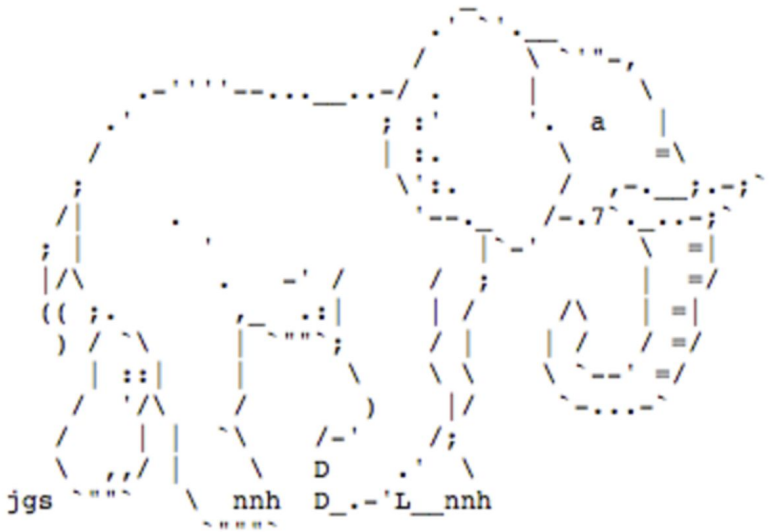
PostgreSQL Tips & Tricks

Why are elephants so sexy?

PostgreSQL Tips & Tricks

- Sexy, whaaat?
- … We’re developers… Show me the code please!



Who am I?

Denis Gasparin
Senior DBA and Web Developer

- Sviluppo di soluzioni software basate su PostgreSQL
  - PHP, NodeJS, Ruby, Python
- Cloud Architect
- Analista e Database Administrator
- Contributor del driver PDO PostgreSQL per PHP
- Pgrepup: tool opensource per l'aggiornamento a caldo di PostgreSQL
- rtshome.pgsql: ruolo ansible per interagire con PostGreSQL

ISO SQL Data types

- Numeric Data Types:
  - Integer, Float, Numeric
- Strings
  - Varchar
- Date and Time
  - With or without timestamps
- Boolean
- Binary
- XML
- Array
- JSON (SQL:2016)

PostgreSQL *Unconventional* Data Types

- Enumerated Data Types
- Geometric Types
- Network Address Types
- Text Search
- UUID
- JSONB
- Composite Types
- Range Data Types
- ltree
- citext

ENUM

- Introdotto inizialmente per compatibilità con MySQL (!) [1]
- Consente di creare un tipo che è un elenco predefinito di valori
  - l'ordine è determinato dall'ordine di inserimento
  - i valori non possono essere eliminati ma solo inseriti o aggiornati
- Casi d'uso:
  - valori costanti che non cambiano o cambiano raramente nel tempo
- Pro:
  - migliora "leggibilità" di alcune query
  - valida il dato automaticamente
- Contro:
  - non usare in caso di valori che possano essere eliminati/aggiornati (usare lookup table)
  - per aggiornarli si deve usare un DDL

(1) https://tapoueh.org/blog/2018/05/postgresql-data-types-enum/

ENUM: un esempio

```
-- status: 0 => new, 1 => in progress, 2 => completed
-- nel codice ci saranno costanti con i valori indicati
-- nel db, per aiutare nella leggibilità, si può usare un commento sul campo 'status'
CREATE TABLE issue (id serial not null primary key, name text, status smallint);
COMMENT ON COLUMN issue.status IS '0 => new, 1 => in progress, 2 => completed'
```

```
| id |                name | status |
|----|---------------------|--------|
|  1 | PgDay presentation |      2 |
|  2 |     Pgday Feedbacks |      1 |
```

```
CREATE TYPE status AS ENUM ('new', 'in_progress', 'completed');
CREATE TABLE issue (id serial not null primary key, name text, status status);
```

```
| id |                name |      status |
|----|---------------------|-------------|
|  1 | PgDay presentation |   completed |
|  2 |     Pgday Feedbacks | in_progress |
```

Network Address

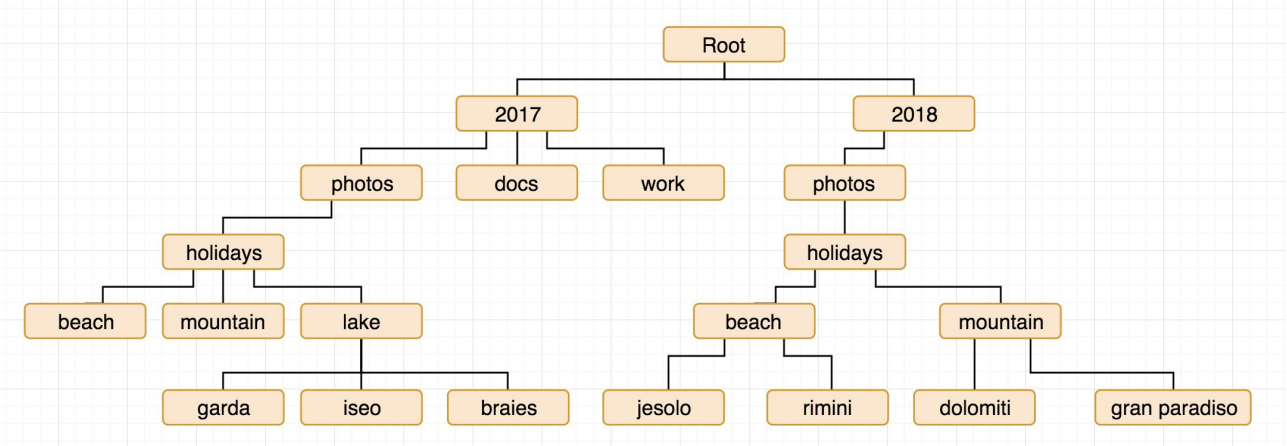
- Memorizzazione efficace di:
  - indirizzi e range di indirizzi
  - mac addresses
- Quattro tipi:
  - cidr
  - inet: più flessibile rispetto a cidr
  - macaddr
  - macaddr8
- Casi d’uso:
  - Indicizzazione di database con indirizzi IP ed informazioni geografiche
  - Firewall PostgreSQL based
- Pro:
  - funzioni specifiche per verificare, ad esempio, se un indirizzo IP è nel range
  - Supporta IPV6

citext

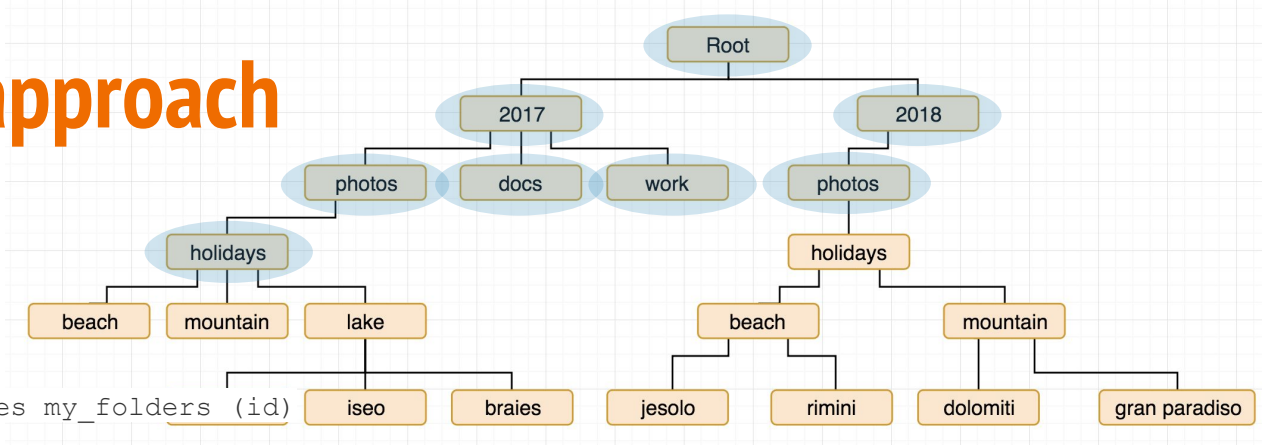
- Case Insensitive Text:
  - consente di effettuare operazioni di comparazione sul testo definito in modo indipendente rispetto al case del testo
- Casi d'uso:
  - campi testuali in cui la maggior parte se non tutte le ricerche sono case insensitive
- Pro:
  - leggermente più veloce rispetto a richiamare lower() manualmente
  - funziona nativamente con tutti gli operatori di testo (=, <>, ~=, ...) e le funzioni (regexp_match, replace, ...)
  - il case-folding dipende dal LC_CTYPE del database: una sola lingua
- Contro
  - il case-folding dipende dal LC_CTYPE del database: lingue diverse
  - da valutare l'uso in base al tipo ed alla quantità di query che coinvolgono il campo

ltree

- rappresentazione e gestione di strutture dati ad albero
  - necessario installare l'estensione ltree
- Pro:
  - semplifica la gestione e la creazione di strutture gerarchiche ad albero
  - ottimizza anche la ricerca grazie all'integrazione con l'indice gist
- Contro:
  - non è nativo (necessita di un'estensione)
- Caso d'uso:

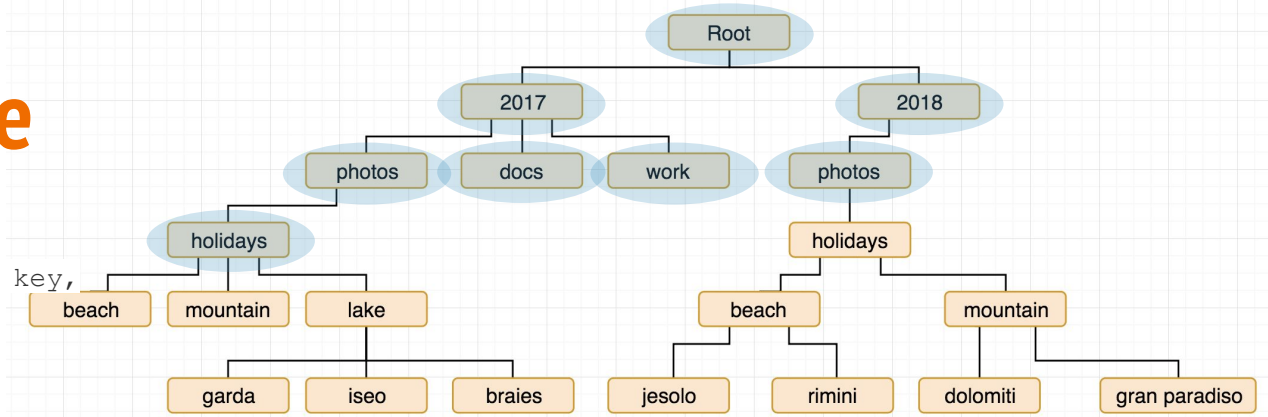


tree - classical approach




```
create table my_folders(
    id serial primary key,
    name text,
    parent_id integer references my_folders (id)
);

insert into my_folders (name, parent_id) values ('root', null); -- 1

insert into my_folders (name, parent_id) values ('2017', 1); -- 2

insert into my_folders (name, parent_id) values ('2018', 1); -- 3

insert into my_folders (name, parent_id) values ('photos', 2); -- 4

insert into my_folders (name, parent_id) values ('docs', 2); -- 5

insert into my_folders (name, parent_id) values ('work, 2); -- 6

insert into my_folders (name, parent_id) values ('photos', 3); -- 7

insert into my_folders (name, parent_id) values ('holidays', 4); -- 8

...
```

tree - using ltree




```
create table my_folders(
    id serial not null primary key,
    name text,
    path ltree
);
create index tree_path_idx on my_folders using gist (path);

insert into my_folders (name, path) values ('root', 'root');

insert into my_folders (name, path) values ('2017', 'root.2017');

insert into my_folders (name, path) values ('2018', 'root.2018');

insert into my_folders (name, path) values ('photos', 'root.2017.photos');

insert into my_folders (name, path) values ('docs', 'root.2017.docs');

insert into my_folders (name, path) values ('work, 'root.2017.work');

insert into my_folders (name, path) values ('photos', 'root.2018.photos');

insert into my_folders (name, path) values ('holidays', 'root.2017.holidays');

...
```

ltree - sample queries

- Ottenere tutte le cartelle che hanno come padre 2017

```
SELECT name FROM my_folders WHERE parent_id = (
     SELECT id FROM my_folders WHERE name = ‘2017’ AND parent_id = 1
);
SELECT name, path FROM my_folders WHERE path <@ 'root.2017’;
```

- Ottenere tutte le cartelle del 2017 che contengono la parola “garda”

```
… Query ricorsiva…

SELECT name, path FROM my_folders WHERE path @ 'root.2017.*.*garda*';
```

- Alcuni operatori e funzioni:
  - @>, <@, @
  - subltree, subpath
  - nlevel, index, lca (lowest common ancestor o longest common prefix)

Standard Boring SQL Queries

- `SELECT * FROM my_table WHERE col = 'A'`
- `SELECT * FROM order INNER JOIN product WHERE order.id = product.order_id`
- `INSERT INTO my_table (col, pol) VALUES ('A', 1);`
- `UPDATE my_table SET pol = 0 WHERE col = 'A'`
- `DELETE FROM...`

...Our *Elephant* can do something more exciting!

JSONB with lateral joins

- Che cos'è una "LATERAL JOIN"?
  - Dalla documentazione:
    "Subqueries appearing in FROM can be preceded by the key word LATERAL. This allows them to reference columns provided by preceding FROM items"

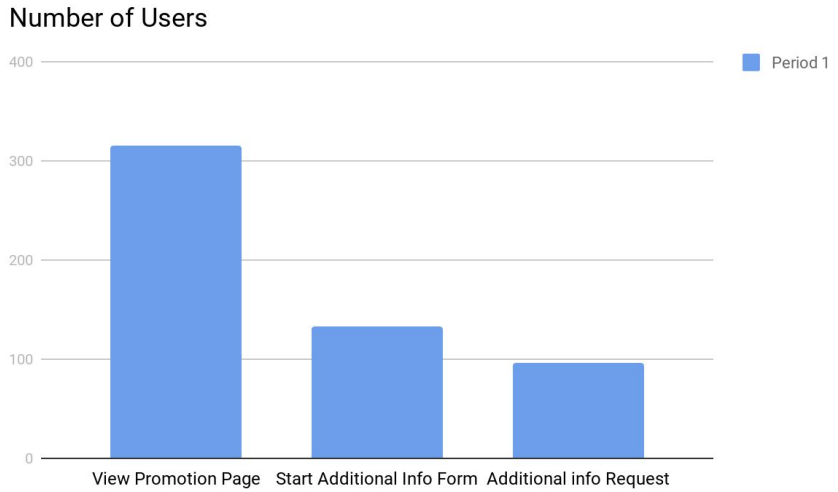
- Esempio:
  - Tabella dove tracciamo tutti gli eventi che avvengono in un sito

```
CREATE TABLE event (
    event_id SERIAL NOT NULL PRIMARY KEY,
    user_id BIGINT,
    time TIMESTAMP NOT NULL,
    data JSONB NOT NULL
);
```

  - Per ogni evento la struttura "data" traccia un tipo dell'evento e di altri dati

JSONB with lateral joins for Funnel!

- Contare tutti gli utenti che hanno visto la pagina di una promozione e poi, di questi, tutti quelli che hanno richiesto maggiori informazioni nell'arco di una settimana

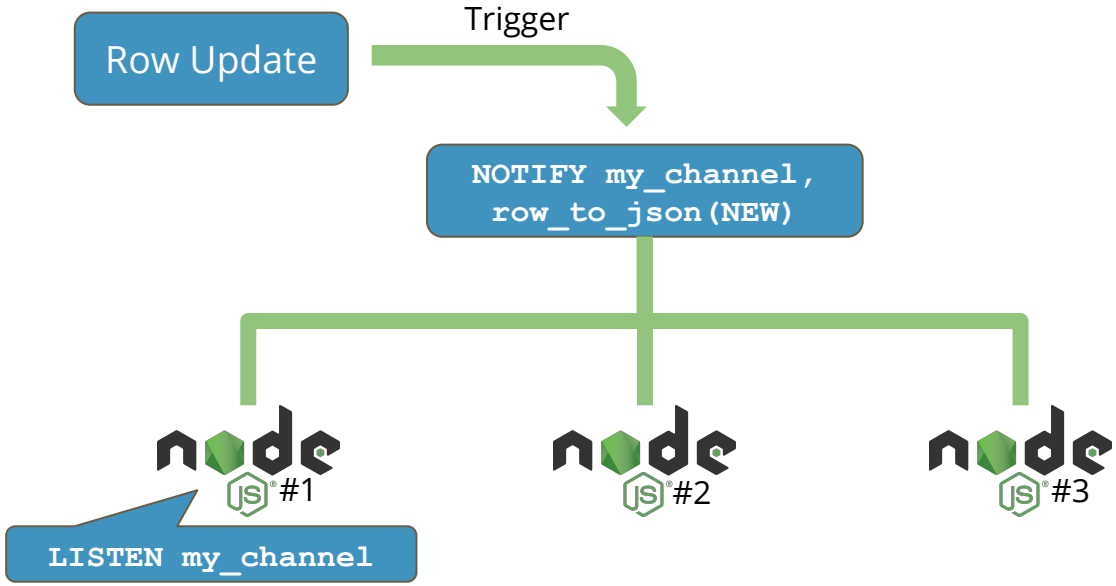


JSONB with lateral joins a GoGo

```
SELECT sum(view_prompage) AS viewed_prompage, sum(start_form) AS start_form
FROM (
    -- Get the first time each user viewed the promotion page.
    SELECT user_id, 1 AS view_prompage, min(time) AS view_prompage_time
    FROM event
    WHERE data->>'type' = 'view_promotion_page'
    GROUP BY user_id
) e1 LEFT JOIN LATERAL (
    -- For each (user_id, view_prompage_time) tuple, get the first time that
    -- user did the start_form event, if one exists within one week.
    SELECT 1 AS start_form, time AS start_form_time
    FROM event
    WHERE user_id = e1.user_id AND data->>'type' = 'start_form' AND
          time BETWEEN view_prompage_time AND (view_prompage_time + interval '1 week')
    ORDER BY time LIMIT 1
) e2 ON true
```

Notify/Listen e JSON

- Invio di messaggi tra processi
- Esempio: 1 writer - N readers

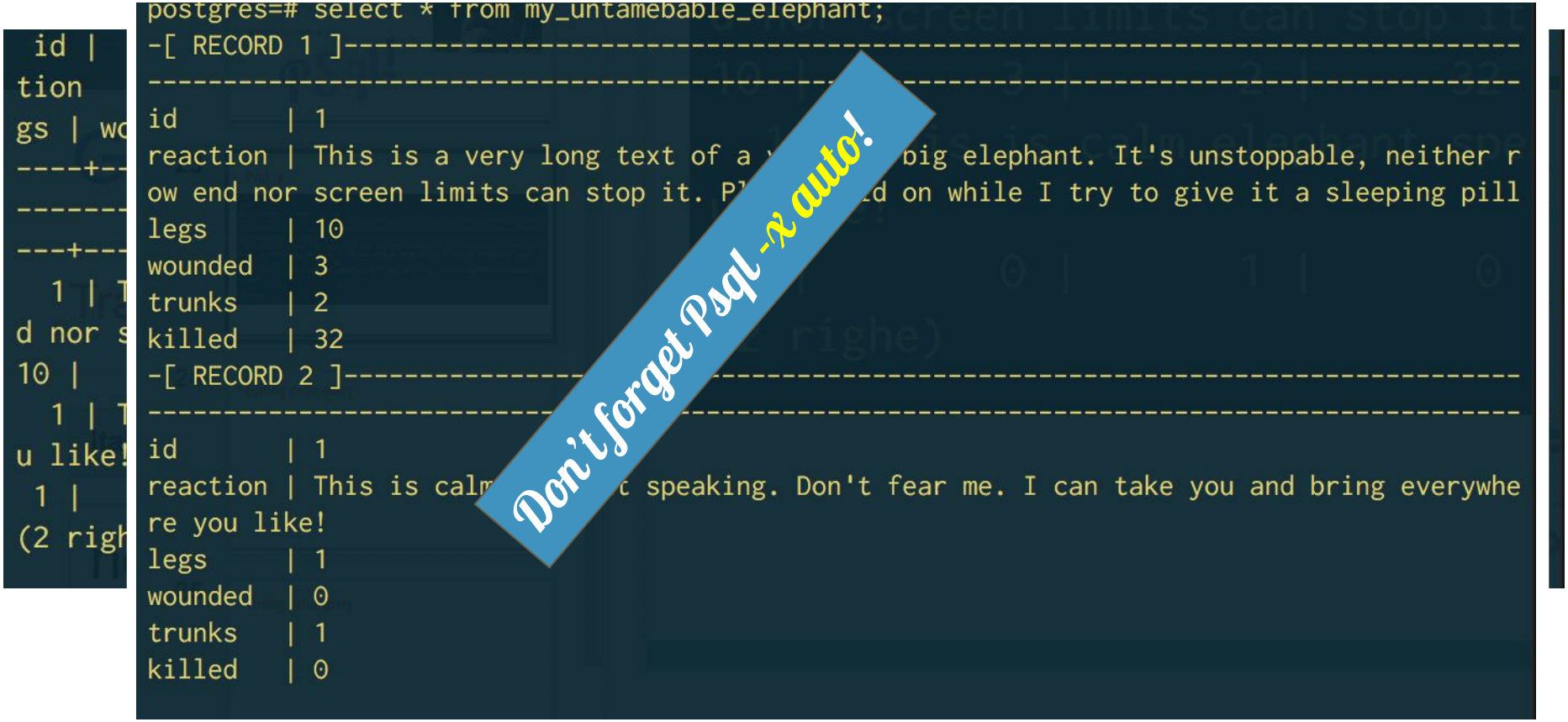


How can we *tame* our *sexy* elephant?

- Molti tool a disposizione:
  - pgadmin
  - omnidb
  - ...
- Ma il preferito è...

psql!

Psql -x


postgres=# select * from my_untamebable_elephant;
-[RECORD 1]
id | 1
reaction | This is a very long text of a big elephant. It's unstoppable, neither r
ow end nor screen limits can stop it. P d on while I try to give it a sleeping pill
legs | 10
wounded | 3
trunks | 2
killed | 32
-[RECORD 2]
id | 1
reaction | This is calm t speaking. Don't fear me. I can take you and bring everywhe
re you like!
legs | 1
wounded | 0
trunks | 1
killed | 0
Don't forget Psql -x auto!

Psql \crosstabview

```
work=# SELECT * FROM stats;
  months   |   age   | count
-----------+---------+-------
 January   | 0-18    |     5
 January   | 19-30   |    26
 January   | 31-40   |    33
 January   | 41-55   |    95
 January   | 56-70   |    73
 February  | 0-18    |    23
 February  | 19-30   |    63
 February  | 31-40   |    20
 February  | 41-55   |    55
 February  | 56-70   |    70
 March     | 0-18    |    34
 March     | 19-30   |    80
 March     | 31-40   |    58
 March     | 41-55   |    98
 March     | 56-70   |    23
 April     | 0-18    |    53
 April     | 19-30   |    83
 April     | 31-40   |    77
 April     | 41-55   |    10
 April     | 56-70   |    20
 May       | 0-18    |    52
 May       | 19-30   |    55
```

```
work=# \crosstabview months age count
  months   | 0-18 | 19-30 | 31-40 | 41-55 | 56-70
-----------+------+-------+-------+-------+-------
 January   |    5 |    26 |    33 |    95 |    73
 February  |   23 |    63 |    20 |    55 |    70
 March     |   34 |    80 |    58 |    98 |    23
 April     |   53 |    83 |    77 |    10 |    20
 May       |   52 |    55 |    40 |    55 |     7
 June      |   76 |    22 |     8 |    83 |    53
(6 rows)
```

Query Editing: \e

- Vi è mai capitato di dover scrivere una query molto lunga...

```
postgres=# insert into my_un          elephant values(1, 'This is a very long text of a very big big elepant
. It''s unstoppable, neithe          screen limits can stop it. Please hold on while I try to give it a
sleeping pill', 10, 3,2,3
```

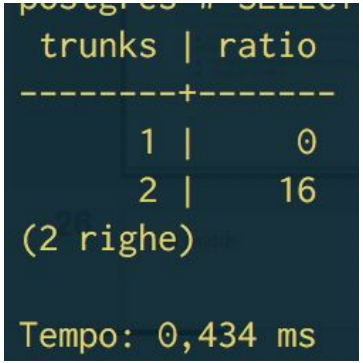
Don't forget PSQL_EDITOR env variable!

- Nooooo!
  - Se avessi `vim` potrei corregg hio...
- E `vim` sia!
  - \e

```
1 psql.edit.21319.sql +                                                                    X
insert into my_untamebable_elephant values(1, 'This is a very long text of a very big big elepant. It''s
unstoppable, neither row end nor screen limits can stop it. Please hold on while I try to give it a
sleeping pill', 10, 3,2,32);
COMMAND  psql.edit.21319.sql  +                                                100%    1:91
/elepa
```

Query Timing

- Come posso fare per sapere quanto dura una query?
  - `EXPLAIN ANALYZE SELECT trunks, SUM(killed)/SUM(trunks) FROM my_untamebable_elephant GROUP BY trunks`
    - Vedo il timing (e non solo 🐮)... non vedo il risultato!
  - \timing
    - Ad ogni query eseguita viene visualizzato il tempo
    - Misurato lato client

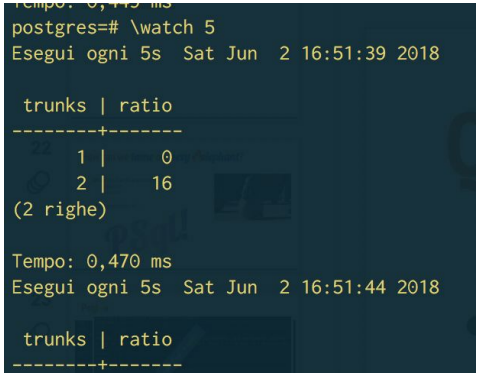


Query Timing... over Time

- E se volessi vedere come il tempo di esecuzione varia nel tempo?

\watch [SEC]

- Eseguire la query una prima volta
- Lanciare il comando `\watch 5`



...and if i need HELP ?

- No, non abbiamo bisogno di Chuck Norris!
- Usiamo
  - \?: aiuto contestuale su tutti i comandi di PSQL
  - \help: manuale abbreviato con la sintassi dei comandi

```
Comando:     ALTER TABLE
Descrizione: cambia la definizione di una tabella
Sintassi:
ALTER TABLE [ IF EXISTS ] [ ONLY ] nome [ * ]
    azione [, ... ]
ALTER TABLE [ IF EXISTS ] [ ONLY ] nome [ * ]
    RENAME [ COLUMN ] nome_colonna TO nuovo_nome_colonna
ALTER TABLE [ IF EXISTS ] [ ONLY ] nome [ * ]
    RENAME CONSTRAINT nome_vincolo TO nuovo_nome_vincolo
ALTER TABLE [ IF EXISTS ] nome
    RENAME TO nuovo_nome
ALTER TABLE [ IF EXISTS ] nome
    SET SCHEMA nuovo_schema
ALTER TABLE ALL IN TABLESPACE nome [ OWNED BY nome_ruolo [, ... ] ]
    SET TABLESPACE nuovo_tablespace [ NOWAIT ]

dove azione è una di:

    ADD [ COLUMN ] nome_colonna tipo_di_dato [ COLLATE ordinamento ] [ vincolo_di_colonna [ ... ] ]
    DROP [ COLUMN ] [ IF EXISTS ] nome_colonna [ RESTRICT | CASCADE ]
```

What to do when my *Elephant* is *Slow*?

- L’indice GIN è utile in molti casi:
  - query `LIKE '%text_to_search%'` (usando pg_trgm)
  - indicizzazione di campi JSONB
  - Full text search

- Esempio per query LIKE:

```
CREATE EXTENSION IF NOT EXISTS pg_trgm;

CREATE INDEX elephant_search_idx ON my_untamebable_elephant
USING gin (reaction gin_trgm_ops);
```

Can Elephants fly?

- Come possiamo far fare a PostgreSQL cose fuori dal normale?
- Usiamo le estensioni:
  - https://pgxn.org/
  - `CREATE EXTENSION`
- Alcuni esempi:
  - ltree
  - pg_trgm

Pgaudit and GDPR headaches

- https://www.pgaudit.org/
- Trasforma le operazioni eseguite in formato utile per analisi a posteriori

```
BEGIN
    EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';
END $$;
```

Standard Logging

```
LOG:  statement: DO $$
BEGIN
    EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';
END $$;
```

Pgaudit Logging

```
AUDIT: SESSION,33,1,FUNCTION,DO,,,"DO $$
BEGIN
    EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';
END $$;"
AUDIT: SESSION,33,2,DDL,CREATE TABLE,TABLE,public.important_table,CREATE TABLE important_table (id INT)
```

Pgaudit and GDPR headaches

- https://www.pgaudit.org/
- Trasforma le operazioni eseguite in formato utile per analisi a posteriori
- Requisiti:
  - Pgsql >= 9.5
- Features:
  - Usa il meccanismo standard di logging di PostGreSQL
  - Classi di comandi da loggare:
    - READ, WRITE
    - FUNCTION, ROLE
    - DDL, MISC
  - Logging per sessione
  - Logging per oggetto

LISTEN questions;

Grazie!

denis@gasparin.net

@rtshome

http://www.gasparin.net