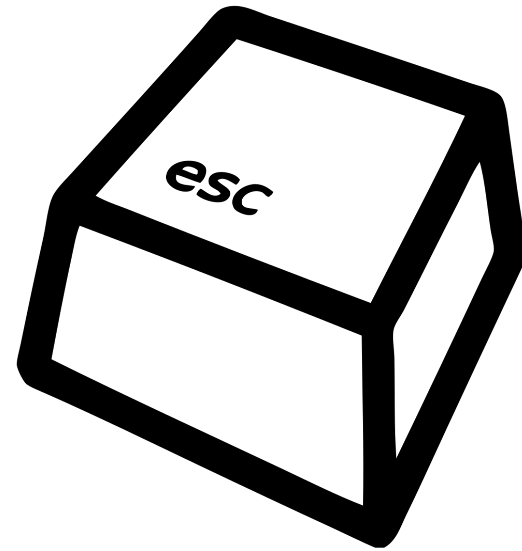


Love Your Database

Chris Mair
http://www.pgtraining.com

2016-09-03

PostgreSQL

- PostgreSQL, come probabilmente sapete, è Il più avanzato database libero
- purtroppo alcuni sviluppatori pensano che SQL e i database relazionali siano una tecnologia vecchia, poco interessante e da evitare :(
- su una delle mailing list di PostgreSQL qualche mese fa si discuteva sui feature da menzionare per convincere la gente che non è così :)
- il thread si trova su **bit.ly/2bIU6Cf**

di cosa parlerò

- in questo talk mostro alcune cose che si possono fare con PostgreSQL e che sono stati menzionati in quel thread:
    - analogie tra SQL e programmazione funzionale
    - pensare in insiemi (e non procedure o oggetti)
    - common table expression (CTE)
    - JSON
    - window function
    - SQL procedurale con PL/PgSQL

SQL e functional program'g

```
Another thing I would recommend is looking at SQL in a way
that is similar to map(), reduce(), and filter() [...]

-- Chris
```

SQL e functional program'g

- esempio: tabella dei premi Nobel:

```
select * from nobel where yr < 1905 order by yr, subject;

  yr  |  subject    |              winner
------+-------------+--------------------------------
 1901 | Chemistry   | Jacobus H. van 't Hoff
 1901 | Literature  | Sully Prudhomme
 1901 | Medicine    | Emil von Behring
 1901 | Peace       | Henry Dunant
 1901 | Peace       | Fréric Passy
 1901 | Physics     | Wilhelm Conrad Röntgen
 1902 | Chemistry   | Emil Fischer
 1902 | Literature  | Theodor Mommsen
 1902 | Medicine    | Ronald Ross
 [...]
 1904 | Literature  | Fréric Mistral
 1904 | Literature  | Joséchegaray
 1904 | Medicine    | Ivan Pavlov
 1904 | Peace       | Institute of International Law
 1904 | Physics     | Lord Rayleigh
(26 rows)
```


PGtraining

SQL e functional program'g

- rispondere alle seguenti domande senza usare cicli e condizionali:
  - chi ha vinto il premio Nobel in chimica nel 1905?
  - quando è stato consegnato il primo premio in Economia?
  - chi ha vinto più di un premio?
- che linguaggio usate per risolvere i problemi? :)

SQL e functional program'g

- chi ha vinto più di un premio nobel?

```
select winner, count(*) from nobel group by winner having count(*) > 1;

----------------------------------------------------------------+-------
 International Committee of the Red Cross                        |     3
 Frederick Sanger                                                |     2
 John Bardeen                                                    |     2
 Linus Pauling                                                   |     2
 Marie Curie                                                     |     2
 Office of the United Nations High Commissioner for Refugees    |     2
(6 rows)
```

- in effetti... no cicli e if... e la potenzialità di parallelizzare l'esecuzione (PostgreSQL >= 9.6)!

pensare in insiemi

```
A standard programmer usually has a problem with thinking
in sets. Instead she usually thinks in terms of loops, and
objects.

-- Szymon
```


PGtraining

pensare in insiemi

- che date in agosto mancano?

```
create table cal (dt date);

insert into cal values
    ('2016-08-01'), ('2016-08-02'), ('2016-08-03'),
    ('2016-08-04'), ('2016-08-05'), ('2016-08-06'),
    ('2016-08-07'), ('2016-08-08'), ('2016-08-09'),
    ('2016-08-10'), ('2016-08-11'), ('2016-08-12'),
    ('2016-08-13'), ('2016-08-14'), ('2016-08-16'),
    ('2016-08-17'), ('2016-08-18'), ('2016-08-19'),
    ('2016-08-20'), ('2016-08-22'), ('2016-08-23'),
    ('2016-08-24'), ('2016-08-25'), ('2016-08-26'),
    ('2016-08-27'), ('2016-08-28'), ('2016-08-29'),
    ('2016-08-30');
```


PGtraining

pensare in insiemi

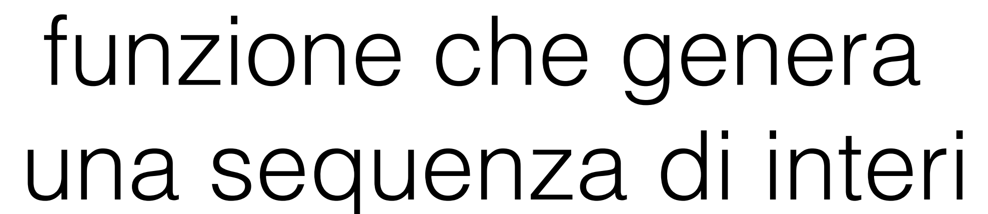
- ecco le date in agosto che mancano!

```
(select '2016-08-01'::date + x as dt from generate_series(0, 30) as x)
  except
(select dt from cal)
  order by dt;

     dt
------------
 2016-08-15
 2016-08-21
 2016-08-31
(3 rows)
```


PGtraining

aritmetica con date e interi

```
(select '2016-08-01'::date + x as dt from generate_series(0, 30) as x)
  except
(select dt from cal)
  order by dt;

     dt
------------
 2016-08-15
 2016-08-21
 2016-08-31
(3 rows)
```



funzione che genera
una sequenza di interi
-
molto utile!

PGtraining

```
(select '2016-08-01'::date + x as dt from generate_series(0, 30) as x)
  except
(select dt from cal)
  order by dt;

     dt
------------
 2016-08-15
 2016-08-21
 2016-08-31
(3 rows)
```

possiamo combinare query usando **union**, **intersect** ed **except**

PGtraining

common table expression

```
CTEs are *way* to hell at the top of that list [...]

-- Guyren
```

common table expression

- sono un modo per dare nomi alle subselect (e così scrivere codice più leggibile) - dal manuale:

```
WITH regional_sales AS (
        SELECT region, SUM(amount) AS total_sales
        FROM orders
        GROUP BY region ),

     top_regions AS (
        SELECT region
        FROM regional_sales
        WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales) )

SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```

common table expression

- la cosa carina e che possono essere "ricorsive", permettendo di fare cose che senza CTE non erano possibili in SQL - esempio:

```
create table geo_tree (
    id int primary key,
    name varchar not null,
    parent_id int references geo_tree(id)
);
```

```
insert into geo_tree values
    (1, 'Europa', null),
    (2, 'Italia', 1),
    (3, 'Slovenia', 1),
    (4, 'Veneto', 2),
    (5, 'T-AA', 1),
    (6, 'Venezia', 4),
    (7, 'Padova', 4),
    (8, 'Venezia', 6),
    (9, 'Tessera', 8);
```

common table expression

- mostra Tessera e il suo parent (join):

```
select geo_tree.*, t2.* from geo_tree inner join
       geo_tree t2 on geo_tree.id = t2.parent_id and t2.name = 'Tessera';

 id |  name   | parent_id | id |  name   | parent_id
----+---------+-----------+----+---------+-----------
  8 | Venezia |         6 |  9 | Tessera |         8
(1 row)
```

- ma se voglio fare un traversal di tutto l'albero? non so quanti join mi serviranno...

common table expression

- soluzione WITH RECURSIVE:

```
with recursive F as (
    select * from geo_tree where name = 'Tessera'
    union all
    select geo_tree.* from geo_tree inner join
           F on geo_tree.id = F.parent_id)

 select * from F;

 id |  name   | parent_id
----+---------+-----------
  9 | Tessera |         8
  8 | Venezia |         6
  6 | Venezia |         4
  4 | Veneto  |         2
  2 | Italia  |         1
  1 | Europa  |
(6 rows)
```

JSON

```
[...] JSON [...]

-- Mike
```

JSON

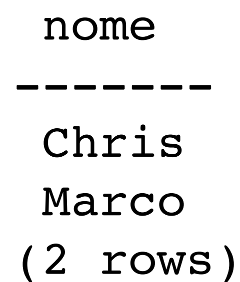
- PostgreSQL ha JSON(B) come data type nativo:

```
create table tab (data json);
insert into tab values ('{"nome":"Chris","x":77,"y":88}');
insert into tab values ('{"nome":"Marco","x":90,"y":80}');

select data->>'nome' as nome from tab;

 nome
-------
 Chris
 Marco
(2 rows)
```

- c'è un ricco set di operatori e funzioni per manipulare il dato JSON(B)

PGtraining

```
create table tab (data json);
insert into tab values ('{"nome":"Chris","x":77,"y":88}');
insert into tab values ('{"nome":"Marco","x":90,"y":80}');

select data->>'nome' as nome from tab;
```



```
 nome
-------
 Chris
 Marco
(2 rows)
```

operatore ->>
estrai campo e trasformi in testo

JSON

- ovviamente si può passare da record a JSON:

```
select to_json(nobel) from nobel where yr < 1905;

                              to_json
-----------------------------------------------------------------------
 {"yr":1901,"subject":"Chemistry","winner":"Jacobus H. van 't Hoff"}
 {"yr":1902,"subject":"Chemistry","winner":"Emil Fischer"}
 {"yr":1903,"subject":"Chemistry","winner":"Svante Arrhenius"}
 [...]
 {"yr":1903,"subject":"Physics","winner":"Henri Becquerel"}
 {"yr":1903,"subject":"Physics","winner":"Pierre Curie"}
 {"yr":1903,"subject":"Physics","winner":"Marie Curie"}
 {"yr":1904,"subject":"Physics","winner":"Lord Rayleigh"}
(26 rows)
```

window function

```
[...] window functions [...]

-- Guyren
```

window function

- raggruppiamo per soggetto:

```
select subject, count(*) from nobel where yr < 1905 group by subject;

  subject   | count
------------+-------
 Medicine   |     4
 Chemistry  |     4
 Physics    |     7
 Peace      |     6
 Literature |     5
(5 rows)
```


PGtraining

window function

- OK, ma se voglio "vedere" all'interno di ciascun gruppo? così non va:

```
select subject, count(*), winner from nobel where yr < 1905 group by subject;

ERROR:  column "nobel.winner" must appear in the GROUP BY clause or be
used in an aggregate function
LINE 1: select subject, count(*), winner from nobel where yr < 1905 ...
```


PGtraining

window function

- in realtà si potrebbe fare così:

```
select subject, count(*), string_agg(winner, ', ') from nobel
    where yr < 1905 group by subject;

  subject    | count |                                                 string_agg
-------------+-------+---------------------------------------------------------------- [...]
 Medicine    |     4 | Emil von Behring, Ronald Ross, Niels Ryberg Finsen, Ivan Pavlov [...]
 Chemistry   |     4 | Jacobus H. van 't Hoff, Emil Fischer, Svante Arrhenius, Sir Wil [...]
 Physics     |     7 | Wilhelm Conrad Röntgen, Hendrik A. Lorentz, Pieter Zeeman, Henr [...]
 Peace       |     6 | Henry Dunant, Fréric Passy, Éie Ducommun, Albert Gobat, Randal  [...]
 Literature  |     5 | Sully Prudhomme, Theodor Mommsen, Bjøjerne Bjøn, Joséchegaray,  [...]
(5 rows)
```

- questo perché PostgreSQL dispone di una funzione `string_agg()`

window function

- c'è un modo più generico: se aggiungo una frase OVER() a und funzione di aggregazione, creo una window function. viene valutata nella sua "finestra" definita da OVER():

```
select *, count(*) over() from nobel where yr < 1905;

  yr  |  subject    |              winner              | count
------+------------+----------------------------------+-------
 1901 | Chemistry  | Jacobus H. van 't Hoff           |    26
 1902 | Chemistry  | Emil Fischer                     |    26
 1903 | Chemistry  | Svante Arrhenius                 |    26
 [..]
 1903 | Physics    | Henri Becquerel                  |    26
 1903 | Physics    | Pierre Curie                     |    26
 1903 | Physics    | Marie Curie                      |    26
 1904 | Physics    | Lord Rayleigh                    |    26
(26 rows)
```

window function

- ho la possibilità di creare delle partition - sono come i group, ma senza l'aggregazione dei record:

```
select *, count(*) over (partition by subject) from nobel where yr < 1905;

  yr  |  subject    |              winner               | count
------+-------------+-----------------------------------+-------
 1901 | Chemistry   | Jacobus H. van 't Hoff            |     4
 1902 | Chemistry   | Emil Fischer                      |     4
 1903 | Chemistry   | Svante Arrhenius                  |     4
 1904 | Chemistry   | Sir William Ramsay                |     4
 1901 | Literature  | Sully Prudhomme                   |     5
 1902 | Literature  | Theodor Mommsen                   |     5
 [...]
 1903 | Physics     | Henri Becquerel                   |     7
 1903 | Physics     | Pierre Curie                      |     7
 1903 | Physics     | Marie Curie                       |     7
 1904 | Physics     | Lord Rayleigh                     |     7
(26 rows)
```

window function

- all'interno delle partition posso definire un ordinamento e oltre le classiche funzioni di aggregazioni ho window function dedicate che lavorano su questo ordinamento, come per esempio `rank()` (ma ce ne sono molte altre):

```
select subject, rank() over (partition by subject order by yr), winner
    from nobel where yr < 1905;

  subject    | rank |              winner
------------+------+--------------------------------
[...]
 Physics     |    1 | Wilhelm Conrad Röntgen
 Physics     |    2 | Hendrik A. Lorentz
 Physics     |    2 | Pieter Zeeman
 Physics     |    4 | Henri Becquerel
 [..]
(26 rows)
```

procedure con PL/PgSQL

```
[...] if the processes require the modification of various
tables within a transaction you may probably prefer to
expose functions as the application interface.

-- Charles
```

procedure con PL/PgSQL

- esempio: booking engine
- è facile che il database sia al centro e gli applicativi (e interfacce) continuano a mutare
- perché non usare una funzione PL/PgSQL da richiamare?
    - nasconde i dettagli del DB allo sviluppatore
    - ci possiamo fidare della transazionalità (anche se lo sviluppatore del front end non trappa tutte le situazioni di errore

procedure con PL/PgSQL

```
create table room (
    cat       varchar not null,
    dt        date not null,
    cnt       int not null,
              unique (cat, dt),
              check (cnt >= 0)
);

insert into room values
    ('tenda', '2016-09-01', 5),
    ('tenda', '2016-09-02', 5),
    ('tenda', '2016-09-03', 5),
    ('tenda', '2016-09-04', 5);

create table booking (
    id        serial primary key,
    f_night   date not null,
    l_night   date not null,
    info      varchar not null,
              check (f_night <= l_night),
              check (length(trim(info)) > 1)
);
```

procedure con PL/PgSQL

```
create function book(b_data json) returns varchar as
$$ declare
        f   date := (b_data->>'f_night')::date;
        l   date := (b_data->>'l_night')::date;
        c   int;
    begin

        -- book all nights and count them
        with book_night as (
            update room set cnt = cnt - 1
                where cat = b_data->>'cat' and dt between f and l
                returning 1
        ) select count(*) into c from book_night;

        -- verify all nights of the stay could be booked
        if c != coalesce(l - f + 1, -1) then
            raise exception '(some) dates/categories unavailable';
        end if;

        -- insert the booking and return the booking id
        insert into booking (f_night, l_night, info)
            values (f, l, b_data->>'info') returning id into c;
        return c;
    end;
$$ language 'plpgsql';
```


PGtraining

procedure con PL/PgSQL

- lo sviluppatore passa il JSON con le info della prenotazione e i vari contraint e l'if nella procedura pensano al resto - siamo certi al 100% che una prenotazione viene eseguita del tutto o non eseguita per niente:

```
select book(
  '{"f_night":"2016-09-01","l_night":"2016-09-03","cat":"tenda","info":"chris"}'
);

 book
------
 1
(1 row)
```

procedure con PL/PgSQL

- risultato:

```
select * from room order by dt;

  cat  |     dt     | cnt
-------+------------+-----
 tenda | 2016-09-01 |   4
 tenda | 2016-09-02 |   4
 tenda | 2016-09-03 |   4
 tenda | 2016-09-04 |   5
(4 rows)

select * from booking;

 id |  f_night   |  l_night   | info
----+------------+------------+-------
  1 | 2016-09-01 | 2016-09-03 | chris
(1 row)
```

procedure con PL/PgSQL

- una chicca! security definer:

```
create function book(b_data json) returns varchar as
$$
[...]
$$ language 'plpgsql' security definer;
```

- security definer vuole dire che la funzione viene eseguita con i diritti di chi l'ha definita, non di chi la sta eseguendo

- posso in questo modo creare un utente web che non ha diritti su room o booking, ma può solo eseguire la funzione - è un pò come lo setuid in Unix!

procedure con PL/PgSQL

- security definer permette a ciccio di eseguire la funzione con i diritti di chris, senza poter accedere alle tabelle di chris:

```
select user;

 current_user
--------------
 ciccio
(1 row)

select book('{ [...] "cat":"tenda","info":"ciccio"}');

 book
------
 2
(1 row)

delete from room;

ERROR:  permission denied for relation room
```

'k thx bye ;)

Il contenuto di queste slide è (C) Chris Mair 2016 rilasciato con licenza Creative Commons BY-SA.