Laboratorio di Sistemi Informativi

[tomo 1]

Master in Tecnologie del Software Libero e Open Source

Stefano Zacchiroli
zack@cs.unibo.it

Outline del corso

Obiettivi e topic

“(R)DBMS per sysadm”

- Aspetti sistemistici
  - amministrazione di DBMS stand-alone ed embedded
  - installazione, configurazione iniziale, controlli di accesso, manutenzione ordinaria e straordinaria, performance tuning
    - tecnologie software: MySQL, PostgreSQL, SQLite
  - installazione di applicazioni che necessitano di un DBMS “generico”
    - tecnologie software: Debian's dbconfig-common

Obiettivi e topic

“(R)DBMS per sviluppatori”

- Aspetti di sviluppo software
  - astrazioni ed API per lo sviluppo di applicativi che usano DBMS per data storage
    - tecnologie software: Perl DBI, Python SQL Object

DBMS per sysadm

info pratiche per l'uso dei DBMS in questo corso, nel nostro laboratorio

Punti di vista

- il punto di vista “anomalo” dei sistemisti
  - i DBMS come dipendenze di altri applicativi, non prodotti finali
- una dicotomia
  1. DBMS stand-alone
     - *servizi* esposti agli utenti (localmente o via rete)
     - data storage centralizzato e gestito dal DBMS, namespace DBMS-specific
  2. DBMS embedded
     - librerie linkate a runtime con altri applicativi
     - data storage sul filesystem e gestito dall'applicazione, locazione application-specific

DBMS deployment

- i DBMS stand-alone sono software complessi
  - tratteremo separatamente i DBMS embedded
- alto grado di coupling con il sistema operativo
  - servizi per molti utenti (accesso locale in multiutenza e via rete)
  - consumo di risorse (memoria, cpu, processi)
  - accesso privilegiato al file system
  - scheduling della manutenzione ordinaria
- la loro amministrazione richiede privilegi
  - e.g. root (a volte meno)

DBMS “virtualizzati”

- per sperimentare l'amministrazione di DBMS useremo macchine virtuali in cui siamo root
  - la nostra scelta: User Mode Linux
- ognuno di noi è root e potrà affrontare tutte le problematiche di amministrazione di DBMS standalone
- abbiamo già kernel UML e file system
- accesso alle macchine virtuali UML
  - sulle macchine host del laboratorio
  - sui server:
    - per{lopiu,lomeno,altro}.m-fosset.almaweb.unibo.it

UML in lab HOWTO

```
( ssh perlopiu.m-fosset.almaweb.unibo.it )

ln -s /home/users/corsi/labrs/linux ~/linux

uml_mkcow cow /home/users/corsi/labrs/root_fs

./linux ubda=cow mem=512M eth0=daemon,,,/var/run/uml-
  utilities/uml_switch.ctl
```

- oppure

```
cp /home/users/corsi/labrs/uml.sh ~/

./uml.sh
```

- o anche

```
/home/users/corsi/labrs/uml.sh
```

UML in lab HOWTO (cont.)

- le istanze UML
  - login “root” (no password)
  - connettività di rete (NAT-like)
  - Ubuntu Feisty (main e universe)
- useremo principalmente i seguenti pacchetti (già installati):
  - mysql-server-5.0
  - postgresql-8.2
  - sqlite3

UML - esercizi

- provate!

DBMS per sysadm

installazione e setup iniziale

Installazione

- (fortunatamente) non è più necessario compilare un DBMS per installarlo
  - milioni di righe di codice ...
- nelle distribuzioni GNU/Linux i grandi attori sono pacchettizzati
  - analizzeremo il casto di distribuzioni Debian-based
- l'installazione consiste quindi in

1. installazione dei pacchetti
2. (post installation setup manuale)
3. creazione di utenti e database
4. (goto 3)

MySQL - Pacchetti

- pacchetto sorgente “mysql-dfsg-5.0”, binari:
  - mysql-server*
    - server MySQL: demone standalone
  - mysql-client*
    - top-level interattivo per query (“mysql”)
    - tool di amministrazione cmdline (“mysqladmin”)
  - libmysqlclient15*, libmysqlclient*-dev
    - librerie (shared e non) per l'accesso via API nativa
  - mysql-common
    - shared stuff, e.g. file di configurazione /etc/mysql/*

MySQL – Pacchetti (cont.)

- attenzione: pacchetti vs metapacchetti
  - pianificate le vostre politiche di upgrade!
  - e.g. mysql-server-5.0 vs mysql-server
- documentazione?
  - non nella distro per problemi di licenza
  - su web MySQL reference manual (version-specific) http://dev.mysql.com/doc/refman/5.0/en/
- bells and whistles
  - mysql-admin: GUI-based administration tool
  - phpmyadmin: web-based administration tool

MySQL – installation HOWTO

- facile:

```
aptitude install mysql-server
```

  - oppure

```
aptitude install mysql-server-5.0
```

- ...
- post installazione
  - leggere /usr/share/doc/mysql-server-5.0/README.Debian.gz (!!!)

MySQL – post installation

- la fasa di post-installazione è in buona parte delegata al pacchetto
  - dbms bootstrap
  - creazione di utenti per la manutenzione ordinaria periodica
  - ...
- manualmente è richiesto il setup della “password di root”
  - senza: tutti gli utenti possono accedere come “root”!

```
/usr/bin/mysqladmin -u root password 'new-password'
```

MySQL – post installation (cont.)

- per comodità, è consigliabile salvare la password dell'utente root (MySQL) come configurazione dell'utente root (sistema)
- il file di configurazione “$HOME/my.cnf”:

```
# an example of $HOME/.my.cnf
[client]
user = “username”
password = “new-password”
```

MySQL – il servizio di sistema

- mysql server è compatibile con la “API” Debian per i servizi di sistema
  - servizio “mysql”
  - script /etc/init.d/mysql

```
Usage: /etc/init.d/mysql start|stop|restart|reload|
  force-reload|status

invoke-rc.d mysql-server start

invoke-rc.d mysql-server stop

invoke-rc.d mysql-server ...
```

- soggetto alla configurazione dei runlevel

MySQL - configurazione

- dopo l'installazione di un servizio ogni buon sysadm si domanda: *« dov'è il file di configurazione? »*
- risposta: /etc/mysql/my.cnf
  - la suite MySQL include diversi eseguibili
    - e.g.: mysql, mysqladmin, mysqld, mysqld_safe, ...
  - ogni parametro di configurazione di un eseguibile può essere specificato a cmdline o inserito in my.cnf
  - my.cnf è diviso in gruppi ini-like (“[gruppo]”)
  - ogni eseguibile legge la conf di uno o più gruppi
  - in Debian anche: /etc/mysql/conf.d/*
  - http://dev.mysql.com/doc/refman/5.0/en/program-options.html

MySQL – configurazione (cont.)

- il servizio MySQL è incarnato dal demone “mysqld”
  - (e dallo script mysqld_safe che lo “sorveglia”...)
  - legge i gruppi “mysqld” e “server”
- http://dev.mysql.com/doc/refman/5.0/en/server-options.html
- tipologie di configurazioni per mysqld
  - data dir, runtime dir, ...
  - cache size e log level
  - networking
  - ...

MySQL – root password

- cookbook per 2 problemi comuni
  1. *cambiare la password di root*

     ```
     /usr/bin/mysqladmin -u root password 'new-password'
     ```

  2. *resettare la password di root* (persa)

http://dev.mysql.com/doc/refman/5.0/en/resetting-permissions.html

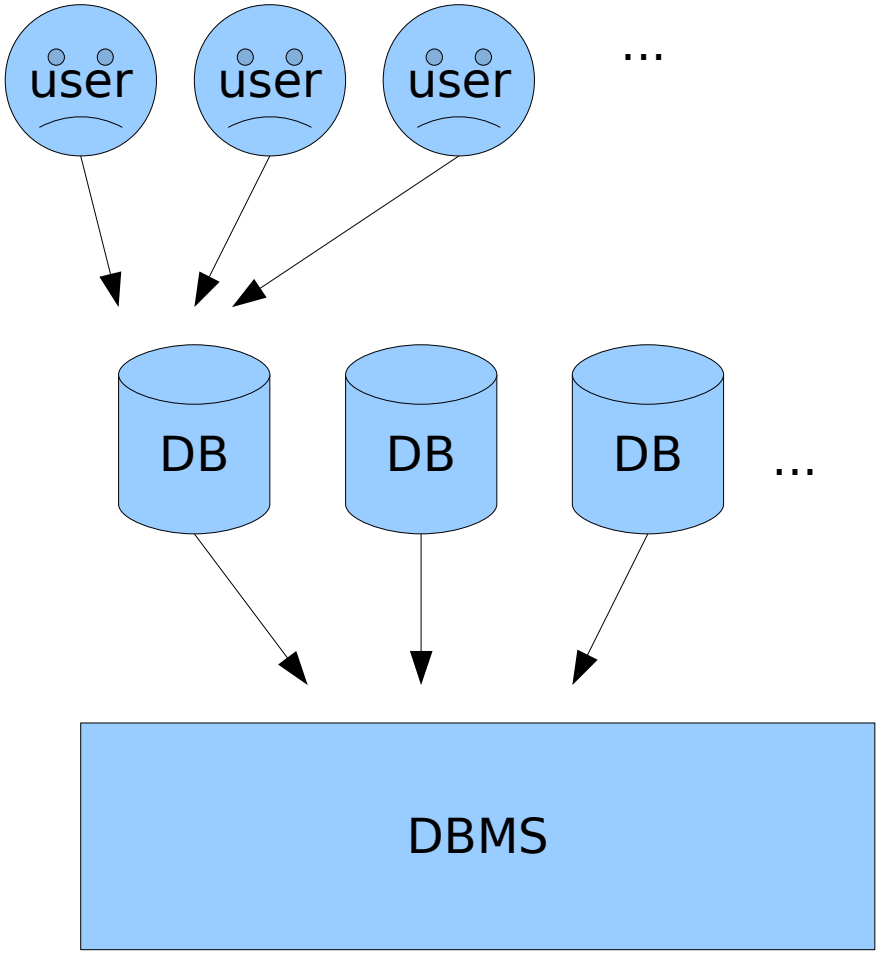
```
invoke-rc.d mysql stop
SET PASSWORD FOR 'root'@'localhost' =
  PASSWORD('MyNewPassword');   # text file ~/reset
mysqld_safe --init-file=~/reset &
rm ~/reset
```

DBMS per sysadm

uso, management, controllo di accesso

1 DBMS, molti DB, 100000 utenti

- usi tipici di DBMS in F/OS includono
  - Web-app (LAMP)
  - applicazioni stand-alone con requisiti:
    - di prestazioni
    - di strutturazione dati
- 1 DBMS -> molti DB
  - DB namespace, piatto
- 1 DB -> molti utenti



Gestione dei DB

- DB diversi, per usi diversi di un DBMS
  - sono necessari meccanismi di creazione/distruzione dei vari DB che popolano il namespace di un DBMS
    - se ne occupa il DCL (Data Control Language)
      - helper tools (e.g. mysqladmin) costruiti sul DCL
- tanti utenti per un DB
  - per il *principio dei privilegi minimi* sono necessarie politiche (ACL) e meccanismi (controllo di accesso)
    - DCL + helper
  - tecnicamente è necessario permettere agli utenti di accedere localmente e/o via rete
    - configurazione del DBMS

MySQL – gestione dei DB

- creazione di database
  - create database db_name; -- dcl
  - mysqladmin create db_name -- helper
- rimozione di database
  - drop database db_name; -- dcl
  - mysqladmin drop db_name -- helper
- ispezione e uso dei database
  - show databases; -- mostra i db correnti
  - use db_name; -- passa al db db_name

Interazione con i DBMS

- gli utenti finali di DBMS difficilmente interagiscono direttamente con essi
  - le applicazioni mediano per loro e filtrano gli errori
    - accedeno al DBMS via API
  - gli utenti finali spesso non sanno neanche dell'esistenza di un DBMS!
- come amministratori (o programmatori) è spesso utile una interazione non mediata
  - motivazioni: test di funzionalità, debugging, auto-apprendimento
- i DBMS offrono *top-level interattivi* (console)
  - accesso mediato dalla API, ma meno “filtrato”

MySQL – shell

- “mysql” è un client per l'accesso a MySQL
  - permette uso interattivo (top-level)
    - GNU readline capabilities
  - è script-friendly: si comporta come un filtro *nix tra standard input e standard output
  - http://dev.mysql.com/doc/refman/5.0/en/mysql.html
- esempi d'uso:

```
mysql db_name                                    # defaults
mysql -p -h host -P port -u user db_name         # tcp/ip
mysql -b db_name < script.sql > output.tab # batch
mysql -X ...                                     # xml output
```

http://dev.mysql.com/doc/refman/5.0/en/mysql-command-options.htm

MySQL – shell (cont.)

- configurazione di “mysql”
  - via gruppo “[mysql]” di my.cnf
- esempio (deja-vu?):

```
[client]
user = “username”
password = “new-password”
```

- ricordate: file di configurazione 1-1 con opzioni cmdline

MySQL – shell (cont.)

```
$ # da utente MySQL root
$ mysqladmin create studenti
$ mysql studenti < studenti.sql
$ # in lab: /home/users/corsi/labrs/studenti.sql
$ mysql studenti
mysql> source studenti_data.sql;
mysql> show tables;
mysql> describe Studenti;
mysql> select * from Studenti;
mysql> ...
mysql> CTRL-D
$
```

MySQL – esercizi

- esercizi dalla slide 101
  http://www.bononia.it/~zack/courses/labsimfosset0607/Lezione%201.pdf
- scrivere ed eseguire le query SQL che ritornano:
  1. gli indirizzi di posta elettronica degli studenti che hanno preso 30 e lode nel corso di Analisi
  2. Il nome e cognome degli studenti che hanno sostenuto almeno un esame (senza duplicati)
- domanda: cosa non è enforced dei vincoli espressi nello schema?
  - http://dev.mysql.com/doc/refman/5.0/en/create-table.html
- creare e popolare il db della slide 106

Controllo di accesso

- i DBMS offrono come *servizi di sistema* accesso a larghe e complesse basi di dati
- il principio dei privilegi minimi è implementato con controllo di accesso a 2 livelli:

  1. *canali di accesso* al servizio DBMS
  2. *fine-grained access control*
     - per-db policy
     - meccanismi di gestione dell'utenza (per gli admin del singolo DB)
     - possibili, ma non necessarie relazioni con l'utenza system-wide

Canali di accesso

- canali di accesso
  - *locale* (e.g.: socket sul filesystem)
  - *TCP/IP* (e.g.: socket TCP)
  - ... ma potenzialmente anche RPC, SOAP, ...
- il DCL dei DBMS è solitamente in grado di distinguere tra utenti locali e utenti remoti
  - non sono necessarie utenze separate per i 2 realm
- la gestione dei canali di accesso si riduce a:

1. abilitare/disabilitare i canali di accesso
2. gestire politiche di accesso extra-DBMS
   - e.g.: firewalling, port binding, file system perm, ...

MySQL – canali di accesso

- MySQL offre 2 canali di accesso
  - file system socket
  - porta TCP/IP
- /etc/mysql/my.cnf, gruppo “mysqld” contiene le configurazione dei canali di accesso
- i default:

```
socket = /var/run/mysqld/mysqld.sock


bind-address = 127.0.0.1
port = 3306
# skip-networking
```

Utenza

- i DBMS offrono una gestione dell'utenza separata rispetto all'utenza di sistema
  - rationale:
    - le basi dati sono complesse, spesso hanno necessità di controllo di accesso diverse da quelle di sistema
    - le entità da controllare sono nel dominio del DBMS, non noto al sistema host
    - gli utenti dei vari db spesso non hanno controparti negli utenti di sistema (caso notevole: web apps)
- è comunque spesso possibile “ereditare” utenti di sistema

Controllo di accesso in SQL

- SQL offre un meccanismo di controllo di accessi
  - basato su:
    - *authorization IDs* (nomi utente)
    - privilegi per effettuare operazioni su tabelle
    - due statement nel DCL: GRANT e REVOKE
- al singolo DBMS viene demandata
  - gestione degli authorization ID
  - più (ovviamente) estensioni e restrizioni DBMS-specific
- analizziamo inizialmente il meccanismo nativo di SQL

Privilegi

- i privilegi di SQL
  1. select
  2. insert
  3. delete
  4. update
  5. references
  6. usage
  7. trigger
  8. execute
  9. under

- 1-4 si applicano a tabelle (o viste) con la semantica ovvia
- 5 permette di referenziare una tabella come FOREIGN KEY
- 6: “uso” in altre dichiarazioni
- 7 definizione di trigger
- 8 esecuzione di stored procedure
- 9 sottotipaggio

Controllo dei privilegi

- ogni query SQL richiede un insieme di privilegi per essere portata a termine
  - Esempio:

```
INSERT INTO Studio(name)
   SELECT DISTINCT studioName
   FROM movie
   WHERE studioName NOT IN
      (SELECT name
       FROM Studio)
```

  - privilegi richiesti:
    - insert su Studio, select su Studio (non implicato)
- *tutti* i privilegi sono necessari per completare la query con successo

Authorization id

- ogni query SQL viene eseguita da un agente che impersona un authorization ID: il *current authorization ID*
  - viene solitamente stabilito all'atto della connesione al DB (via API o altri client)
    - e.g. mysql -u user db_name < foo.sql
    - fa si che il current authorization ID per l'esecuzione delle query contenute nel file foo.sql sia “user”
  - può cambiare in corso d'opera con appositi statement SQL (uso raro)

GRANT statement

- lo statement GRANT permette ad un utente (i.e. un agente in esecuzione con un certo authorization ID) di delegare privilegi ad altri utenti
  - tipicamente, esiste un utente super-user che possiede tutti i privilegi: è necessario per il bootstrap del processo di delega
- sintassi:

```
GRANT <privilege list> ON <db element> TO <user list>
    [WITH GRANT OPTION]
```

GRANT statement (cont.)

```
GRANT <privilege list> ON <db element> TO <user list>
  [WITH GRANT OPTION]
```

- db element rappresenta una entità referenziabile del db (e.g. una tabella, un campo, l'intero db)
- privilege list rappresenta la lista dei privilegi che si vuole delegare
- user list rappresenta la lista degli utenti ai quali delegare i privilegi
- “WITH GRANT OPTION”, se presente, permette agli utenti designati di delegare a loro volta i privilegi ottenuti ad altri utenti

- GRANT è eseguibile solo da utenti che possiedono tutti i privilevi da delegare

REVOKE statement

- sintassi:

```
REVOKE <privilege list> ON <db element> FROM <user
  list>
```

- semantica intuitiva e duale a quella di GRANT

MySQL - utenza

- il sistema di privilegi di MySQL si occupa di
  0. autenticare connessioni di utenti locali e remoti
     - outcome: connessione permessa o rifiutata
  1. associare gli utenti che si sono connessi ad un insieme di privilegi
  2. verificare, query per query, che i privilegi necessari ad eseguire la query siano associati all'utente che si è connesso
- (0) è una funzionalità addizionale rispetto a quanto supportato da SQL standard ed è implementato da molti DBMS come controllo addizionale di sicurezza

MySQL – authorization id

- l'identità in MySQL è determinata dalla coppia:
  1. l'host dal quale proviene la connessione
  2. lo username specificato da chi richiede la connessione
     - N.B. in alcuni client (e.g. “mysql”) uno username non specificato ha come default prima il file di configurazione, se esiste, poi l'utente di sistema, ma questo non implica alcuna correlazione tra utenti di sistema e utenti MySQL
- rationale:
  - macchine diverse = realm di protezione diversi

```
SELECT CURRENT_USER();    -- utente corrente
```

MySQL - controlli

- il controllo di accesso effettuato da “mysqld” si divide in 2 fasi:
  1. controllo del permesso di connettersi *(connection verification)*
  2. controllo query per query dei privilegi *(request verification)*
- in entrambi le fasi il server fa affidamento su tabelle del db “mysql” dette *grant table*
  - “user”, “db”, “host” (coarse grained access control)
  - “tables_priv”, “columns_priv”, ... (fine grained)
  - ogni tabella contiene *scope columns* (il contesto della riga) e *privilege columns* (i privilegi garantiti)

MySQL – privilegi

- in aggiunta ai privilegi di SQL MySQL offre privilegi molto fini per controllare
  - chi può creare e rimuovere elementi del db (CREATE/DROP)
  - azioni su viste
  - azioni amministrative sul db (e.g. shutdown)
  - azioni su indici
- http://dev.mysql.com/doc/refman/5.0/en/privileges-provided.html

MySQL – connection verification

- una connessione da user@host (con password pwd) è accettata se
  - nella tabella “user” esiste una riga t.c. Host=”host”, User=”user” (, Password=”pwd”)
- notazioni
  - tutti i campi supportano le usuali wildcard “%” (sequenza arbitraria di caratteri) e “_” (un carattere arbitrario)
  - host specificati come IP supportano netmask
    - e.g. “192.168.0.0/255.255.255.0”

MySQL – request verification

- tabelle
  - la tabella “user” stabilisce i privilegi DBMS wide
    - e.g. se in “user” viene garantito il privilegio “DELETE” l'utente può cancellare righe dalle tabelle di tutti i DB
  - “db” e “host” garantiscono privilegi DB-specific
  - “tables_priv”, “columns_priv”, ... DB-element specific
- GRANT e REVOKE possono essere usati per modificare tutti i privilegi d'accessi visti in MySQL
- ma è consigliato l'uso di statement specifici per l'utenza

MySQL – gestione dell'utenza

- creazione di utenti (senza privilegi)

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
```

  - poi GRANT/REVOKE ....
- rimozione di utenti

```
DROP USER user
```

- password

```
SET PASSWORD [FOR user] = PASSWORD('some password')
```

- ispezione dei privilegi

```
SHOW GRANTS [FOR user]
```

- http://dev.mysql.com/doc/refman/5.0/en/account-management-sql.html

MySQL – esercizi

- nel db “studenti” precedentemente creato impostare i permessi come segue
  1. creare un account “admin” con tutti i privilegi possibili sul db (tranne la grant option)
  2. creare un account “segretario” (con password) che possa ispezionare e modificare le tabelle “Studenti” e “Corsi” e che inoltre possa ispezionare la tabella “Esami”
  3. creare un account “docente” (con password) che possa inserire nuove righe nella tabella “Esami”
  - l'account “admin” deve potere accedere solo da localhost, gli altri solo dalla rete 192.168.0.0/24

MySQL – storage engine

- nello schema del nostro esempio molti constraint sono stati ignorati
  - check, foreign key, ...
  - perché?
    - perché lo *storage engine* di default per tabelle non li supporta
- altre feature di MySQL sono storage-dependent
  - transazioni, fulltext index, ...
- molte di queste feature sono supportate dallo storage InnoDB

MySQL – scelta dello storage

- lo storage engine tabilisce la rappresentazione fisica dei dati in “memoria”
  - vari trade-off: efficienza, compattezza, feature
- lo storage viene scelto all'atto di creazione di una tabella

```
CREATE TABLE tbl_name (create_definition,...)
  [table_option ...]

table_option:

    ENGINE = engine_name

    ...
```

- http://dev.mysql.com/doc/refman/5.0/en/create-table.html
- lo storage di default è MyISAM

MySQL – storage engines

- alcuni storage engine di MySQL
  - bdb (BerkeleyDB): transaction-safe, page locking, deprecato
  - csv (comma separated value) !!!
  - InnoDB: transaction-safe, row locking, foreign keys
  - memory: heap representation, memory only
  - MyISAM: MySQL default, portabile
  - NDB: clustered, fault-tolerant
- http://dev.mysql.com/doc/refman/5.0/en/storage-engines.html

MySQL - MyISAM

- `CREATE TABLE t (i INT) ENGINE = MYISAM;`
- on disk
  - ogni tabella è rappresentata da 3 file:
    - .frm (table format), .MYD (data), .MYI (index)
    - `ls /var/lib/mysql/`*`dbname/`*
- tradeoff: performance a discapito di feature
  - non sono supportate foreign key e check constraint
  - migliori performance in lettura di altri storage
  - sono supportati indici fulltext
- http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html

MySQL - InnoDB

- `CREATE TABLE t (i INT) ENGINE = INNODB;`
- feature
  - row-level locking, ACID-transactions, foreign keys
  - nessun limite sulle dimensioni delle tabelle
  - alte prestazioni su dati voluminosi (ordine dei TB)
- on disk
  - tablespace privato (per tabella o per db): molti file o anche partizioni raw
- configurazione
  - deve essere abilitato in my.cnf, è il default
- http://dev.mysql.com/doc/refman/5.0/en/innodb.html

MySQL – esercizi

- ricreate il database “studenti” degli esempi precedenti utilizzando InnoDB come storage engine
- quali constraint sono ora enforced da MySQL?
- cosa manca?

DBMS per sysadmin

introduzione all'amministrazione di PostgreSQL

Postgres - pacchetti

- pacchetto sorgente “postgresql-8.2”, binari:
  - postgresql-8.2
    - server: demone standalone
  - postgresql-client-8.2
    - client console-like e tool amministrativi
  - libpq5*, libpq*-dev
    - librerie (shared e non) per l'accesso via API nativa
  - postgresql-common
    - (source package: postgresql-common)
    - shared stuff, e.g. file di configurazione /etc/mysql/*
    - management di più versioni di Postgres

Postgres - pacchetti (cont.)

- binari:
  - libecpg*, libpgtypes*
    - librerie per *Embedded PostgreSQL for C (EPCG)*
      - sviluppo in C con query SQL (a Postgres) verbatim
      - non è una versione di Postgres embedded!
  - postgresql-server-dev-8.2
    - librerie (dev part) per implementare estensioni SSI di Postgres (ad esempio in C)
    - non per client application
  - postgresql-contrib-8.2
    - estensioni di terze parti
      - e.g.: gist, crypto support, fulltext search, xml storage, tipi per ISBN, ...

Postgres - pacchetti (cont.)

- binari:
  - postgresql-pl{tcl,perl,python}*
    - supporto per l'implementazione di stored procedure in tcl/perl/python
  - postgresql-doc-8.2
    - documentazione in formato HTML

      `/usr/share/doc/postgresql-doc-8.2/`
- metapacchetti
  - postgresql, postgresql-{client,doc,contrib}

Postgres – installation HOWTO

- facile:

  ```
  aptitude install postgresql
  ```

  - oppure

  ```
  aptitude install postgresql-8.2
  ```

- creati automaticamente durante il postinst
  - un cluster “main”
  - un superuser “postgres”
    - cui corrisponde un utente di sistema “postgres” che può accedere come superuser a Postgres

Postgres – servizio e configurazione system-wide

- il demone è integrato come servizio di sistema e implementa la usuale API Debian
  - servizio “postgresql-X.Y”
    - e.g. postgresql-8.2
  - usual stuff

    ```
    invoke-rc.d postgresql-8.2 start/stop/...
    ```

    - soggetto alla configurazione dei runlevel
- file di configurazione system-wide
  - /etc/postgresql/*X.Y*/*cluster*/*.conf
    - e.g.: /etc/postgresql/8.2/main/postgresql.conf

Postgres – gestione dei db

- l'amministrazione di postgres solitamente avviene usando l'utente (di sistema) postgres
  - è possibile delegare l'uso di questo utente ad utenti diversi da root con tecniche usuali
    - sudo, password protected account, ...
  - è possibile che altri utenti di sistemi diventino superuser postgres
- nuovi db si possono creare con il comando createdb (nel PATH dell'utente postgres)

```
createdb [mydb]     # default: username (di sistema)
dropdb mydb         # duale: rimuove un db
```

Postgres - shell

- “psql” è un client per l'accesso a Postgres
  - permette uso interattivo (top-level)
- uso:

  `psql [--password] [`*`mydb`* `[`*`username`*`]]`
- prompt shell like:

```
mysql>          -- prompt for “normal” users
mysql#          -- prompt for super users
```

- permette query interattive SQL e offer comandi non-SQL (*psql commands*) per altri task
  - e.g.: ispezione dei db disponibili: `\l`
- help in linea: (`\h` per SQL), (`\?` per psql)

Postgres – utenza

- l'utenza gestita da Postgres è indipendente dagli utenti di sistema
- è basata su *roles* (gli “utenti” di Postgres)
  - creazione

```
CREATE ROLE name;  -- SQL
createuser name    -- shell
```

  - rimozione

```
DROP ROLE name;    -- SQL
dropuser name      -- shell
```

  - ispezione

```
SELECT rolname FROM pg_roles;     -- SQL
\du                               -- psql
```

Postgres – utenza (cont.)

- per motivi di bootstrap il ruolo “postgres” è predefinito
  - più in generale: nome di chi ha creato il cluster
- ogni connessione a Postgres è effetuata in un ruolo ben preciso
- il processo di *autenticazione* stabilisce l'identità di un utente
  - outcome: permesso o divieto di connettersi

Postgres - autenticazione

- è controllata dal file di conf “pg_hba.conf”
  - file testuale, insieme di record, uno per riga
    - campi (ordinati):
      1. tipo di connessione
      2. IP address range
      3. db name
      4. user name
      5. metodo di autenticazione
- semantica:
  - ad ogni connessione: viene selezionato il primo record i cui campi da 1 a 4 corrispondono al tentativo di connessione
  - l'utente viene autenticato con il metodo del campo 5

Postgres – auth (cont.)

- possibili formati dei record di pg_hba.conf

```
local      database  user  auth-method  [auth-option]
host       database  user  CIDR-address  auth-method  [auth-option]
hostssl    database  user  CIDR-address  auth-method  [auth-option]
hostnossl  database  user  CIDR-address  auth-method  [auth-option]
```

  - in alternativa a CIDR sono possibili 2 campi IP address / network mask
- tipi di connessione
  - local (locale via socket)
  - host{ssl,nossl} (TCP/IP con/senza SSL)
  - host (TCP/IP whatever)

Postgres – auth (cont.)

- record di pg_hba.conf

```
host      database  user  CIDR-address  auth-method  [auth-option]
```

- database identifica un db
  - valori notevoli: “all”, “sameuser”
- user identifica un utente
  - valori notevoli: “all”, “@file”, “+group”
- CIDR-address identifica un (range di) IP
  - ip address (x.y.z.w)
  - address range (x.y.z.w/m)

Postgres – auth (cont.)

- record di pg_hba.conf

```
host      database  user  CIDR-address  auth-method  [auth-option]
```

- auth-method, metodo di autenticazione:
  - trust/reject
    - permetti/rifiuta l'accesso incondizionatamente
  - md5/crypt/password
    - password base, con vari tipi di password che vengono comunicati lungo la connessione (occhio allo sniffing)
  - ident, pam, krb5, ldap

Postgres - password

- le pasword possono essere associate agli utenti all'atto della creazione via createuser
  - di default ad un utente non viene associata nessuna password
    - i suoi tentativi di connessione che richiedono un metodo di autenticazione password based falliranno!
- è possibile cambiare la password di un utente utilizzando ALTER ROLE (SQL)
  - e.g.: `ALTER ROLE davide WITH PASSWORD 'hu8jmn3';`

Postgres - privilegi

- GRANT e REVOKE come d'abitudine
- inoltre ad ogni ruolo sono associati *attributi*
  - creati alla creazione del ruolo, modificabili con ALTER ROLE
  - esempi notevoli:
    - LOGIN (permette ad un utente di connettersi, c'è di default per utenti creati con “CREATE USER” o a cmdline)
    - SUPERUSER (non sottosta a controllo di permessi)
    - CREATEDB (permette di creare nuovi db)
    - CREATEROLE (permette di creare nuovi ruoli)

Postgres - gruppi

- i ruoli di Postgres possono simulare gruppi di utenti
  - e.g.

```
CREATE ROLE joe LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;   -- no LOGIN
CREATE ROLE wheel NOINHERIT;   -- no LOGIN
GRANT admin TO joe;      -- joe in “group” admin
GRANT wheel TO admin;    -- admin in “group” wheel
```

- “INHERIT” permette di eriditare i permessi dei ruoli di un utente
  - cambio di ruolo (se posseduto): `SET ROLE` *`role_name`*`;`

Postgres - esercizi

- redo...
  - creazione del database “studenti” visto in precedenza (adattando lo schema studenti.sql se necessario) e popolazione dello stesso
  - quali dei vincoli espressi nello schema non sono enforced?
  - creazione degli utenti “admin” e “segretario” come in precedenza
- ...and improve!
  - creazioned di due utenti “pciaccia” e “zack” con ruolo “docente”
    - il ruolo ha i permessi visti in precedenza per “docente”