Postgres Demystified

@craigkerstiens

heroku
PSA: Macs

Postgres.app
PSA #2

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
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

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

- smallint
- bigint
- integer
- numeric
- float
- serial
- money
- date
- interval
- integer
- timestamptz
- boolean
- char
- enum
- line
- point
- polygon
- box
- circle
- inet
- array
- XML
- numeric
- path
- varchar
- text
- tsquery
- timetz
- time
- timestamp
- tsvector
- UUID
- inet
- array
- date
- interval
- integer
CREATE TABLE item (  id serial NOT NULL,  name varchar (255),  tags varchar(255) [],  created_at timestamp );
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());
```
CREATE TABLE talks
(
    room int,
    during tsrange
);

INSERT INTO talks VALUES
(
    3,
);
CREATE TABLE talks
(
    room int,
    during tsrange
);

INSERT INTO talks VALUES
(
    3,
);
CREATE TABLE talks
(
    room int,
    during tsrange
);

INSERT INTO talks VALUES
(
    3,
    '[2013-04-04 13:00, 2013-04-04 13:50)'
);
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"
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"
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-oss
- trigram
- pgstattuple
- pgcrypto
- pgrowlocks
- fuzzystrmatch
- isn
- cube
- ltree
- earthdistance
- dict_int
- dict_xsyn
- tablefunc
- unaccent
- btree_gist
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(),
    now());
INSERT INTO users
VALUES (1,
'craig.kerstiens@gmail.com',
'sex => "M", state => “California”',
now(),
now()
);
SELECT
  '{"id":1,"email":
    "craig.kerstiens@gmail.com"},'}::json;
SELECT
    '{"id":1, "email":
    "craig.kerstiens@gmail.com"}',::json;

V8 w/ PLV8
SELECT
    '{"id":1,"email":
       "craig.kerstiens@gmail.com"},{}'::json;

create or replace function js(src text) returns text as $$
    return eval(
        "(function() { " + src + "}())"
    )();
$$ LANGUAGE plv8;
SELECT '{"id":1,"email": "craig.kerstiens@gmail.com"}'::json;

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?

It is a mystery
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
SELECT last_name
FROM employees
WHERE salary >= 50000;
# 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

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

**QUERY PLAN**

```
Seq Scan on employees
  width=6) 
  Filter: (salary >= 50000)
(3 rows)
```

<table>
<thead>
<tr>
<th>startup time</th>
<th>max time</th>
<th>rows return</th>
</tr>
</thead>
<tbody>
<tr>
<td>(cost=0.00..35811.00)</td>
<td>35811.00</td>
<td>1</td>
</tr>
</tbody>
</table>
# 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)
Rough guidelines

Rare queries < 100ms
Common queries < 10 ms
# EXPLAIN ANALYZE

```sql
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)  
startup time: 2.401  
max time: 295.247  
rows return: 1428

Filter: (salary >= 50000)  
actual time: 295.379  
(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)
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
hstore
uuid-ossp
trigram
pgstattuple

isn
ltree
earthdistance

cube
tablefunc

unaccent

pgcrypto
fuzzystrmatch

pgrowlocks
dict_int

earthdistance

btree_gist
dict_xsyn

unaccent
Extensions

dblink  hstore  uuid-oss

citext  pgcrypto  trigram

isn  earthdistance

cube  fuzzystrmatch

tablefunc  dict_int

unaccent  btree_gist

pgstat  pgrowlocks

dict_xsyn
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;
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

<table>
<thead>
<tr>
<th>relname</th>
<th>percent_of_used</th>
<th>rows_in_table</th>
</tr>
</thead>
<tbody>
<tr>
<td>events</td>
<td>0</td>
<td>669917</td>
</tr>
<tr>
<td>app_infos_user_info</td>
<td>0</td>
<td>198218</td>
</tr>
<tr>
<td>app_infos</td>
<td>50</td>
<td>175640</td>
</tr>
<tr>
<td>user_info</td>
<td>3</td>
<td>46718</td>
</tr>
<tr>
<td>rollouts</td>
<td>0</td>
<td>34078</td>
</tr>
<tr>
<td>favorites</td>
<td>0</td>
<td>3059</td>
</tr>
<tr>
<td>schema_migrations</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>authorizations</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>delayed_jobs</td>
<td>23</td>
<td>0</td>
</tr>
</tbody>
</table>
Rough guidelines

Cache hit > 99%
Index hit > 95%

Indexes on > 10k rows
pg_stat_statements
$ select * from pg_stat_statements where query ~ 'from users where email';

<table>
<thead>
<tr>
<th>userid</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid</td>
<td>16388</td>
</tr>
<tr>
<td>query</td>
<td>select * from users where email = ?;</td>
</tr>
<tr>
<td>calls</td>
<td>2</td>
</tr>
<tr>
<td>total_time</td>
<td>0.000268</td>
</tr>
<tr>
<td>rows</td>
<td>2</td>
</tr>
<tr>
<td>shared_blks_hit</td>
<td>16</td>
</tr>
<tr>
<td>shared_blks_read</td>
<td>0</td>
</tr>
<tr>
<td>shared_blks_dirtied</td>
<td>0</td>
</tr>
<tr>
<td>shared_blks_written</td>
<td>0</td>
</tr>
<tr>
<td>local_blks_hit</td>
<td>0</td>
</tr>
<tr>
<td>local_blks_read</td>
<td>0</td>
</tr>
<tr>
<td>local_blks_dirtied</td>
<td>0</td>
</tr>
<tr>
<td>local_blks_written</td>
<td>0</td>
</tr>
<tr>
<td>temp_blks_read</td>
<td>0</td>
</tr>
<tr>
<td>temp_blks_written</td>
<td>0</td>
</tr>
<tr>
<td>time_read</td>
<td>0</td>
</tr>
<tr>
<td>time_write</td>
<td>0</td>
</tr>
</tbody>
</table>
$ select * from pg_stat_statements where query ~ 'from users where email';

<table>
<thead>
<tr>
<th>userid</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbid</td>
<td>16388</td>
</tr>
<tr>
<td>query</td>
<td>select * from users where email = ?;</td>
</tr>
<tr>
<td>calls</td>
<td>2</td>
</tr>
<tr>
<td>total_time</td>
<td>0.000268</td>
</tr>
<tr>
<td>rows</td>
<td>2</td>
</tr>
<tr>
<td>shared_blks_hit</td>
<td>16</td>
</tr>
<tr>
<td>shared_blks_read</td>
<td>0</td>
</tr>
<tr>
<td>shared_blks_dirtied</td>
<td>0</td>
</tr>
<tr>
<td>shared_blks_written</td>
<td>0</td>
</tr>
<tr>
<td>local_blks_hit</td>
<td>0</td>
</tr>
<tr>
<td>local_blks_read</td>
<td>0</td>
</tr>
<tr>
<td>local_blks_dirtied</td>
<td>0</td>
</tr>
<tr>
<td>local_blks_written</td>
<td>0</td>
</tr>
<tr>
<td>temp_blks_read</td>
<td>0</td>
</tr>
<tr>
<td>temp_blks_written</td>
<td>0</td>
</tr>
<tr>
<td>time_read</td>
<td>0</td>
</tr>
<tr>
<td>time_write</td>
<td>0</td>
</tr>
</tbody>
</table>
SELECT
  (total_time / 1000 / 60) as total,
  (total_time/calls) as avg,
query
FROM pg_stat_statements
ORDER BY 1 DESC
LIMIT 100;
<table>
<thead>
<tr>
<th>total</th>
<th>avg</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>295.76</td>
<td>10.13</td>
<td>SELECT id FROM users...</td>
</tr>
<tr>
<td>219.13</td>
<td>80.24</td>
<td>SELECT * FROM ...</td>
</tr>
</tbody>
</table>

(2 rows)
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
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;
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-oss

citext  pgcrypto  trigram  pgstattuple

isn  ltree  fuzzystrmatch
cube  earthdistance

unaccent  dict_int

tablefunc  dict_xsyn

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>example</td>
</tr>
<tr>
<td>2</td>
<td>example2</td>
</tr>
</tbody>
</table>
Foreign Data Wrappers

oracle
mysql
odbc
twitter
files
www
couch
ldap
informix
sybase
redis
s3
jdbc
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

<table>
<thead>
<tr>
<th>id</th>
<th>email</th>
<th>visits</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><a href="mailto:Gaye.Monteith@aol.com">Gaye.Monteith@aol.com</a></td>
<td>48</td>
</tr>
<tr>
<td>16</td>
<td><a href="mailto:Yuki.Alber@yahoo.com">Yuki.Alber@yahoo.com</a></td>
<td>48</td>
</tr>
<tr>
<td>18</td>
<td><a href="mailto:Marquis.Tartaglia@aol.com">Marquis.Tartaglia@aol.com</a></td>
<td>44</td>
</tr>
<tr>
<td>31</td>
<td><a href="mailto:Collin.Parrilla@gmail.com">Collin.Parrilla@gmail.com</a></td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td><a href="mailto:Letitia.Tripodi@aol.com">Letitia.Tripodi@aol.com</a></td>
<td>41</td>
</tr>
<tr>
<td>12</td>
<td><a href="mailto:Jami.Jeon@yahoo.com">Jami.Jeon@yahoo.com</a></td>
<td>49</td>
</tr>
<tr>
<td>44</td>
<td><a href="mailto:Brady.Paramo@gmail.com">Brady.Paramo@gmail.com</a></td>
<td>44</td>
</tr>
<tr>
<td>47</td>
<td><a href="mailto:Karole.Sosnowski@gmail.com">Karole.Sosnowski@gmail.com</a></td>
<td>44</td>
</tr>
<tr>
<td>39</td>
<td><a href="mailto:Nydia.Bukowski@aol.com">Nydia.Bukowski@aol.com</a></td>
<td>47</td>
</tr>
<tr>
<td>40</td>
<td><a href="mailto:Cheryl.Crissman@gmail.com">Cheryl.Crissman@gmail.com</a></td>
<td>44</td>
</tr>
<tr>
<td>46</td>
<td><a href="mailto:Laronda.Razor@yahoo.com">Laronda.Razor@yahoo.com</a></td>
<td>44</td>
</tr>
<tr>
<td>14</td>
<td><a href="mailto:Jenee.Morrissey@gmail.com">Jenee.Morrissey@gmail.com</a></td>
<td>47</td>
</tr>
</tbody>
</table>
Readability (CTEs)

CTEs – Common Table Expressions
Commonly “With clauses”

Views within a specific query
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;
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;
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
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;
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;
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;
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;
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
- Upsert
- Listen/notify
- hstore
- arrays
- pg_search
- upsert
- queue_classic
- 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!