The world's most advanced class on PostgreSQL... o quasi!

4 Marzo - 8 Aprile 2015

L'Antipasto - 11 Marzo

GrappaLUG

Gruppo utenti GNU/Linux di Bassano del Grappa

Smart Solutions

IT PUG

Riepilogo della puntata precedente

```
####################### Avviamo la macchina virtuale ######################
$ vagrant up
$ vagrant ssh

########################### Creazione del database ###########################
$ sudo su -
$ su -l postgres
$ psql -U postgres template1

Psql (9.4.1)
Digita "help" per avere un aiuto.

template1=# CREATE DATABASE corso;
CREATE DATABASE

template1=# \c corso
You are now connected to database "corso" as user "postgres".

corso=# \l -- visualizzazione dei database del cluster
```

Utenti

- Abbiamo creato un database utilizzando il superutente predefinito “postgres”
- Normalmente i database vengono creati con un utente specifico in base ad alcuni criteri:
  - Tipologia di applicazione
  - Gestione dei permessi di accesso
- Tipologia di utenti:
  - SUPERUSER: può fare qualsiasi cosa… usare con cautela!
  - CREATEDB: può creare database
  - CREATEROLE: può creare altri ruoli
  - LOGIN: può autenticarsi al database
  - REPLICATION: utente che può avviare la streaming replication

Creiamo un utente!

Psql ci avverte che siamo superutenti!

```
corso=# \help CREATE USER -- \help visualizza l'help sql contestuale!
Command:     CREATE USER
Description: define a new database role
Syntax:
CREATE USER name [ [ WITH ] option [ ... ] ]
###### Output troncato ######

corso=# CREATE USER utente WITH PASSWORD 'password';
CREATE ROLE

corso=# CREATE TABLE test(a INTEGER);
CREATE TABLE

corso=# \c – utente -- ci connettiamo al db come utente “utente”
FATAL:  Peer authentication failed for user "utente"
Previous connection kept
```

Come mai?

- La sicurezza su PostgreSQL è a due livelli:
  - Connessione: file pg_hba.conf
  - Permessi sugli oggetti del db: CREATE USER/ROLE
- A cos'è dovuto l'errore di prima?
  - pg_hba.conf

File pg_hba.conf

- Con un editor di testo modifichiamo il file:

```
$ vi /etc/postgresql/9.4/main/pg_hba.conf

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
host    all             all             127.0.0.1/32            md5
```

- Type: local (unix socket), host (tcp socket), hostssl
- Database
- User
- Address
- Method: trust/peer/md5/password/ldap/radius/pam

Esercizio 1

- Modificare il file pg_hba.conf per accettare connessioni dall'utente creato precedentemente
- Utilizzare psql per collegarci al db
- Creare una tabella `test_utente` con un unico campo intero
- Eseguire il comando SQL SELECT per selezionare dei record dalla tabella

Suggerimento: Ricordiamoci il reload di PostgreSQL!

Gestione dei permessi sugli oggetti

Psql ci avverte che siamo utenti normali

```
corso=> \dt -- visualizziamo le tabelle del db
            List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | test        | table | postgres
 public | test_utente | table | utente
(2 rows)

corso=> \d test_utente
  Table "public.test_utente"
 Column |  Type   | Modifiers
--------+---------+-----------
 a      | integer |

corso=> SELECT * FROM test;
ERROR:  permission denied for relation test
```

Esercizio 2

- Utilizziamo il comando SQL GRANT per dare accesso all'utente “utente” alla tabella “test” creata dal superutente “postgres”
- Suggerimenti:
  - \help...
  - \c...

Creazione di tabelle

- Diversi modi:
  - CREATE TABLE
  - CREATE TABLE AS
  - SELECT … INTO … FROM
- \help CREATE TABLE

Tipi di tabelle

- Temporanee
  - Eliminate al termine della sessione o della transazione
- Unlogged:
  - Dati non scritti nel WAL
  - Non sono replicate e non sopravvivono ad un crash!
- Eriditarietà:
  - Utilizzato per il partizionamento

Esempio di creazione di una tabella

```
corso=> CREATE TABLE regione(
   id INTEGER NOT NULL PRIMARY KEY,
   regione TEXT NOT NULL UNIQUE
);

corso=> CREATE TABLE provincia(
   id INTEGER NOT NULL PRIMARY KEY,
   provincia TEXT NOT NULL UNIQUE,
   regione INTEGER NOT NULL REFERENCES regione(id)
);

corso=> CREATE TABLE comune(
   id SERIAL NOT NULL PRIMARY KEY,
   comune TEXT NOT NULL,
   provincia INTEGER NOT NULL REFERENCES provincia(id)
);

corso=> \i dati_geografici.sql
```

Esercizio 3

- Creiamo una tabella anagrafica con i campi:
  - Id: intero seriale
  - Nome, cognome: testo
  - Codice fiscale: testo con vincolo di univocità
  - Data di inserimento: timestamp
- Popolare la tabella con il comando \copy
- \x
- \e, \p, \s, \w

Inserimento, modifica, cancellazione di record

```
corso=> \help INSERT

Description: create new rows in a table
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT INTO table_name [ ( column_name [, ...] ) ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] )
[, ...] | query }
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]

corso=> \help UPDATE

Description: update rows of a table
Syntax:
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [ ONLY ] table_name [ * ] [ [ AS ] alias ]
    SET { column_name = { expression | DEFAULT } |
          ( column_name [, ...] ) = ( { expression | DEFAULT }
[, ...] ) } [, ...]
    [ FROM from_list ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]

corso=> \help DELETE
```

Esercizio 4

- Cancellare la regione con nome “Regione inesistente”
- Modificare la provincia “Vincenza” in “Vicenza”
- Inserire il comune mancante “Bassano del Grappa”
- Con una query otteniamo il seguente risultato...

Tipi di dato in PostgreSQL

- Numerici
- Monetari
- Stringhe
- Binari
- Date/Time
- Booleani
- Enumerazioni
- Geometrici
- Xml
- Json
- Array
- Composite Types
- Range Types

Grazie!

denis@gasparin.net
http://www.gasparin.net