INSERT INTO users
VALUES (
 1,
 'craig.kerstiens@gmail.com',
 'sex => "M", state => “California”',
 now(),
 now()
);
```

JSON

JSON

```
SELECT
     '{"id":1,"email":
       "craig.kerstiens@gmail.com",}'::json;
```

JSON

```
SELECT
     '{"id":1,"email":
       "craig.kerstiens@gmail.com",}'::json;
```

V8 w/ PLV8

JSON

```
SELECT
     '{"id":1,"email":
       "craig.kerstiens@gmail.com",}'::json;
```

V8 w/ PLV8

```
create or replace function
js(src text) returns text as $$
  return eval(
  "(function() { " + src + "})"
  )();
$$ LANGUAGE plv8;
```

JSON

```
SELECT
    '{"id":1,"email":
      "craig.kerstiens@gmail.com",}'::json;
```

V8 w/ PLV8

```
create or replace function
js(src text) returns text as $$
  return eval(
    "(function() { " + src + "})"
  )();
$$ LANGUAGE plv8;
```

JS Injection in DB: Bad Idea

Full Text Search

PostGIS

Performance

Sequential Scans

Sequential Scans

They're Bad

Sequential Scans

They're Bad (most of the time)

Indexes

Indexes

They're Good

Indexes

They're Good (most of the time)

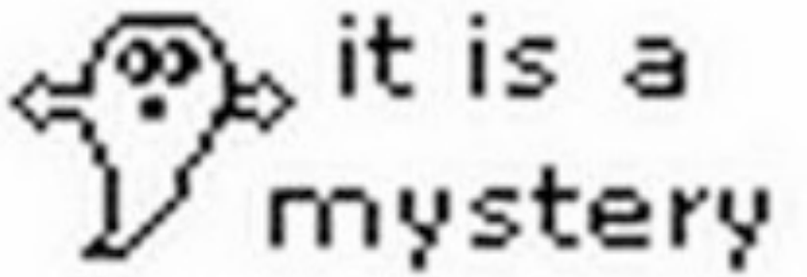
Indexes

Indexes

B-Tree
Gin
Gist
KNN
SP-Gist

Indexes

Which do I use?



Indexes

B-Tree

Default

Usually want this

Indexes

Gin

User w/ multiple values 1 column hstore/array

Indexes

Gist

Full text search

Shapes

GIS

Indexes

B-Tree
Gin
Gist
KNN
SP-Gist

Understanding Performance

Understanding Query Performance

Understanding Query Performance

```
SELECT last_name
FROM employees
WHERE salary >= 50000;
```

Explain

```
# EXPLAIN
  SELECT last_name
  FROM employees
  WHERE salary >= 50000;

                        QUERY PLAN
-----------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1
width=6)
  Filter: (salary >= 50000)
(3 rows)
```

Explain

```
# EXPLAIN
  SELECT last_name
  FROM employees
  WHERE salary >= 50000;
                        QUERY PLAN
-----------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1
width=6)
  Filter: (salary >= 50000)
(3 rows)
```

startup time

max time

rows return

Explain Analyze

```
# EXPLAIN ANALYZE
  SELECT last_name
  FROM employees
  WHERE salary >= 50000;
                    QUERY PLAN
-----------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1
width=6)  (actual time=2.401..295.247 rows=1428
loops=1)
  Filter: (salary >= 50000)
Total runtime: 295.379
(3 rows)

  Filter: (salary >= 50000)
(3 rows)
```

startup time

max time

rows return

actual time

Rough guidelines

Rare queries < 100ms

Common queries < 10 ms

Explain Analyze

```
# EXPLAIN ANALYZE
  SELECT last_name
  FROM employees
  WHERE salary >= 50000;
                    QUERY PLAN
-----------------------------------------------------------------
Seq Scan on employees (cost=0.00..35811.00 rows=1
width=6)  (actual time=2.401..295.247 rows=1428
loops=1)
  Filter: (salary >= 50000)
Total runtime: 295.379
(3 rows)

  Filter: (salary >= 50000)
(3 rows)
```

startup time

max time

rows return

actual time

Indexes!

```
# CREATE INDEX idx_emps ON employees (salary);
EXPLAIN ANALYZE
  SELECT last_name
  FROM employees
  WHERE salary >= 50000;
                        QUERY PLAN
-----------------------------------------------------------
Index Scan using idx_emps on employees
(cost=0.00..8.49 rows=1 width=6) (actual time =
0.047..1.603 rows=1428 loops=1)
  Index Cond: (salary >= 50000)
Total runtime: 1.771 ms
(3 rows)
```

Indexes!

```
# CREATE INDEX idx_emps ON employees (salary);
EXPLAIN ANALYZE
   SELECT last_name
   FROM employees
   WHERE salary >= 50000;
                        QUERY PLAN
-----------------------------------------------------------
Index Scan using idx_emps on employees
(cost=0.00..8.49 rows=1 width=6) (actual time =
0.047..1.603 rows=1428 loops=1)
   Index Cond: (salary >= 50000)
Total runtime: 1.771 ms
(3 rows)
```

Indexes!

```
# CREATE INDEX idx_emps ON employees (salary);
EXPLAIN ANALYZE
   SELECT last_name
   FROM employees
   WHERE salary >= 50000;
                        QUERY PLAN
-----------------------------------------------------------
Index Scan using idx_emps on employees
(cost=0.00..8.49 rows=1 width=6) (actual time =
0.047..1.603 rows=1428 loops=1)
   Index Cond: (salary >= 50000)
Total runtime: 1.771 ms
(3 rows)
```

Indexes!

```
# CREATE INDEX idx_emps ON employees (salary);
EXPLAIN ANALYZE
   SELECT last_name
   FROM employees
   WHERE salary >= 50000;
                        QUERY PLAN
-----------------------------------------------------------
Index Scan using idx_emps on employees
(cost=0.00..8.49 rows=1 width=6) (actual time =
0.047..1.603 rows=1428 loops=1)
   Index Cond: (salary >= 50000)
Total runtime: 1.771 ms
(3 rows)
```

Pro Tips

Pro Tips

CREATE INDEX CONCURRENTLY

Pro Tips

CREATE INDEX CONCURRENTLY

CREATE INDEX WHERE foo=bar

Pro Tips

CREATE INDEX CONCURRENTLY

CREATE INDEX WHERE foo=bar

SELECT * WHERE foo LIKE ‘%bar% is BAD

Pro Tips

CREATE INDEX CONCURRENTLY

CREATE INDEX WHERE foo=bar

SELECT * WHERE foo LIKE ‘%bar% is BAD
SELECT * WHERE Food LIKE ‘bar%’ is OKAY

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

pgrowlocks

isn

ltree

fuzzystrmatch

cube

earthdistance

dict_int

tablefunc

unaccent

btree_gist

dict_xsyn

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

pgrowlocks

isn

ltree

fuzzystrmatch

cube

earthdistance

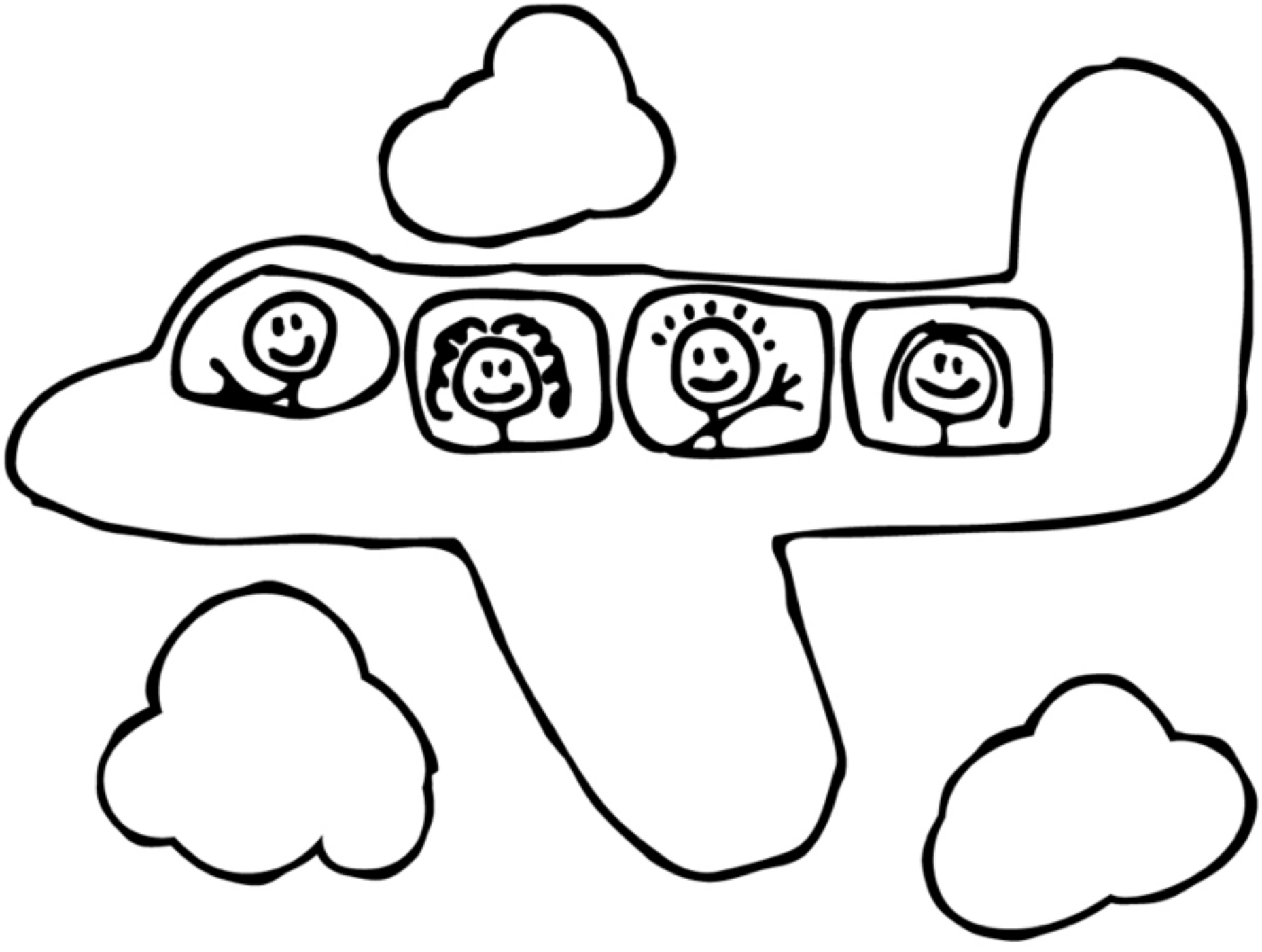
dict_int

tablefunc

unaccent

btree_gist

dict_xsyn



Cache Hit Rate

```
SELECT
        'index hit rate' as name,
        (sum(idx_blks_hit) - sum(idx_blks_read)) /
sum(idx_blks_hit + idx_blks_read) as ratio
      FROM pg_statio_user_indexes
      union all
      SELECT
       'cache hit rate' as name,
        case sum(idx_blks_hit)
          when 0 then 'NaN'::numeric
          else to_char((sum(idx_blks_hit) -
sum(idx_blks_read)) / sum(idx_blks_hit + idx_blks_read),
'99.99')::numeric
        end as ratio
      FROM pg_statio_user_indexes;)
```

Index Hit Rate

```
SELECT
  relname,
  100 * idx_scan / (seq_scan + idx_scan),
  n_live_tup
FROM pg_stat_user_tables
ORDER BY n_live_tup DESC;
```

Index Hit Rate

relname	percent_of_used	rows_in_table
events	0	669917
app_infos_user_info	0	198218
app_infos	50	175640
user_info	3	46718
rollouts	0	34078
favorites	0	3059
schema_migrations	0	2
authorizations	0	0
delayed_jobs	23	0

Rough guidelines

Cache hit > 99%

Index hit > 95%

Indexes on > 10k rows

pg_stat_statements

pg_stat_statements

$ select * from pg_stat_statements where query ~ 'from users where email';

userid	16384
dbid	16388
query	select * from users where email = ?;
calls	2
total_time	0.000268
rows	2
shared_blks_hit	16
shared_blks_read	0
shared_blks_dirtied	0
shared_blks_written	0
local_blks_hit	0
local_blks_read	0
local_blks_dirtied	0
local_blks_written	0
temp_blks_read	0
temp_blks_written	0
time_read	0
time_write	0

pg_stat_statements

$ select * from pg_stat_statements where query ~ 'from users where email';

userid	16384
dbid	16388
query	select * from users where email = ?;
calls	2
total_time	0.000268
rows	2
shared_blks_hit	16
shared_blks_read	0
shared_blks_dirtied	0
shared_blks_written	0
local_blks_hit	0
local_blks_read	0
local_blks_dirtied	0
local_blks_written	0
temp_blks_read	0
temp_blks_written	0
time_read	0
time_write	0

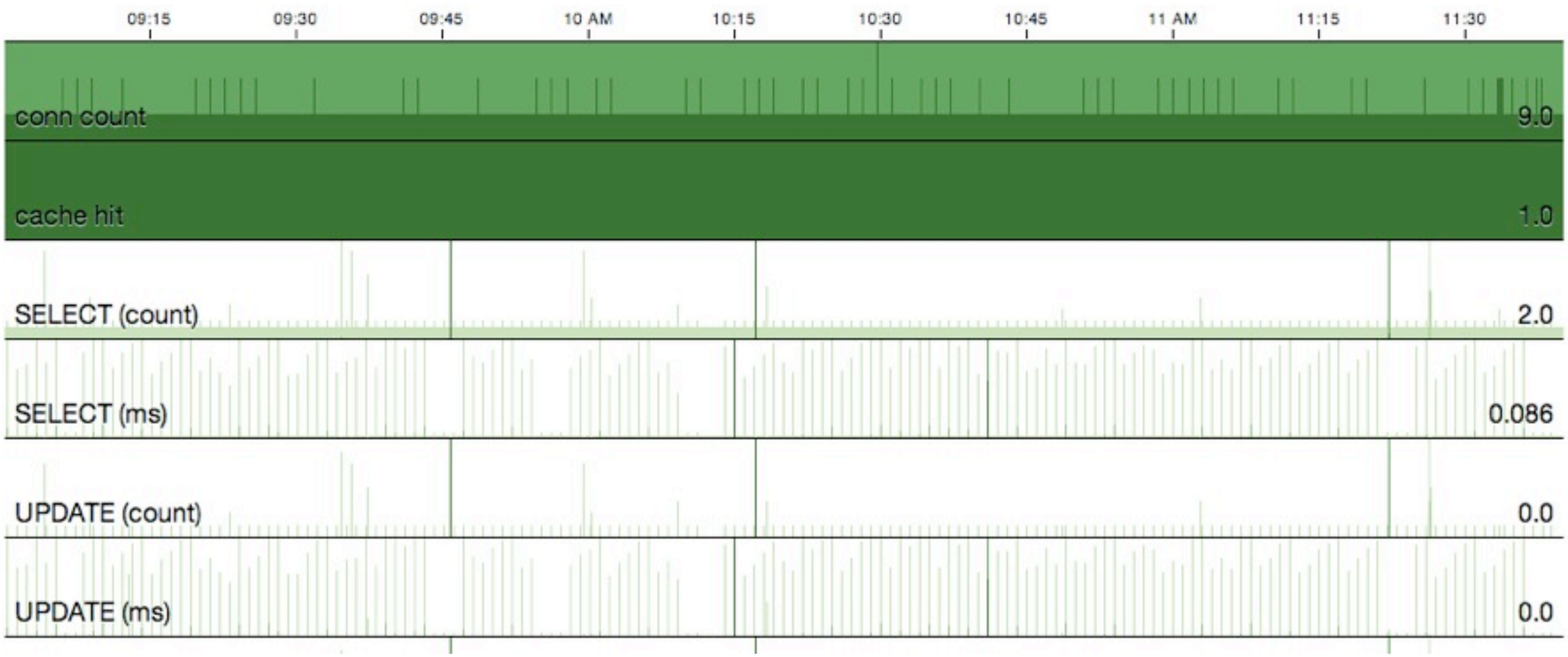
pg_stat_statements

```
SELECT
   (total_time / 1000 / 60) as total,
   (total_time/calls) as avg,
   query
FROM pg_stat_statements
ORDER BY 1 DESC
LIMIT 100;
```

pg_stat_statements

```
 total   |    avg   |              query
---------+----------+---------------------------
  295.76 |   10.13 | SELECT id FROM users...
  219.13 |   80.24 | SELECT * FROM ...
(2 rows)
```

datascope



https://github.com/will/datascope

heroku-pg-extras

```
heroku pg:cache_hit
heroku pg:index_hit
heroku pg:ps
heroku pg:locks
heroku pg:kill
heroku pg:index_size
heroku pg:unused_indexes
heroku pg:seq_scans
heroku pg:mandelbrot
```

Querying

Window Functions

Window Functions

```
SELECT
 email,
 users.data->'state',
 sum(total(items)),
 rank() OVER
  (PARTITION BY users.data->'state'
  ORDER BY sum(total(items)) desc)
FROM
 users, purchases
WHERE purchases.user_id = users.id
GROUP BY 1, 2;
```

Window Functions

```
SELECT
 email,
 users.data->'state',
 sum(total(items)),
 rank() OVER
  (PARTITION BY users.data->'state'
  ORDER BY sum(total(items)) desc)
FROM
 users, purchases
WHERE purchases.user_id = users.id
GROUP BY 1, 2;
```

Extensions

dblink

hstore

uuid-ossp

trigram

pgstattuple

citext

pgcrypto

isn

ltree

fuzzystrmatch

cube

dict_int

earthdistance

dict_xsyn

unaccent

btree_gist

tablefunc

pgrowlocks

Fuzzy String Match

```
SELECT
   soundex('Craig'),
   soundex('Will'),
   difference('Craig', 'Will');

SELECT
   soundex('Craig'),
   soundex('Greg'),
   difference('Craig', 'Greg');
```

Moving Data Around

```
\copy (SELECT * FROM users) TO ‘~/
users.csv’;
```

```
\copy users FROM ‘~/users.csv’;
```

dblink

```
SELECT dblink_connect('myconn', 'dbname=postgres');
SELECT * FROM dblink('myconn','SELECT * FROM foo') AS
t(a int, b text);

   a     |        b
---------+-------------
   1     | example
   2     | example2
```

Foreign Data Wrappers

oracle

mysql

odbc

twitter

sybase

redis

jdbc

files

couch

s3

www

ldap

informix

mongodb

Foreign Data Wrappers

```
CREATE EXTENSION redis_fdw;

CREATE SERVER redis_server
   FOREIGN DATA WRAPPER redis_fdw
   OPTIONS (address '127.0.0.1', port '6379');

CREATE FOREIGN TABLE redis_db0 (key text, value text)
   SERVER redis_server
   OPTIONS (database '0');

CREATE USER MAPPING FOR PUBLIC
   SERVER redis_server
   OPTIONS (password 'secret');
```

Redis in my Postgres

```
SELECT *
FROM redis_db0

SELECT
 id,
 email,
 value as visits
FROM
 users,
 redis_db0
WHERE ('user_' || cast(id as text)) =
cast(redis_db0.key as text)
      AND cast(value as int) > 40;
```

Redis in my Postgres

id	email	visits
2	Gaye.Monteith@aol.com	48
16	Yuki.Alber@yahoo.com	48
18	Marquis.Tartaglia@aol.com	44
31	Collin.Parrilla@gmail.com	46
6	Letitia.Tripodi@aol.com	41
12	Jami.Jeon@yahoo.com	49
44	Brady.Paramo@gmail.com	44
47	Karole.Sosnowski@gmail.com	44
39	Nydia.Bukowski@aol.com	47
40	Cherryl.Crissman@gmail.com	44
46	Laronda.Razor@yahoo.com	44
14	Jenee.Morrissey@gmail.com	47

Readability (CTEs)

CTEs – Common Table Expressions
Commonly “With clauses”

Views within a specific query

Readability (CTEs)

```
WITH top_5_products AS (
  SELECT products.*, count(*)
  FROM products, line_items
  WHERE products.id = line_items.product_id
  GROUP BY products.id
  ORDER BY count(*) DESC
  LIMIT 5
)

SELECT users.email, count(*)
FROM users, line_items, top_5_products
WHERE line_items.user_id = users.id
  AND line_items.product_id = top_5_products.id
GROUP BY 1
ORDER BY 1;
```

Readability (CTEs)

```
WITH top_5_products AS (
  SELECT products.*, count(*)
  FROM products, line_items
  WHERE products.id = line_items.product_id
  GROUP BY products.id
  ORDER BY count(*) DESC
  LIMIT 5
)

SELECT users.email, count(*)
FROM users, line_items, top_5_products
WHERE line_items.user_id = users.id
  AND line_items.product_id = top_5_products.id
GROUP BY 1
ORDER BY 1;
```

Readability (CTEs)

```
WITH top_5_products AS (
  SELECT products.*, count(*)
  FROM products, line_items
  WHERE products.id = line_items.product_id
  GROUP BY products.id
  ORDER BY count(*) DESC
  LIMIT 5
)

SELECT users.email, count(*)
FROM users, line_items, top_5_products
WHERE line_items.user_id = users.id
  AND line_items.product_id = top_5_products.id
GROUP BY 1
ORDER BY 1;
```

Few More Things

Postgresql-hll

Postgresql-hll

KMV

Bit pattern observables

Stochastic Averaging

Harmonic Averaging

Postgresql-hll

Uniques

&

Big data

Postgresql-hll

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

INSERT INTO daily_unique_purchases (date, users)
SELECT
  occurred_at::date,
  hll_add_agg(hll_hash_integer(user_id))
FROM purchases
GROUP BY 1;
```

Postgresql-hll

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

INSERT INTO daily_unique_purchases (date, users)
SELECT
  occurred_at::date,
  hll_add_agg(hll_hash_integer(user_id))
FROM purchases
GROUP BY 1;
```

Postgresql-hll

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

INSERT INTO daily_unique_purchases (date, users)
SELECT
  occurred_at::date,
  hll_add_agg(hll_hash_integer(user_id))
FROM purchases
GROUP BY 1;
```

Postgresql-hll

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

INSERT INTO daily_unique_purchases (date, users)
SELECT
  occurred_at::date,
  hll_add_agg(hll_hash_integer(user_id))
FROM purchases
GROUP BY 1;
```

Postgresql-hll

```
SELECT
  date,
  hll_cardinality(users)
FROM daily_unique_purchases ;
```

```
SELECT
  EXTRACT(MONTH FROM date) AS month,
  hll_cardinality(hll_union_agg(users))
FROM daily_unique_purchases
WHERE date >= '2012-01-01' AND
      date <  '2013-01-01'
GROUP BY 1;
```

Extras

Listen/Notify

Per Transaction Synchronous Replication

SELECT for UPDATE

Native in Ruby

Full text search	pg_search
Upsert	upsert
Listen/notify	queue_classic
hstore	sequel
arrays	sequel

TLDR

Datatypes
Conditional Indexes
Transactional DDL
Foreign Data Wrappers
Concurrent Index Creation
Extensions
Common Table Expressions

Fast Column Addition
Listen/Notify
Table Inheritance
Per Transaction sync replication
Window functions
NoSQL inside SQL
Momentum

Thanks!