Breve guida a MySQL

(versione per Linux)

Indice

1 Introduzione **3**

2 Attivazione del DBMS **5**

3 Creare una base di dati **7**

4 Creare ed eliminare le tabelle **9**

4.1 Creare tabelle 9

4.2 Eliminare tabelle 10

5 Inserire tuple **11**

6 Interrogare tabelle **13**

7 Utilizzare il catalogo **14**

8 Eseguire comandi memorizzati su file **17**

9 Supporto transazionale in MySQL **18**

Capitolo 1

Introduzione

Per le esercitazioni del corso di basi di dati, si fa uso di un DBMS open-source, MySQL 5 che gira sotto WindowsNT/2000/XP, sotto Linux e sotto Mac OS X. Tale DBMS è disponibile gratuitamente su http://dev.mysql.com/downloads/. Per maggiori informazioni su questo DBMS si faccia riferimento alla Documentazione MySQL. In particolare il manuale di MySQL include un buon tutorial che introduce all'uso del sistema, con maggiori dettagli rispetto al presente documento.

Nota: MySQL 5 presenta sostanziali cambiamenti rispetto alle versioni precedenti. In particolare, introduce un adeguato supporto di sotto-query negli *statement select* e la possibilità di definire viste. Poichè nel corso facciamo uso di entrambi, le versioni precedenti alle 5 di MySQL non sono da considerarsi adatte.

Nota2: Questa guida fa riferimento all'uso di MySQL su sistemi Linux. Installazione del DBMS MySQL 5 (in ambiente Linux)

L'installazione che andremo ad illustrare, fa riferimento ai pacchetti della lista Linux (non RPM package), sono tuttavia presenti precompilati per l'installazione immediata per varie architetture quali *SuSe*, *Ubuntu* ed altre.

Per prima cosa, è importante verificare di soddisfare i pre-requisiti richiesti:

- glibc-2.2 o versione superiore
- Compilatore C/C++
- GNU gunzip
- GNU tar

Una volta che i pre-requisiti sono stati soddisfatti, possiamo procedere con l'installazione di MySQL 5.0.

Per prima cosa effettuare il download nella sezione Linux (non RPM package) del file corrispondente alla nostra architettura.

Una volta che il file è stato scaricato aprire un terminale e *loggarsi* come **root**:

```
user@localhost:>su
password: *******
root@localhost#:
Adesso seguire le seguenti istruzioni:
root@localhost#: groupadd mysql
root@localhost#: useradd -g mysql mysql
root@localhost#: cd /usr/local
root@localhost#: gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf
-
root@localhost#: ln -s full-path-to-mysql-VERSION-OS mysql
root@localhost#: cd mysql
root@localhost#: scripts/mysql_install_db --user=mysql
root@localhost#: chown -R root .
root@localhost#: chown -R mysql data
root@localhost#: chgrp -R mysql .
root@localhost#: bin/mysqld_safe --user=mysql &
```

Per avere maggiori dettagli sull'installazione, potete visitare la guida ufficiale su MySQL 5.0 all'indirizzo:

http://dev.mysql.com/doc/refman/5.0/en/installing-binary.html
e per ulteriori approfondimenti sulla compilazione manuale per l'installazione la seguente guida:

http://dev.mysql.com/doc/refman/5.0/en/quick-install.html.
Questa installazione base di MySQL permette di utilizzare il software direttamente da shell, esistono tuttavia vari tool grafici che possono agevolare l'uso, sempre sul sito ufficiale sono disponibili i download al link:
http://dev.mysql.com/downloads/gui-tools/5.0.html
non che le relative guide (anche in italiano): http://dev.mysql.com/doc/

Capitolo 2

Attivazione del DBMS

MySQL richiede sempre un avvio del server per poter accedere al DBMS, è quindi indispensabile avviare la connessione ad ogni utilizzo.
Per fare ciò innanzitutto *logghiamoci* come root e accediamo da terminale nella directory `/usr/local/mysql/bin/`
Adesso che ci troviamo nella directory possiamo avviare il server con il seguente comando:

```
root@localhost#: ./mysqld --user root
```

o configurare l'avvio automatico come illustrato sul manuale di riferimento di MySQL 5 .

Da questo momento in poi potremmo accedere al servizio MySQL e alle nostre basi di dati.

Al termine del lavoro, è buona norma chiudere il server accedendo come root sempre nella directory `/usr/local/mysql/bin/` da terminale e digitare:

```
root@localhost#: ./mysqladmin --user root shutdown Accesso al DBMS
```

A questo punto si possono digitare comandi SQL, che andranno ad operare sul DBMS a cui si è connessi. Per avere accesso all'interprete di comandi SQL che ci viene messo a disposizione dall'applicazione, dobbiamo recarci nella seguente directory:

```
root@localhost#: .cd /usr/local/mysql/bin/
```

una volta entrati possiamo usare il comando mysql nel seguente modo:

```
root@localhost#: ./mysql -u root
```

comparirà il messaggio:

```
Welcome to the MySQL monitor. Commands end with ; or \{\}g. Your
MySQL connection id is 2 to server version: 5.0.24a-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

con il promt:

```
mysql>
```

da cui inserire i nostri comandi SQL.
I parametri `-u root` stanno a simboleggiare `-user root`, ovvero si indica che, l'utente che vuole accedere, è root.
Un esempio di comando che possiamo usare, è chiedere quali basi di dati sono gestite dal DBMS e quindi sono accessibili dall'utente root.
Per fare ciò digitiamo:

```
mysql> show databases;
```

Per chiudere l'interprete dei comandi digitiamo:

```
mysql> quit
Bye
```

Nota: MySQL 5.0, prevede la presenza di un utente con particolari privilegi di amministrazione su tutti le basi di dati, tra i quali quelli di sistema, che assolutamente non vanno toccati. L'utente root subito dopo l'installazione, è senza password ed è cosa buona assegnarne una per evitare spiacevoli intrusioni nel nostro lavoro.
Per fare questo eseguire l'accesso come l'utente root:

```
root@localhost#: ./mysql -u root
```

settare adesso la nuova password:

```
mysql> SET PASSWORD FOR root@localhost =
PASSWORD('tua_nuova_password');
```

una volta settata la password, basta accedere come root nel seguente modo:

```
root@localhost#: ./mysql -u root -p

Password: *******
```

Capitolo 3

Creare una base di dati

Vediamo ora come creare una nuova 'base di dati'. La base di dati che vogliamo creare si chiama menagerie, ed è presa dal tutorial presente nel manuale di riferimento di MySQL 5.

Per poter creare una base di dati dobbiamo avere diritti d'accesso sufficienti. Per fare ciò ancor prima di creare la base di dati vera e propria accediamo al server come root, come già mostrato, se tutto sarà fatto bene avremo il seguente messaggio:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 5.0.24a-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

Come utente root diamo il comando SQL, creiamo un nuovo utente 'mysql' (ancora senza password)

```
mysql> create user mysql;
Query OK, 0 rows affected (0.00 sec)
```

e diamo a questo nuovo utente tutti i diritti sulla base di dati menagerie che andremo a creare tra poco. Qualora per qualche ragione volessimo eliminare l'utente appena creato lo possiamo fare con un comando SQL:

```
mysql> drop user mysql;
```

Una volta creato l'utente mysql gli diamo tutti i diritti di accesso relativamente alla base di dati menagerie.

```
mysql> GRANT ALL ON menagerie.* TO mysql;
Query OK, 0 rows affected (0.02 sec)
```

A questo punto usciamo come root

```
mysql> quit
Bye
```

e ci ricolleghiamo come mysql.

```
root@localhost#: ./mysql -u mysql

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 5.0.24a-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

Ora finalmente creiamo la base di dati menagerie

```
mysql> CREATE DATABASE menagerie;
Query OK, 1 row affected (0.03sec)
```

Per controllare che la base di dati sia stata effetivamente creata digitiamo il comando SQL:

```
mysql> SHOW databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| menagerie          |
| test               |
+--------------------+
3 rows in set(0.01sec)
```

Per utilizzare la base di dati appena creata diamo il comando:

```
mysql> use menagerie;
Database changed
```

che pone nel contesto della base di dati menagerie.

Va ricordato che ciò che vieni indicato come basi di dati negli statement SQL è sostanzialmente un *name space*. Il comando `use` cambia appunto il name space corrente. Oggetti di altri *name-space* possono comunque essere risolti facendo precedere il loro nome dal nome del loro name-space e dal punto.

Nota: Tenere presente che il nome utente mysql è puramente fittizio, ognuno può assegnare l'user name che più preferisce.

Capitolo 4

Creare ed eliminare le tabelle

4.1 Creare tabelle

In MySQL possiamo eseguire qualsiasi comando SQL. Per creare una tabella si usa il comando SQL standard create table avente la forma seguente:

```
create table <nome-tabella> (
<lista di attributi e tipi ad essi associati>
);
```

Nello scrivere un comando si può andare a capo. Se il comando occupa più linee, ad ogni *return* viene dato un prompt speciale con il numero di linea corrente fino a che non si digita il punto e virgola.
Per esempio per creare la tabella `pet` seguente:

```
CREATE TABLE pet ( /* "-- " indica un commento che arriva fino alla
fine della riga corrente */
 name     VARCHAR(20),  -- stringa di al più 20 caratteri
 owner    VARCHAR(20),  --  ''
 species  VARCHAR(20),  --  ''
 sex      CHAR(1),      -- stinga di un carattere
 birth    DATE,         -- data
 death    DATE          --  ''
)
```

diamo il comando:

```
mysql> CREATE TABLE pet (
 ->       name VARCHAR(20),
 ->       owner VARCHAR(20),
 ->       species VARCHAR(20),
 ->       sex CHAR(1),
```

```
  ->       birth DATE,
  ->       death DATE
  ->    );
Query OK, 0 rows affected (0.35 sec)
```

Si noti il 'punto e virgola' finale, che abbiamo gia usato altre volte. Il 'punto e virgola' serve ad indicare all'interprete dei comandi che il comando corrente è terminato.

4.2 Eliminare tabelle

Per eliminare una tabella dalla base di dati, si esegue:

```
drop table <nome-tabella> ;
```

In generale dopo aver creato tabelle di prova si suggerisce di eliminarle con `drop table` per mantenere pulita la base di dati.
Ad esempio eseguire:

```
drop table pet;
```

alla fine della sessione di prova.

Capitolo 5

Inserire tuple

Dopo aver creato una tabella possiamo inserirvi delle tuple utilizzando il comando `insert`. La maniera più semplice è inserire direttamente i valori:

```
insert into <nome-tabella>
values( <lista ordinata dei valori da inserire negli attributi>);
```

Per esempio, scrivere:

```
mysql > INSERT INTO pet VALUES
('Fluffy','Harold','cat','F','1993-02-04',NULL);

Query OK, 1 row affected (0.34 sec)
```

Similmente possiamo inserire altre tuple popolando così la nostra tabella:

```
INSERT INTO pet VALUES ('Claws','Gwen','cat','m','1994-03-17',NULL);

INSERT INTO pet VALUES
('Buffy','Harold','dog','f','1989-05-13',NULL); INSERT INTO pet
VALUES ('Fang','Benny','dog','m','1990-08-27',NULL); INSERT INTO pet
VALUES ('Bowser','Diane','dog','m','1979-08-31','1995-07-29');

INSERT INTO pet VALUES
('Chirpy','Gwen','cat','f','1998-09-11',NULL); INSERT INTO pet
VALUES ('Whistler','Gwen','bird',NULL,'1997-12-09',NULL); INSERT
INTO pet VALUES ('Slim','Benny','bird','m','1996-04-29',NULL);

INSERT INTO pet VALUES
('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

Si ricorda che SQL è case-insensitive, cioè **non** distingue tra maiuscolo e minuscolo, tranne ovviamente nelle stringhe, quindi il valore 'F' dell' attributo sex nella prima riga e diverso da 'f' nelle altre righe.

Per cancellare ed aggiornare tuple si utilizzano i comandi standard SQL update e delete. Per esempio:

```
mysql> UPDATE pet SET sex = 'f' WHERE sex = 'F';

Query OK, 1 row affected (0.34 sec)
Rows matched: 4 Changed: 1 Warnings: 0
```

modifica 'F' in 'f' nell'attributo sex, mentre

```
mysql> DELETE FROM pet WHERE year(birth) < '1990';

Query OK, 2 rows affected (0.00 sec)
```

elimina le tuple in cui anno di nascita, ottenuto applicando la funzione predefinita year all'attributo birth, è $<$ di 1990.

Capitolo 6

Interrogare tabelle

Possiamo vedere quali sono i valori memorizzati in una tabella attraverso una semplice query:

```
select *
from <nome-tabella> ;
```

Per esempio, dopo aver creato la tabella pet ed aver inserito le tuple riportate sopra possiamo farci restituire il contenuto della tabella con lo statement

```
select * from pet;
```

restituisce il seguente risultato

```
mysql> select * from pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | F    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | cat     | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | bird    | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+

9 rows in set (0.00 sec)
```

Capitolo 7

Utilizzare il catalogo

Il DBMS mantiene tutte le informazioni sulla base di dati (*metadati*) in delle tabelle di sistema che formano il cosiddetto catalogo. Le tabelle del catalogo sono interrogabili attraverso SQL esattamente come le tabelle create dagli utenti. MySQL mantiene il catalogo su una base di dati chiamata INFORMATION_SCHEMA. In particolare in questa base di dati è presente la tabella TABLES che contiene le informazioni su tutte le tabelle della base di dati.
Per utilizzare la base di dati INFORMATION_SCHEMA eseguiamo il comando:

```
mysql> use information_schema Reading table information for
       completion of table and column names You can turn off this feature
       to get a quicker startup with -A

Database changed
```

A questo punto possiamo eseguire delle query su INFORMATION_SCHEMA per esempio:

```
mysql> select table_name, table_schema from tables;
+---------------------------------------+--------------------+
|table_name                             | table_schema       |
+---------------------------------------+--------------------+
|CHARACTER_SETS                         | information_schema |
|COLLATIONS                             | information_schema |
|COLLATION_CHARACTER_SET_APPLICABILITY  | information_schema |
|COLUMNS                                | information_schema |
|COLUMN_PRIVILEGES                      | information_schema |
|KEY_COLUMN_USAGE                       | information_schema |
|ROUTINES                               | information_schema |
|SCHEMATA                               | information_schema |
```

```
|SCHEMA_PRIVILEGES                       | information_schema |
|STATISTICS                              | information_schema |
|TABLES                                  | information_schema |
|TABLE_CONSTRAINTS                       | information_schema |
|TABLE_PRIVILEGES                        | information_schema |
|TRIGGERS                                | information_schema |
|USER_PRIVILEGES                         | information_schema |
|VIEWS                                   | information_schema |
|pet                                     | menagerie          |
+----------------------------------------+--------------------+

17 rows in set (0.01 sec)
```

Dato il nome della tabella è possibile ottenere i suoi attributi (nome e tipo) con il comando:

```
describe <nome-tabella>
```

Per esempio, per sapere gli attributi memorizzati dal sistema in TABLES si può usare:

```
mysql> describe tables;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
|TABLE_CATALOG    | varchar(512) | YES  |     | NULL    |       |
|TABLE_SCHEMA     | varchar(64)  | NO   |     |         |       |
|TABLE_NAME       | varchar(64)  | NO   |     |         |       |
|TABLE_TYPE       | varchar(64)  | NO   |     |         |       |
|ENGINE           | varchar(64)  | YES  |     | NULL    |       |
|VERSION          | bigint(21)   | YES  |     | NULL    |       |
|ROW_FORMAT       | varchar(10)  | YES  |     | NULL    |       |
|TABLE_ROWS       | bigint(21)   | YES  |     | NULL    |       |
|AVG_ROW_LENGTH   | bigint(21)   | YES  |     | NULL    |       |
|DATA_LENGTH      | bigint(21)   | YES  |     | NULL    |       |
|MAX_DATA_LENGTH  | bigint(21)   | YES  |     | NULL    |       |
|INDEX_LENGTH     | bigint(21)   | YES  |     | NULL    |       |
|DATA_FREE        | bigint(21)   | YES  |     | NULL    |       |
|AUTO_INCREMENT   | bigint(21)   | YES  |     | NULL    |       |
|CREATE_TIME      | datetime     | YES  |     | NULL    |       |
|UPDATE_TIME      | datetime     | YES  |     | NULL    |       |
|CHECK_TIME       | datetime     | YES  |     | NULL    |       |
|TABLE_COLLATION  | varchar(64)  | YES  |     | NULL    |       |
|CHECKSUM         | bigint(21)   | YES  |     | NULL    |       |
|CREATE_OPTIONS   | varchar(255) | YES  |     | NULL    |       |
```

```
|TABLE_COMMENT    | varchar(80)  | NO   |     |         |       |
+-----------------+--------------+------+-----+---------+-------+

21 rows in set (0.00 sec)
```

Si noti che i valori contenuti in tutte queste tabelle (nomi di tabelle, colonne, viste, vincoli, utenti, schemi, etc.) sono stringhe scritte completamente in MAIUSCOLO. Per cui se si vuole far riferimento a questi nomi in query SQL bisogna scrivere le stringhe completamente in maiuscolo.
Si noti che anche se stiamo usando la base di dati INFORMATION_SCHEMA possiamo fare riferimento alle tabelle di una diversa base di dati usando il nome completo della stessa. Per esempio:

```
mysql> describe menagerie.pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+

6 rows in set (0.01 sec)
```

Capitolo 8

Eseguire comandi memorizzati su file

Anziché eseguire comandi SQL digitandoli su terminale è spesso più conveniente scriverli in un file di testo e poi richiamarli dal interprete dei comandi MySQL.
Supponiamo di aver scritto alcuni comandi SQL in un file `foo.sql` nella directory corrente.
Possiamo eseguire il file da MySQL con il comando:

```
mysql> source foo.sql
```

oppure:

```
mysql \. foo.sql
```

E' ovviamente possibile anche specificare il path completo del file.
Ad esempio possiamo scrivere:

```
@d:\mia_cartella\foo.sql.
```

Nota: per realizzare una base di dati è utile preparare i seguenti file:

- Un file per creare tabelle e viste;
- Un file per rimuovere tutte le tabelle e le viste;
- Un file per popolare le tabelle, cioè per inserire tuple nelle tabelle.

Capitolo 9

Supporto transazionale in MySQL

MySQL per default ha supporto transazionale limitato. Per attivarle il pieno supporto transazionale si deve far partire il server MySQL con una specifica opzione:

```
root@localhost#: ./mysqld --user=root
--default-storage-engine=InnoDB Starting mysqld daemon with
databases from /usr/local/mysql/data
```

L'opzione `-default-storage-engine=InnoDB` specifica che si vuole utilizzare il sottosistema transazionale di MySQL che si chiama *InnoDB*. Per attivare questa opzione si devono avere i privilegi di root sul DBMS (ecco perché usiamo `-user=root`).
Un file per popolare le tabelle, cioè per inserire tuple nelle tabelle.
Si noti che in MySQL si possono associare specifici sottosistemi di gestione dei dati (*storage engine*) anche alle tabelle singolarmente e tra vari *storage engine* messi a disposizione da MySQL troviamo:

- InnoDB, che come detto offre pieno supporto transazionale;
- MyISAM, che è lo storage engine di default di MySQL;
- Memory, che lavora interamente in memoria RAM.

Nel seguito noi assumeremo di aver attivato InnoDB.

```
root@localhost#: ./mysql -u mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.24a-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> use menagerie Reading table information for completion of
table and column names You can turn off this feature to get a
quicker startup with -A

Database changed
```

Ora definiamo una tabella di prova:

```
mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A));
Query OK, 0 rows affected (0.01 sec)
```

e disabilitiamo l'*autocommit* che rende permanenti i cambiamenti sulla base di dati in modo automatico ponendo la variabile di sistema `AUTOCOMMIT` a 0:

```
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
```

Ora inseriamo una tupla:

```
mysql> INSERT INTO CUSTOMER VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
```

e rendiamo la modifica permanente in modo esplicito tramite un COMMIT:

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Inseriamo ora un'altra tupla:

```
mysql> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM CUSTOMER;

+------+--------+
| A    | B      |
+------+--------+
|   10 | Heikki |
|   15 | John   |
+------+--------+
2 rows in set (0.00 sec)
```

Ma prima di rendere permanente la modifica, diamo l'istruzione di *ROLLBACK* che annulla tutti i cambiamenti fatti fino all'ultimo *COMMIT*:

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM CUSTOMER;

+------+--------+
| A    | B      |
+------+--------+
| 10   | Heikki |
+------+--------+
1 row in set(0.00 sec)
```

Infine rimettiamo la variabile di sistema *AUTOCOMMIT* al suo valore di default: 1:

```
mysql> SET AUTOCOMMIT = 1;
Query OK, 0 rows affected (0.00 sec)
```

Per verificare il valore corrente della variabile *AUTOCOMMIT* possiamo dare il comando:

```
mysql> select @@AUTOCOMMIT;

+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
1 row in set(0.00 sec)
```

Abbiamo detto che lo store engine InnoDB offre pieno supporto transazionale come previsto in SQL standard. Invece lo storage engine MyISAM (lo storage engine di default di MySQL) non offre la possibilità di disabilitare l'*autocommit* rendendo impossibile scrivere transazioni formate da più istruzioni SQL. Infine si noti che in InnoDB quando si esegue un rollback, non vengono ritratti gli statement `create table`, `drop table`, ecc. a meno che non si riferiscano ad una tabella temporanea, in questo MySQL si distinge dallo standard di SQL (come per altro diversi altri DBMS, incluso diversi prodotti dalla Oracle).
Si veda la Documentazione MySQL per maggiori approfondimenti.

Questo documento è stato scritto dallo studente Roberto Zenobio per il Corso di Basi di Dati, Laurea in Ingegneria Informatica, Università di Roma La Sapienza, anno accademico 2006/07. Il documento è la versione in ambiente Linux di un analogo documento scritto da Giuseppe De Giacomo per lo stesso corso. Nel redigere lo stesso si è fatto riferimento a documenti analoghi scritti da Domenico Lembo per edizioni precedenti del corso di Basi di Dati e di Progetto di Basi di Dati, Laurea in Ingegneria Informatica, Università di Roma La Sapienza, ed al Tutorial presente nel Manuale di Riferimento di MySQL 5.