

# n ~~10~~ Things you probably didn't know about PostgreSQL

# How it's pronounced

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

<http://www.postgresql.org/message-id/2693.1152762174@sss.pgh.pa.us>

**psql**

\e

\l  
\l +

`\d [pattern]`  
`\d+ [pattern]`

\x auto

```
\pset linestyle unicode  
\pset border 2  
\pset null ☒  
\pset format wrapped
```

\timing

```
$ cat ~/.psqlrc
\set PROMPT1 '\%[%\033[33;1m%]%'x'%[%\033[0m%]%
[%\033[1m%]%'/%[%\033[0m%]%'R%'# '
\x auto
\pset linestyle unicode
\pset border 2
\pset null ☒
\pset format wrapped
\timing
```

```
$ print $PSQL_EDITOR
vim -c ':set ft=mysql'
```

# Datatypes

time with time zone  
bigint  
macaddr  
tsquery  
money  
interval  
numeric  
bytea  
character varying  
tsvector  
cidr  
point  
path  
bigserial  
timestamp  
timestamp with time zone  
time  
box  
uuid  
boolean  
circle  
line  
lseg  
bit  
bit varying  
with time zone  
smallserial  
text  
serial  
date  
inet  
integer  
json  
double precision  
real

time with time zone  
bigint  
macaddr  
tsquery  
**money**  
interval  
numeric  
bytea  
timestamp  
character varying  
tsvector  
cidr  
point  
path  
lseg  
bit  
timestamp with time zone  
bigserial  
time  
box  
uuid  
boolean  
circle  
line  
bit varying  
with time zone  
smallint  
date  
**inet**  
integer  
json  
**polygon**  
double precision  
real  
smallserial  
text  
serial

# Extensions

pgstattuple

pgrowlocks

pgcrypto

hstore

isn

uuid-ossp

ltree

cube

citext

trigram

fuzzystrmatch

tablefunc

pgstattuple

pgrowlocks

pgcrypto

**hstore**

isn

**uuid-ossp**

ltree

**cube**

**citext**

trigram

tablefunc

fuzzystrmatch

# Hstore

```
CREATE EXTENSION hstore;

CREATE TABLE users (
    id uuid PRIMARY KEY DEFAULT uuid_generate_v4(),
    email character varying(255),
    data hstore DEFAULT '',
    created_at timestamp with time zone DEFAULT now(),
    updated_at timestamp with time zone DEFAULT now()
);

INSERT INTO users (email, data) VALUES ('cypher@nuclearsquid.com', 'homepage => "http://nuclearsquid.com", github => "cypher", twitter => "nuclearsquid", adn => "cypher", "a key" => "a value'');
```

```
SELECT id, email FROM users WHERE data -> 'github' = 'cypher';
```

| id                                   | email  |
|--------------------------------------|--|
| 75ba2347-1d7a-4ffa-b9f6-369df7b35665 | <a href="mailto:cypher@nuclearsquid.com">cypher@nuclearsquid.com</a> |

```
SELECT skeys(data) FROM users;
```

| skeys    |
|----------|
| adn      |
| a key    |
| github   |
| twitter  |
| homepage |

# Strings only

# JSON/p|v8

```
SELECT '{"id": 1, "email": "cypher@nuclearsquid.com"}'::json;
```

```
json
```

```
{"id": 1, "email": "cypher@nuclearsquid.com"}
```

```
SELECT row_to_json(row) FROM (SELECT * FROM actors LIMIT 5 OFFSET 3000) row;
```

```
row_to_json
```

```
{"actor_id":3001,"name":"Lotte Lenya"}  
{"actor_id":3002,"name":"Lotte Verbeek"}  
{"actor_id":3003,"name":"Lou Costello"}  
{"actor_id":3004,"name":"Lou Diamond Phillips"}  
{"actor_id":3005,"name":"Lou Eppolito"}
```

```
CREATE TYPE rec AS (i integer, t text);
CREATE FUNCTION set_of_records() RETURNS SETOF rec AS
$$
    // plv8.return_next() stores records in an internal tuplestore,
    // and return all of them at the end of function.
    plv8.return_next( { "i": 1, "t": "a" } );
    plv8.return_next( { "i": 2, "t": "b" } );

    // You can also return records with an array of JSON.
    return [ { "i": 3, "t": "c" }, { "i": 4, "t": "d" } ];
$$
LANGUAGE plv8;

DO $$ plv8.elog(NOTICE, 'this', 'is', 'inline', 'code') $$ LANGUAGE
plv8;
```

# ARRAYS

```
CREATE TABLE posts (
    id      integer primary key,
    title   text,
    body    text,
    tags    text[]
);
```

```
SELECT title FROM posts WHERE tags &&
ARRAY['ruby', 'rails'];
```

```
CREATE TABLE posts (
    id      integer primary key,
    title   text,
    body    text,
    tags    text[]
);
```

```
SELECT title FROM posts WHERE tags @>
ARRAY['ruby', 'rails'];
```

# Ranges

```
SELECT daterange('("Jan 1 2013", "Jan 15  
2013")') @> 'Jan 10 2013'::date;
```

|          |
|----------|
| ?column? |
| t        |

```
CREATE TABLE reservation (room int, during tsrange);
INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
```

```
SELECT * FROM reservation WHERE during @> '2010-01-01 15:00'::timestamp;
```

| room | during   |
|------|--|
| 1108 | [ "2010-01-01 14:30:00", "2010-01-01 15:30:00" ) |

```
ALTER TABLE reservation  
    ADD EXCLUDE USING gist (room with =, during WITH &&);
```

```
INSERT INTO reservation VALUES  
(1108, '[2010-01-01 11:30, 2010-01-01 13:00]');
```

```
INSERT INTO reservation VALUES  
(1108, '[2010-01-01 14:45, 2010-01-01 15:45]');
```

```
ERROR: conflicting key value violates exclusion constraint  
"reservation_during_excl"
```

```
DETAIL: Key (during)=(["2010-01-01 14:45:00", "2010-01-01  
15:45:00")) conflicts with existing key (during)=(["2010-01-01  
14:30:00", "2010-01-01 15:30:00"]).
```

# Indexes

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

# CREATE INDEX CONCURRENTLY

```
CREATE INDEX users_active_index ON users(id)
    WHERE active IS TRUE;
CREATE INDEX posts_titles ON posts(id, title)
    WHERE deleted = false;
```

# pg\_stat\_statements

<http://www.postgresql.org/docs/9.2/static/pgstatstatements.html>

```
SELECT query, calls, total_time FROM pg_stat_statements ORDER BY total_time DESC;
```

| query  | calls | total_time |
|--|-------|------------|
| SELECT title, cube_distance(genre, ?) dist FROM movies WHERE cube_enlarge(?::cube, ?, ?) @> genre ORDER BY dist; | 1     | 36.833     |
| SELECT title FROM movies WHERE title @@ ?;   | 1     | 34.396     |
| SELECT COUNT(*) FROM movies WHERE TITLE !~ ?;  | 8     | 27.433     |
| SELECT name, dmetaphone(name), dmetaphone_alt(name), metaphone(name, ?), soundex(name) FROM actors;              | 1     | 20.438     |
| SELECT * FROM actors WHERE name % ?;   | 1     | 18.608     |

# Full Text Search

```
SELECT title FROM movies WHERE title ILIKE  
'stardust%';
```

| title             |
|-------------------|
| Stardust Memories |
| Stardust          |

```
SELECT COUNT(*) FROM movies WHERE TITLE !~*  
'^the.*';
```

|       |
|-------|
| count |
| 2211  |

```
SELECT movie_id, title FROM movies WHERE  
levenshtein(lower(title), lower('a hard day  
night')) <= 3;
```

| movie_id | title              |
|----------|--------------------|
| 245      | A Hard Day's Night |

```
SELECT show_trgm('Avatar');
```

| show_trgm                                     |
|---|
| {"a", "av", "ar", "ata", "ava", "tar", "vat"} |

```
CREATE INDEX movies_title_trigram ON movies  
USING gist (title gist_trgm_ops);
```

```
SELECT title FROM movies WHERE title %  
'Avatre';
```

|        |
|--------|
| title  |
| Avatar |

```
SELECT title FROM movies WHERE title @@ 'night  
& day';
```

| title                         |
|-------------------------------|
| A Hard Day's Night            |
| Six Days Seven Nights         |
| Long Day's Journey Into Night |

A Hard Day's Night  
Six Days Seven Nights  
Long Day's Journey Into Night

```
SELECT title FROM movies WHERE  
to_tsvector(title) @@ to_tsquery('english ',  
'night & day');
```

| title                         |
|-------------------------------|
| A Hard Day's Night            |
| Six Days Seven Nights         |
| Long Day's Journey Into Night |

```
SELECT to_tsvector('A Hard Day''s Night'),  
to_tsquery('english', 'night & day');
```

| to_tsvector                | to_tsquery      |
|----------------------------|-----------------|
| 'day':3 'hard':2 'night':5 | 'night' & 'day' |

```
SELECT * FROM actors WHERE name = 'Broos Wils';
```

| actor_id | name |
|----------|------|
|          |      |

```
SELECT * FROM actors WHERE name % 'Broos Wils';
```

| actor_id | name |
|----------|------|
|          |      |

```
SELECT title FROM movies NATURAL JOIN  
movies_actors NATURAL JOIN actors WHERE  
metaphone(name, 6) = metaphone('Broos Wils', 6);
```

| title                     |
|---------------------------|
| The Fifth Element         |
| Twelve Monkeys            |
| Armageddon                |
| Die Hard                  |
| Pulp Fiction              |
| The Sixth Sense           |
| Blind Date                |
| Die Hard with a Vengeance |
| ...                       |

```
SELECT name, dmetaphone(name), dmetaphone_alt(name), metaphone(name, 8), soundex(name) FROM actors;
```

| name                     | dmetaphone | dmetaphone_alt | metaphone | soundex |
|--------------------------|------------|----------------|-----------|---------|
| 50 Cent                  | SNT        | SNT            | SNT       | C530    |
| A Martinez               | AMRT       | AMRT           | AMRTNS    | A563    |
| A. Michael Baldwin       | AMKL       | AMXL           | AMXLBLTW  | A524    |
| Aaron Eckhart            | ARNK       | ARNK           | ARNKHRT   | A652    |
| Aaron Paul               | ARNP       | ARNP           | ARNPL     | A651    |
| Aaron Stanford           | ARNS       | ARNS           | ARNSTNFR  | A652    |
| Abbie Cornish            | APKR       | APKR           | ABKRNX    | A126    |
| Abby Dalton              | APTL       | APTL           | ABTLTN    | A134    |
| Abhay Deol               | APTL       | APTL           | ABHTL     | A134    |
| Abraham Sofaer           | APRH       | APRH           | ABRHMSFR  | A165    |
| Adam Baldwin             | ATMP       | ATMP           | ATMBLTWN  | A351    |
| Adam Beach               | ATMP       | ATMP           | ATMBX     | A351    |
| Adam Hann-Byrd           | ATMN       | ATMN           | ATMHNBRT  | A355    |
| Adam Lavorgna            | ATML       | ATML           | ATMLFRKN  | A354    |
| Adam Roarke              | ATMR       | ATMR           | ATMRRK    | A356    |
| Adam Sandler             | ATMS       | ATMS           | ATMSNTLR  | A352    |
| Adam Storke              | ATMS       | ATMS           | ATMSTRK   | A352    |
| Adam Trese               | ATMT       | ATMT           | ATMTRS    | A353    |
| Adam West                | ATMS       | ATMS           | ATMWST    | A352    |
| Addison Richards         | ATSN       | ATSN           | ATSNRXRT  | A325    |
| Adele Mara               | ATLM       | ATLM           | ATLMR     | A345    |
| Aden Young               | ATNN       | ATNN           | ATNYNK    | A355    |
| Adewale Akinnuoye-Agbaje | ATLK       | ATLK           | ATWLKNYK  | A342    |
| Adolfo Celi              | ATLF       | ATLF           | ATLFLSL   | A341    |
| Adolphe Menjou           | ATLF       | ATLF           | ATLFMNJ   | A341    |
| (and so on)              |            |                |           |         |

# Hyperdimensional cubes

# Genres

- No movie is 100% comedy or 100% tragedy
- Each genre is an axis
- Each movie gets a score between 0 and 10 for each genre
- 0 is nonexistent, 10 strongest

```

SELECT
    title,
    cube_distance(genre, '(0,7,0,0,0,0,0,0,0,7,0,0,0,0,10,0,0,0)' ) dist
FROM movies
WHERE
    cube_enlarge('(0,7,0,0,0,0,0,0,0,7,0,0,0,0,10,0,0,0)')::cube, 5, 18) @> genre
ORDER BY dist;

```

| title  | dist             |
|--|------------------|
| Star Wars                                      | 0                |
| Star Wars: Episode V - The Empire Strikes Back | 2                |
| Avatar   | 5                |
| Explorers                                      | 5.74456264653803 |
| Krull  | 6.48074069840786 |
| E.T. The Extra-Terrestrial                     | 7.61577310586391 |

# Common Table Expressions

```
CREATE TABLE department (
    id INTEGER PRIMARY KEY,      -- department ID
    -- upper department ID
    parent_department INTEGER REFERENCES department,
    name TEXT -- department name
);
```

```
INSERT INTO department (id, parent_department,
"name")
VALUES
    (0, NULL, 'ROOT'), (1, 0, 'A'),
    (2, 1, 'B'), (3, 2, 'C'),
    (4, 2, 'D'), (5, 0, 'E'),
    (6, 4, 'F'), (7, 5, 'G');
```

```
WITH RECURSIVE subdepartment AS
(
    -- non-recursive term
    SELECT * FROM department WHERE name = 'A'
    UNION ALL
    -- recursive term
    SELECT d.*
    FROM
        department AS d
    JOIN subdepartment AS sd
        ON (d.parent_department = sd.id)
)
SELECT *
FROM subdepartment
ORDER BY name;
```

```
WITH moved_rows AS (
    DELETE FROM products
    WHERE
        "date" >= '2010-10-01' AND
        "date" < '2010-11-01'
    RETURNING *
)
INSERT INTO products_log
SELECT * FROM moved_rows;
```

# Custom composite types

```
CREATE TYPE squid AS (
    length      float,
    tentacles   integer,
    weight      float
);
```

```
CREATE TABLE atlantic_squid (
    squid_key    bigserial primary key,
    a_squid     squid
);
```

```
CREATE TABLE pacific_squid
(LIKE atlantic_squid INCLUDING ALL);
```

```
INSERT INTO atlantic_squid(a_squid) VALUES  
('(12.5, 4, 5.7)::squid');
```

```
SELECT * FROM atlantic_squid WHERE  
(a_squid).length > 12;
```

# Regexes

```
SELECT name FROM actors WHERE name ~* 'll$';
```

| name            |
|-----------------|
| Alan Marshall   |
| Albert Hall     |
| Amy Ingersoll   |
| Andie MacDowell |
| André Morell    |
| Angeline Ball   |

...

# Listen/Notify

```
A> LISTEN work_available;
```

```
B> NOTIFY work_available, '44924';
```

A> Asynchronous notification "work\_available" with payload "44924" received from server process with PID 77946.

```
B> SELECT pg_notify('work_available', '1337');
```

A> Asynchronous notification "work\_available" with payload "1337" received from server process with PID 77946.

`gem 'queue_simple'`

# Foreign Data Wrappers

```
SELECT from_user, created_at, text FROM twitter  
WHERE q = '#viennarb';
```

- CSV files
- json
- S3
- Redis
- Oracle
- MySQL
- CouchDB
- Redis
- Neo4J

# PostGIS

<http://postgis.net>

# postgres-hll

<https://github.com/aggregateknowledge/postgresql-hll>

# <http://explain.depesz.com>

| HTML |           | TEXT      |        | STATS |       |   |  |
|------|-----------|-----------|--------|-------|-------|---|--|
| #    | exclusive | inclusive | rows x | rows  | loops | node  |  |
| 1.   | 8.759     | 18.527    | ↓ 50.0 | 50    | 1     | → Hash Join<br>(cost=245.19..20648.12 rows=1 width=8) (actual time=5.754..18.527 rows=50 loops=1)<br>Hash Cond: (p1.docid = p2.docid)<br>Join Filter: ((p1.bbox && p2.bbox) AND _st_contains(p1.bbox, p2.bbox))<br>Rows Removed by Join Filter: 11376 |  |
| 2.   | 4.308     | 4.308     | ↓ 2.5  | 9692  | 1     | → Index Scan using idx_nodelabel_btree on n_test_temp p1<br>(cost=0.00..196.27 rows=3914 width=128) (actual time=0.025..4.308 rows=9692 loops=1)<br>Index Cond: (nodelabel = 790)   |  |
| 3.   | 2.645     | 5.460     | ↓ 1.4  | 5329  | 1     | → Hash<br>(cost=196.27..196.27 rows=3914 width=128) (actual time=5.460..5.460 rows=5329 loops=1)<br>Buckets: 1024 Batches: 1 Memory Usage: 818kB  |  |
| 4.   | 2.815     | 2.815     | ↓ 1.4  | 5329  | 1     | → Index Scan using idx_nodelabel_btree on n_test_temp p2<br>(cost=0.00..196.27 rows=3914 width=128) (actual time=0.015..2.815 rows=5329 loops=1)<br>Index Cond: (nodelabel = 552)   |  |

# 9.3

- Faster
- Uses mmap (no more shmax fiddling!)
- Better concurrency/improved locking
- Better and easier replication
- Faster failover
- Updateable views

# 9.3

- More Hstore functions
- More JSON operators and functions
- Lateral queries
- Custom background worker processes
- Generally more awesomeness
- Release probably in September

# Recap

- It's pronounced "Postgres-Q-L"
- Many handy datatypes available
- Lots of handy extensions
- Hstore - NoSQL in your SQL
- JSON/plv8 - Webscale in your SQL
- Arrays are useful™
- Ranges
- Full Text Search built-in

**PostgreSQL is a powerful  
data platform**

# ORMs will get in your way

(or just use Sequel)

# Questions?

GitHub: cypher

ADN: cypher

CC-BY-NC-SA 3.0

# Sources

- Adam Sanderson, “Postgres, the Best Tool You’re Already Using” (<http://www.confreaks.com/videos/2469-railsconf2013-postgres-the-best-tool-you-re-already-using>)
- Craig Kierstens, “Postgres Demystified” (<http://www.confreaks.com/videos/2338-mwrc2013-postgres-demystified>)
- Peter van Hardenberg, “Postgres, The Bits You Haven’t Found” (<http://postgres-bits.herokuapp.com/>)
- Selena Deckelmann, “Schema liberation with JSON and plv8 (and Postgres)” (<https://speakerdeck.com/selenamarie/schema-liberation-with-json-and-plv8-and-postgres>)
- “Seven Databases in Seven Weeks”, by Eric Redmond & Jim R. Wilson (Pragmatic Programmers, 2012)
- <http://www.craigkerstiens.com>
- <http://www.depesz.com/>
- <http://slid.es/xzilla/postgres-9-3>
- <http://labria.github.io/2013/04/28/rails-4-postgres-uuid-pk-guide/>
- <http://www.postgresql.org/docs/current/static/>
- <https://code.google.com/p/plv8js/wiki/PLV8>
- <http://citusdata.com/blog/65-run-sql-on-json-files-without-any-data-loads>
- [https://postgres.herokuapp.com/blog/past/2013/6/5/javascript\\_in\\_your\\_postgres/](https://postgres.herokuapp.com/blog/past/2013/6/5/javascript_in_your_postgres/)
- <http://blog.endpoint.com/2013/06/postgresql-as-nosql-with-data-validation.html>
- <http://slideshare.net/PGExperts/pg-pyandsquidpypgday>