

II DBMS PostgreSQL

Laboratorio di basi di dati
Marco Frasca, Stefano Montanelli
Dipartimento di Informatica
Università degli Studi di Milano
<http://islab.di.unimi.it/bdlab1>
{marco.frasca,stefano.montanelli}@unimi.it



Supporto allo standard

- PostgreSQL supporta lo standard SQL99
 - Complex queries, Foreign keys, Triggers, Views, Transactional integrity, Multiversion concurrency control
- Estensioni
 - Data types
 - Functions
 - Aggregate functions
 - Index methods
 - Procedural languages
 - Concurrency and transaction management
 - Il costrutto ASSERTION è implementato mediante uso di trigger
 - Le viste materializzate sono realizzate mediante tabelle e trigger

Cluster e istanze di PostgreSQL

- Un cluster è una collezione di basi di dati gestite da una singola istanza di PostgreSQL
 - il cluster è un'area del disco rigido destinata a memorizzare i file delle basi di dati presenti sul DBMS
- Sul medesimo elaboratore, è possibile installare più istanze di PostgreSQL
 - le istanze sono fra loro indipendenti
 - ogni istanza memorizza e gestisce la propria collezione di basi di dati

Controllo dell'accesso

- Un cluster è dotato di un file **pg_hba.conf** mediante il quale è possibile definire le politiche di accesso al DBMS con regole analoghe a quelle di un firewall
- Le modifiche al file hba.conf vengono recepite al successivo riavvio del DBMS
- hba.conf è composto da regole, la cui priorità segue l'ordine di definizione

Regole di accesso

Ogni regola ha la seguente struttura

- **type** = { local | host | hostssl }
 - indica se la regola si applica a connessioni local (socket unix), host (socket TCP/IP) o hostssl (socket TCP/IP + SSL)
 - Le connessioni local sono utilizzate solo su piattaforma UNIX. Su piattaforma Windows vengono sempre utilizzate connessioni host
- **database** = { all | sameuser | <lista_db> }
 - indica i database a cui si applica la regola
- **user** = { all | <lista_utenti> }
 - indica gli utenti a cui si applica la regola
- **cidr-address** = <indirizzo/i_IP>
 - indica l'indirizzo IP (o la famiglia di indirizzi IP) a cui si applica la regola
- **method** = { trust | reject | md5 | ident | pam }
 - azione da compiere quando la regola viene attivata

Metodi di accesso

I principali metodi previsti da hba.conf sono:

- **trust**: accesso consentito senza verificare le credenziali dell'utente
- **reject**: accesso negato
- **md5**: accesso consentito previa verifica delle credenziali dell'utente sul DBMS (credenziali trasmesse su canale criptato con MD5)
- **Ident/peer**: accesso consentito agli oggetti del DBMS su cui l'utente di sistema da cui arriva la chiamata ha privilegi
- **pam**: accesso consentito previa verifica delle credenziali dell'utente di sistema da cui arriva la chiamata mediante PAM

Esempio di hba.conf

TYPE	DB	USER	CIDR-ADDR	METHOD
host	all	stefano	127.0.0.1/32	trust
host	all	all	127.0.0.1/32	md5
host	all	all	0.0.0.0/0	reject

- Domande

- L'utente "stefano" può collegarsi da remoto alla base di dati "ospedale"?
- Quale accesso è previsto per l'utente "mario"?

Creazione di schemi di basi di dati

- Una base di dati viene creata con la seguente istruzione

```
CREATE SCHEMA [NomeSchema]
[AUTHORIZATION Username]
[{ElementoSchema}]
```

- **NomeSchema.** E' il nome dello schema creato. Se omissso, coincide con il nome dell'utente che ha lanciato il comando
- **Username.** E' il nome dell'utente proprietario dello schema. Se omissso, coincide con il nome dell'utente che ha lanciato il comando
- **ElementoSchema.** Permette di specificare le strutture appartenenti allo schema

Contenuto di uno schema

- All'interno di uno schema possono essere creati i seguenti elementi (ElementoSchema nella slide precedente)
 - Domini (CREATE DOMAIN)
 - Tabelle (CREATE TABLE)
 - Asserzioni (CREATE ASSERTION)
 - Viste (CREATE VIEW)
 - Creazione utenti (CREATE USER)
 - Privilegi (GRANT / REVOKE)

L'istruzione CREATE DATABASE

- Oltre all'istruzione CREATE SCHEMA, i DBMS possiedono anche l'istruzione CREATE DATABASE che tuttavia NON è un'istruzione standard di SQL

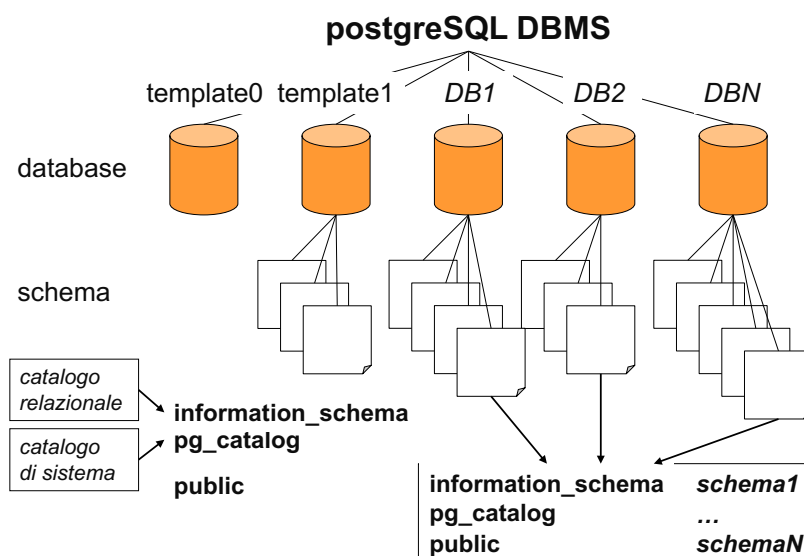
```
CREATE DATABASE NomeDB  
[ [WITH] [OWNER [=] Username]  
[ ENCODING [=] encoding ] ]
```

- **NomeDB.** E' il nome della base di dati creata
- **Username.** E' il nome dell'utente proprietario della base di dati. Se omissso, coincide con il nome dell'utente che ha lanciato il comando
- **Encoding.** Indica la codifica dei caratteri da utilizzare nella base di dati (e.g., SQL_ASCII, UTF8)

Schema vs. database

- La relazione fra database e schema dipende dal DBMS su cui tali oggetti sono creati
- Esempi
 - Oracle Express Edition
 - Esiste un unico database (non esiste CREATE DATABASE) in cui sono creati tutti gli schemi (CREATE SCHEMA)
 - PostgreSQL
 - possono essere creati più database (CREATE DATABASE) e ogni database può contenere più schemi (CREATE SCHEMA)

Schema vs. database - PostgreSQL



Schema vs. database - PostgreSQL

- Nella prassi, per creare una base di dati in postgresQL utilizzare il comando CREATE DATABASE
- Nel database è possibile creare schemi con il comando CREATE SCHEMA
- Se non si creano schemi, gli elementi della base di dati saranno inseriti nello schema public

Catalogo relazionale

- Il catalogo relazionale fornisce una descrizione degli elementi dello schema che costituiscono la base di dati
- Nei DBMS relazionali il catalogo è costituito a sua volta da una base di dati (riflessività)
- In PostgreSQL il catalogo relazionale si chiama information_schema
- In ogni database PostgreSQL esiste anche uno schema chiamato pg_catalog. Questo è un catalogo di sistema (definito secondo un modello proprietario) che contiene i built-in data type, le funzioni e gli operatori previsti dal DBMS

Creazione di utenti

- Lo standard SQL lascia all'implementazione la definizione del comando per la creazione di utenti
- PostgreSQL realizza la creazione di utenti mediante CREATE USER e CREATE ROLE che sono fra loro alias

```
CREATE USER <nome_utente> [ [ WITH ] option [ ... ] ]
```

elenco di opzioni disponibili (in corsivo le opzioni predefinite):

```
SUPERUSER | NOSUPERUSER |  
CREATEDB | NOCREATEDB |  
CREATEROLE | NOCREATEROLE |  
[ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password' |  
VALID UNTIL 'timestamp'
```

Creazione di tabelle

- L'istruzione per la creazione di tabelle è conforme allo standard SQL

```
CREATE TABLE NomeTabella (  
    NomeAttributo Dominio [ValoreDefault] [Vincoli]  
    {, NomeAttributo Dominio [ValoreDefault] [Vincoli]}  
    [AltriVincoli]  
)
```


Tipi di dato (domini) elementari

Numerico esatto (virgola fissa)	Intero	Integer
		Smallint
	Intero e decimale	Numeric
		Decimal
Numerico approssimato (virgola mobile)	Real	Non supportano confronti di uguaglianza/disuguaglianza sui valori memorizzati
	Double precision	
	Float	
Testuale	Character (char)	
	Character varying (varchar)	
Booleano	Bit, Boolean	
	Bit varying	
Temporale	Date	
	Time	
	Timestamp	

Laboratorio di basi di dati, Università degli Studi di Milano

17

Ulteriori tipi di dato – BLOB e CLOB

- Consentono di rappresentare oggetti di grandi dimensioni (e.g., dati multimediali, semistrutturati)
 - sequenza di valori binari (blob, binary large object)
 - sequenza di caratteri (clob, character large object)
- Il sistema garantisce la memorizzazione, ma non è possibile utilizzare il valore come criterio di selezione per le ricerche e le interrogazioni
- Sono implementati con caratteristiche diverse a seconda del sistema
- In alternativa a BLOB e CLOB, è possibile salvare questi oggetti in una cartella specifica del file system e memorizzare nella base di dati il path che punta al file
 - ↑ gestione più semplice dal punto di vista della base di dati
 - ↓ problemi di sicurezza/omonimia per la gestione dei file

Laboratorio di basi di dati, Università degli Studi di Milano

18

BLOB e CLOB in PostgreSQL

- Esistono due strategie per memorizzare oggetti BLOB in PostgreSQL
 - tipo di dato **bytea**. L'oggetto è memorizzato come sequenza di byte. Richiede l'uso di funzioni di streaming per caricare/accedere i valori inseriti (metodo suggerito per oggetti di piccole dimensioni)
 - tipo di dato **oid**. L'oggetto è memorizzato in una tabella separata (e gestita dal DBMS) mentre l'attributo memorizza un id che punta all'oggetto (metodo suggerito per oggetti di grandi dimensioni)
- Per memorizzare oggetti CLOB utilizzare il tipo di dato **text** che può contenere una stringa di lunghezza variabile senza limite superiore (salvo restrizioni di tipo disk quota)

Il tipo di dati *serial* di PostgreSQL

- PostgreSQL mette a disposizione il tipo di dato *serial*
- Si tratta di un tipo di dato numerico intero su cui è definita una *sequenza di passo 1*
- E' molto utile per realizzare attributi contenenti identificatori numerici autogenerati
- Nonostante sia di utilizzo molto comune, NON è un tipo di dato standard

Esempi

- Quale tipo di dato è più appropriato per ciascuno dei seguenti casi?
 - matricola di studente universitario
 - indirizzo di residenza di un cliente
 - stipendio di un dipendente
 - fotografia di un edificio
 - contatore degli accessi ad un sito
 - contratto di lavoro
 - una codifica per uno dei seguenti valori: Milano, Roma, Firenze, Torino
 - ultima connessione di un utente applicativo
 - poltrone libere/occupate in una sala cinematografica

Dump di una base di dati PostgreSQL

- E' l'istruzione per esportare una base di dati ospitata su un DBMS
- In PostgreSQL, il dump di una base di dati è possibile dal prompt dei comandi mediante l'utility **pg_dump** utilizzando la seguente istruzione:

```
pg_dump [opzioni] [ -f <path_output_file> ] <nome_db>
          [-h <host>] [-U <utente>]
```

- Come risultato, la base di dati <nome_db> viene esportata nel file <path_output_file>
- Mediante le opzioni è possibile configurare il comportamento di pg_dump. Per esempio, è possibile specificare se salvare solo lo schema o solo le istanze di <nome_db>. Consultare il manuale per maggiori dettagli
- In aggiunta a pg_dump, l'utility **pg_dumpall** consente il dump di un intero cluster (tutte le basi di dati ospitate su un DBMS)

Restore di una base di dati PostgreSQL

- L'istruzione di restore si realizza invocando il client psql e ridirigendo l'input

```
psql <nome_db> [-h <host>] [-U <utente>] <
    <path_input_file>
```

- Le istruzioni contenute nel file <path_input_file> vengono eseguite nella base di dati <nome_db>
- <path_input_file> è il risultato di una precedente istruzione dump
- E' necessario che il database vuoto ed eventuali utenti associati a <nome_db> siano già stati creati sul DBMS prima di lanciare il comando
- Il medesimo risultato si ottiene con il comando **pg_restore**

Piano di esecuzione dei comandi SQL

- Il comando EXPLAIN di PostgreSQL consente di visualizzare e analizzare il piano di esecuzione di un comando SQL
- Il piano di esecuzione è la rappresentazione della sequenza di istruzioni che saranno eseguite internamente al DBMS per arrivare al risultato del comando

```
EXPLAIN [ ANALYZE ] [ VERBOSE ] comando_SQL
```

- **analyze:** esegue il comando SQL e mostra i tempi di esecuzione delle varie istruzioni che compongono il piano di esecuzione
- **verbose:** mostra una rappresentazione del piano di esecuzione molto dettagliata (utile per fare debugging)

Piano di esecuzione dei comandi SQL

- Il comando EXPLAIN è utile per valutare le prestazioni di un'interrogazione SQL e comparare l'efficienza di soluzioni alternative alla medesima richiesta

```
SELECT cliente
FROM acquisto
WHERE prodotto = xxx AND
cliente IN (SELECT cliente
            FROM acquisto
            WHERE prodotto = yyy)
```

```
SELECT cliente
FROM acquisto
WHERE prodotto = xxx
INTERSECT
SELECT cliente
FROM acquisto
WHERE prodotto = yyy
```

- quale delle due soluzioni è più efficiente?
analizzare con explain...