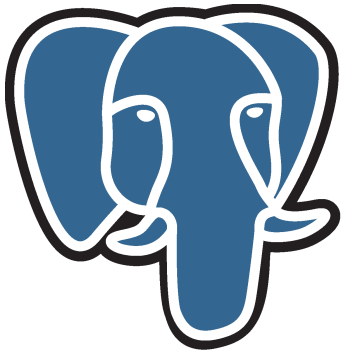


PostgreSQL

the world's most advanced open source database

In Tour con gli Elefanti

Linux Day 2014 - Montebelluna

Prima di partire...

Smart Solutions

Denis Gasparin

Senior DBA and Web Developer

- Sviluppo di soluzioni software basate su PostgreSQL
- Analista e Database Administrator
- Contributor del driver PDO PostgreSQL per PHP
- Membro di IT-PUG

Prepariamo le valigie!

- Installiamo PostgreSQL

```
$ sudo su -
$ vi /etc/apt/sources.list.d/pgdg.list

# Inserire questa riga nel file
deb http://apt.postgresql.org/pub/repos/apt/ wheezy-pgdg main

$ curl https://www.postgresql.org/media/keys/ACCC4CF8.asc |apt-key add -

$ apt-get update
$ apt-get install postgresql-9.3
$ su -l postgres
$ psql -U postgres template1
psql (9.3.5)
Digita "help" per avere un aiuto.

template1=#
```

- Le valigie sono pronte... partiamo!

Letture utili durante il Viaggio

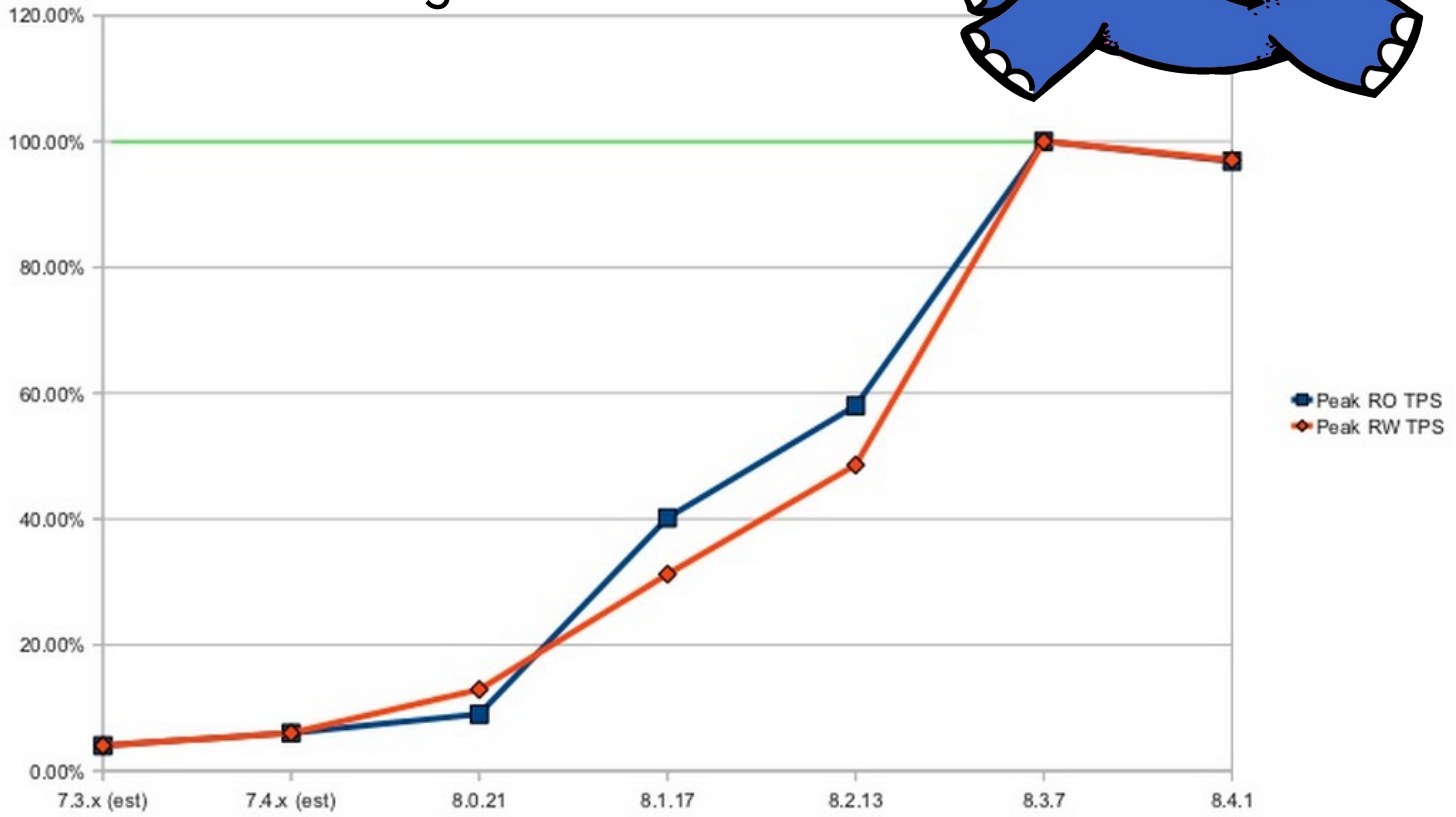
- **Psql:**
  - Client nativo per PostgreSQL
  - Completo e manutenuto insieme al backend
  - Facile da usare
  - Programmabile ed integrabile facilmente con la shell
- **PgAdmin:**
  - Client grafico per PostgreSQL
  - Editing delle tabelle e dei dati semplice e veloce
  - http://www.pgadmin.org

Si parte!

- Caratteristiche principali di PostgreSQL:
  - Semplice da installare e da configurare
  - Stabile
  - Affidabile
  - Veloce

PgSql 9.3

PgSql 7.3



http://suckit.blog.hu/2009/09/29/postgresql_history

Guardiamo il Panorama, presente e futuro!

- **PostgreSQL è Cross Platform:**
  - si installa su Linux, FreeBSD, Solaris, Mac OS X e Windows
  - anche su sistemi Embedded!
- **Disponibile anche su molte piattaforme Cloud:**
  - Amazon RDS, Heroku, RedHat OpenShift
- **Connettività con altri database** (anche NO-SQL)
- **Nuovi datatypes**: JSON, RANGE, HSTORE
- **Scalabilità orizzontale**: data partitioning e data sharding
- **Affidabilità**: replica master-slave e multi-master (in futuro)

@ https://twitter.com/Astro_Elephant

… Non finisce qui!

- **Estensibile:**
  - Repository di estensioni su http://pgxn.org/
- **Compliant agli standard:**
  - ANSI-SQL:2008
  - ACID: http://it.wikipedia.org/wiki/ACID
- **Licenza PostgreSQL:**
  - simile alla licenza BSD e MIT
  - Consente redistribuzione del software gratuitamente

Primi Passi con PostgreSQL e Psql

- Creiamo un utente che possa creare un database:

```
template1=# CREATE USER montellug WITH PASSWORD 'linuxday' CREATEDB SUPERUSER;
CREATE ROLE
```

- Creiamo un database:

```
$ psql -h 127.0.0.1 -U montellug template1
Sei collegato al database "template1" con nome utente "montellug".

template1=# CREATE DATABASE elephants_tour
CREATE DATABASE
```

- Ci colleghiamo al database:

```
template1=# \c elephants_tour montellug
Sei collegato al database "elephants_tour" con nome utente "montellug".

elephants_tour=# \dt  (visualizziamo le tabelle nel database appena creato)
Nessuna relazione trovata
```


Primi Passi con PostgreSQL e Psql

- Creiamo una tabella:

```
elephants_tour=# CREATE TABLE posto_visitato (
    id                  serial NOT NULL PRIMARY KEY,
    nome                text NOT NULL,
    quando              date NOT NULL,
    tipo                text CHECK (tipo IN('hotel','residence','campeggio')),
    durata_soggiorno    interval hour to minute
);
CREATE TABLE
```

- Inseriamo alcuni dati:

```
INSERT INTO posto_visitato (nome, quando, tipo, durata_soggiorno)
    VALUES
    ('Montebelluna', '2014-10-25', 'hotel', '00:40'),
    ('Castelfranco', '2014-10-26', 'residence', '02:00'),
    ('San Martino di Lupari', '2014-10-27', 'campeggio', '12:00'),
    ('Tombolo', '2014-10-28', 'campeggio', '12:00'),
    ('Galliera Veneta', '2014-10-29', 'campeggio', '12:00');
```


Ho sbagliato strada, posso tornare indietro?

- **Certo! Puoi usare una Transazione!**

```
elephants_tour=# BEGIN;
BEGIN

elephants_tour=# DELETE FROM posto_visitato WHERE nome = 'Montebelluna';
DELETE 1

elephants_tour=# ROLLBACK;
ROLLBACK

– Montebelluna è ancora al suo posto!
```

```
elephants_tour=# BEGIN;
BEGIN

elephants_tour=# DELETE FROM posto_visitato WHERE nome = 'Tombolo';
DELETE 1

elephants_tour=# COMMIT;
COMMIT
– Abbiamo cancellato Tombolo!
```


Le Transazioni ed i comandi DDL

- DDL = Data Definition Language (comandi CREATE e DROP TABLE, etc)

PostgreSQL

```
db=# BEGIN;
BEGIN

db=# DROP TABLE posto_visitato;
DROP TABLE;

db=# ROLLBACK;
ROLLBACK

db=# \dt
```

MySQL

```
mysql> BEGIN;
Query OK, 0 rows affected (0,00 sec)

mysql> DROP TABLE posto_visitato;
Query OK, 0 rows affected (0,01 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0,01 sec)

mysql> SHOW TABLES;
```

Che risultato vi aspettate?

```
              Lista delle relazioni
 Schema |      Nome      |  Tipo   | Proprietario
--------+----------------+---------+-----------
 public | posto_visitato | tabella | montellug
```

```
Empty set (0,00 sec)
```

Le Transazioni ed i comandi DDL

- **PostgreSQL esegue tutti i comandi DDL in transazione**
  - Consente di annullare eventuali cambiamenti in ogni momento
  - Utile in caso di script SQL per migrazioni e/o conversioni di un DB
    - Si è certi che la migrazione o va a buon fine o fallisce
    - Compliance ACID

Che bel Tipo sto Elefante: JSON

- **JSON:**
  - Javascript Object Notation: trasformazione in stringa di un oggetto JS
  - Principali utilizzi:
    - Chiamate AJAX di una pagina Web
    - Richieste e Risposte nei servizi REST
    - Molti database NO-SQL sono interrogati via JSON

```
elephant_tours=# BEGIN;
BEGIN

elephant_tours=# ALTER TABLE posto_visitato ADD COLUMN dati json;
ALTER TABLE;

elephant_tours=# COMMIT;
COMMIT

db=# \d posto_visitato
```


Che bel Tipo sto Elefante: JSON

```
UPDATE posto_visitato SET dati = '
    { "sindaco": {"nome": "Marzio", "cognome": "Favero", "eta": 49},
      "abitanti": 30845,
      "altitudine": 109
    }' WHERE nome = 'Montebelluna';

UPDATE posto_visitato SET dati = '
    { "sindaco": {"nome": "Jerry", "cognome": "Boratto", "eta": 44},
      "abitanti": 13233,
      "altitudine": 40
    }' WHERE nome = 'San Martino di Lupari';


SELECT * FROM posto_visitato WHERE cast(dati->>'abitanti' AS integer) > 15000;

SELECT SUM(cast(dati->>'abitanti' AS integer)) FROM posto_visitato;

SELECT * FROM posto_visitato WHERE cast(dati->'sindaco'->>'eta' AS integer) >
45;

WITH s AS (SELECT nome ,quando FROM posto_visitato)
    SELECT array_to_json(array_agg(row_to_json(s)),true) FROM s;
```


Che bel Tipo sto Elefante: RANGE

- **RANGE:**
  - Consente di rappresentare un intervallo di valori di un certo tipo
  - Tipi supportati: interi, numeric, timestamps e date

```
CREATE TABLE prenotazione(
    id serial NOT NULL PRIMARY KEY,
    giorni daterange
);

INSERT INTO prenotazione(giorni) VALUES
    ('[2014-10-01, 2014-10-10)'),
    ('[2014-10-11, 2014-10-25)'); – notare le parentesi [ e )

– range contiene una data
SELECT * FROM prenotazione WHERE giorni @> '2014-10-05'::date;

– sovrapposizione di range
SELECT * FROM prenotazione WHERE giorni && '[2014-10-05, 2014-10-
15]'::daterange;
```

Common Table Expressions

- SQL è un linguaggio dichiarativo:
  - Con una query chiedo al database di ritornare dei dati ma non come farlo
- Common Table Expressions (CTE) trasformano SQL in linguaggio imperativo
  - Consentono di spezzare una query in parti più semplici
  - Ogni parte può essere una query di selezione ma anche di INSERT, UPDATE o DELETE
  - CTE ricorsive

Common Table Expressions

```
WITH regional_sales AS (
        SELECT region, SUM(amount) AS total_sales
        FROM orders
        GROUP BY region
    ),
    top_regions AS (
        SELECT region
        FROM regional_sales
        ORDER BY total_sales DESC LIMIT 10
    )
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;


WITH old_logs AS
    (DELETE FROM log WHERE date <='2014-06-01' RETURNING *)
INSERT INTO log_backup SELECT * FROM old_logs;
```


Window Functions

- Una window function è simile ad una funzione di raggruppamento
  - Effettua dei calcoli su un gruppo di righe (la cosiddetta finestra)
  - Non fa collassare le righe in un'unica riga
- La "finestra" viene specificata con le istruzioni "OVER" e "PARTITION BY"

```
– confrontare lo stipendio di un dipendente rispetto alla media del suo
dipartimento

SELECT nome, dipartimento, salario,
       avg(salario) OVER (PARTITION BY dipartimento)
FROM dipendente;

– ordinare i dipendenti per stipendio descrescente nello stesso dipartimento

SELECT nome, dipartimento, salario,
       rank() OVER (PARTITION BY dipartimento ORDER BY salario DESC)
FROM dipendente;
```

Foreign Data Wrappers

- Sono estensioni che consentono di collegare Postgres ad altri database o sorgenti dati
- https://wiki.postgresql.org/wiki/Foreign_data_wrappers
- Esempio: file data wrapper

```
$ apt-get install postgresql-contrib-9.3

CREATE EXTENSION file_fdw;
CREATE SERVER filesystem_server FOREIGN DATA WRAPPER file_fdw;

CREATE FOREIGN TABLE dipendente_da_csv (
   id integer NOT NULL,
   nome text NOT NULL,
   dipartimento text NOT NULL,
   salario numeric(7,2) NOT NULL
) SERVER filesystem_server
OPTIONS
(format 'text', filename '/tmp/dipendente.csv', delimiter E'\t', null '');
```

Viste Materializzate

- Le "Viste" in PostgreSQL sono una semplice riscrittura di una query

```
CREATE VIEW admins AS SELECT * FROM dipendente WHERE dipartimento='ADMIN';

EXPLAIN SELECT * FROM admins ;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on dipendente  (cost=0.00..19.00 rows=4 width=82)
   Filter: (dipartimento = 'ADMIN'::text)
```

- Le "Viste materializzate" memorizzano i dati in una tabella:
  - Query di selezione molto più veloci
  - Non sono consentite operazioni di INSERT, DELETE, UPDATE
  - Possono essere aggiornate globalmente

Viste Materializzate

```
CREATE MATERIALIZED VIEW m_admins AS SELECT * FROM dipendente WHERE
dipartimento='ADMIN';

EXPLAIN SELECT * FROM m_admins;
                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on m_admins  (cost=0.00..17.20 rows=720 width=82)

– Un insert nella tabella originale non viene “visto” nella vista
materializzata

INSERT INTO dipendente VALUES
    (15,'OBAMA','ADMIN',20000);

– Bisogna usare lo statemente “REFRESH”
REFRESH MATERIALIZED VIEW m_admins;
```

Viste Materializzate e Foreign Data Wrappers

- Le viste materializzate consento di velocizzare moltissimo l'accesso a sorgenti dati esterne:
  - Importano i dati nativamente in PostgreSQL
  - Le viste materializzate sono indicizzabili

```
CREATE MATERIALIZED VIEW m_dipendente_da_csv AS SELECT * FROM
dipendente_da_csv;

SELECT * FROM m_dipendente_da_csv;

CREATE INDEX idx_dipartimento ON m_dipendente_da_csv(dipartimento);
```

- Da PostgreSQL 9.4, le viste potranno essere aggiornate senza ottenere un LOCK esclusivo sulla vista materializzata

Sicurezza: backup e replica

- PostgreSQL offre innumerevoli **metodi di backup**:
  - **Logico**: dump SQL del database
    - pg_dump, pg_dumpall
  - **Fisico**:
    - Copia fisica della cartella con il DB e memorizzazione dei WAL log generati dalle transazioni
    - Consente il ripristino in un certo momento nel tempo
  - **Replica**:
    - Un database master può essere replicato su diverse istanze slave
    - Molto veloce ed ottimizzato anche per replica geografica

Non viaggiate mai da soli!

- La comunità di PostgreSQL è molto attiva
  - Potete trovare aiuto nelle mailing list
    - http://www.postgresql.org/list/
  - Oppure nel canale IRC
    - http://www.postgresql.org/community/irc/
- In Italia esiste IT-PUG, fondato nel 2007
  - Organizza il PG-DAY Italiano
  - Mailing List in Italiano
  - http://www.itpug.org

Ci vediamo a Prato – 7 Novembre 2014!

Iscrivetevi! http://2014.pgday.it/

Grazie!

Smart Solutions

denis@gasparin.net

http://www.gasparin.net