



Basi Dati

Il Database Management System: MySQL

a.a. 2021/2022

Prof.ssa G. Tortora

Prof. M. Risi

MySQL ()

- È un DBMS (Database Management System) per database relazionali, sviluppato dalla MySQL AB



(<http://www.mysql.com>), basato sul linguaggio di definizione ed interrogazione dei dati SQL (Structured Query Language).

- Queste le sue caratteristiche:
 - È un software **libero** (Open Source). Il codice sorgente è disponibile al pubblico, tutti possono usarlo senza pagare diritti d'autore di nessun tipo, possono modificarlo, migliorarlo e adattarlo alle più svariate esigenze.
 - È ottimizzato per la velocità di esecuzione. Alcune caratteristiche più complesse di SQL sono state sacrificate per garantire una maggiore velocità.
 - Questo lo rende particolarmente adatto ad applicazioni via web.

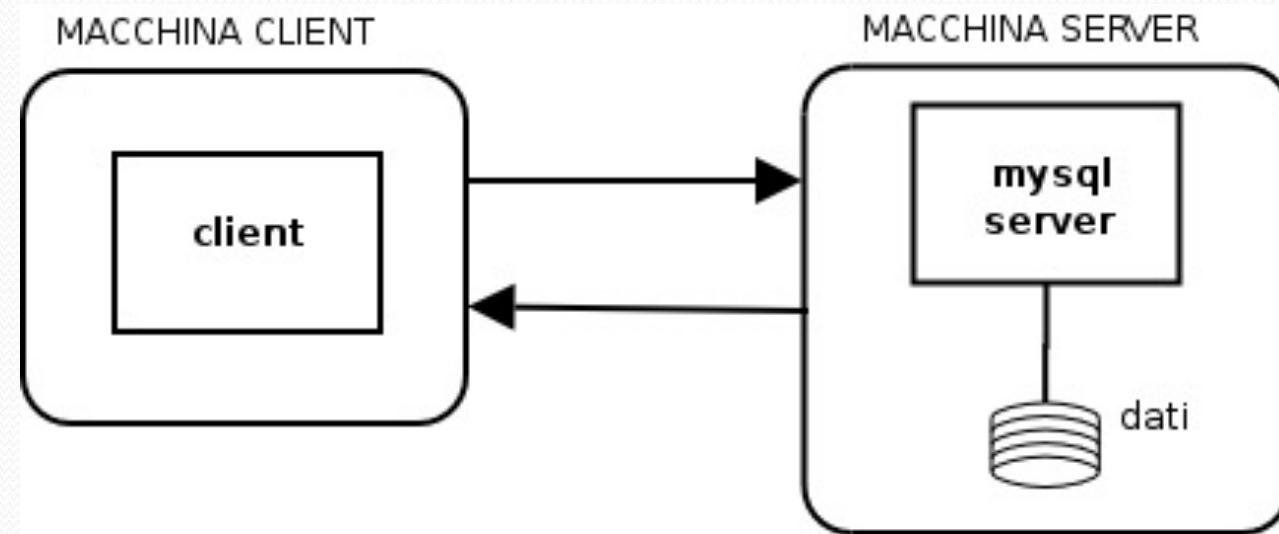
MySQL (2)

- La MySQL AB offre, a pagamento, supporto commerciale, training, certificazioni che lo rendono adatto anche all'utilizzo aziendale.
- È disponibile per un'ampia varietà di hardware e sistemi operativi: *Windows, Linux, OS/2, MacOs, Solaris*.
- È affiancato da un'ampia gamma di software che lo rendono interfacciabile con i più comuni linguaggi e ambienti di programmazione.
- Connettività a livello applicativo tramite socket *TCP/IP* e supporto *ODBC* per la connessione con client Windows, Linux, Mac OS X, and Unix.
- MySQL Connector/J è il driver ufficiale per MySQL per le piattaforma Java.

(<http://dev.mysql.com/downloads/connector/j/> contenuto nel pacchetto di installazione di MySQL)

Cosa è MySQL?

- MySQL è un sistema **client-server**. Il vero e proprio DBMS è il server, che gestisce i database ed esegue i comandi SQL.
- Per poter collegarsi al **server** e impartire ad esso dei comandi, è necessario un programma **client**.
- Il più semplice di tutti è il **monitor** di MySQL.
(<http://dev.mysql.com/downloads/workbench> contenuto nel pacchetto di istallazione di MySQL)



Installazione MySQL 5.7.2 (ultima 8.0.22)

- Passi da eseguire in ambiente Windows:
 - Dal sito ufficiale dei produttori di MySQL scaricare il pacchetto di installazione - *Community Server*: (<http://dev.mysql.com/downloads/mysql>).



- Lanciare il **setup** e seguire la procedura di installazione.

Installazione MySQL - Passi

- Verrà quindi avviato l'**Installation Wizard** di MySQL, che ci guiderà attraverso i seguenti passi:
 1. Accettare il License Agreement per acquisire la licenza d'uso.
 2. Per prima cosa ci viene chiesto che tipo di installazione vogliamo eseguire: scegliamo “**Developer Default**”; è possibile sceglierne altre tra cui “Custom”.
 3. Viene eseguito un check dei requisiti di sistema, ed eventualmente vengono richiesti/istallati i file necessari.
 4. A questo punto viene effettuata l'installazione vera e propria.
 5. Viene lanciato il **Product Configuration**, composto da:
 1. Configurazione macchina: selezionare “Development Machine”
 2. Connattività (lasciare i valori di default).

(continua)

Installazione MySQL – Passi (2)

- 3. Impostazione della **password di root**, digitandola due volte:
ovviamente è importantissimo ricordarla, perchè sarà quella che ci consentirà di amministrare il nostro server!
 - 4. Ci viene chiesto se vogliamo eseguire MySQL come servizio (1), se vogliamo lanciarlo automaticamente (2).
-
- Eseguiamo il comando “Execute” per eseguire la configurazione.
 - Viene eseguito un test di connessione al server per verificarne lo stato.
 - Dopo avere dato tutte le conferme, l'installazione e la configurazione sono terminate! Dovremmo avere il nostro server installato nella directory "[C:/Programmi/MySQL/MySQL Server 5.x/](#)". La sottodirectory "**data**" è quella che contiene fisicamente le tabelle.

Di seguito gli screenshot dell'installazione

License Agreement

MySQL Installer

MySQL. Installer
Adding Community

License Agreement

Choosing a Setup Type

Installation

Installation Complete

License Agreement

To proceed you must accept the Oracle Software License Terms.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble
=====

The licenses for most software are designed to take away your freedom
to share and change it. By contrast, the GNU General Public License is
intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users. This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it. (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.) You can apply it to
your programs, too.

When we speak of free software, we are referring to freedom, not price.
Our General Public Licenses are designed to make sure that you have
the freedom to distribute copies of free software (and charge for this

I accept the license terms

Next > Cancel

Setup Type

MySQL Installer

MySQL. Installer

Adding Community

License Agreement

Choosing a Setup Type

Installation

Installation Complete

Choosing a Setup Type

Please select the Setup Type that suits your use case.

- Developer Default**
Installs all products needed for MySQL development purposes.
- Server only**
Installs only the MySQL Server product.
- Client only**
Installs only the MySQL Client products, without a server.
- Full**
Installs all included MySQL products and features.
- Custom**
Manually select the products that should be installed on the system.

Setup Type Description

Installs the MySQL Server and the tools required for MySQL application development. This is useful if you intend to develop applications for an existing server.

This Setup Type includes:

- * MySQL Server
- * MySQL Workbench
The GUI application to develop for and manage the server.
- * MySQL for Excel
Excel plug-in to easily access and manipulate MySQL data.
- * MySQL for Visual Studio
To work with the MySQL Server from VS.
- * MySQL Connectors
Connector/Net, Java, C/C++, ODBC and others.

< Back Next > Cancel

Requirements

MySQL Installer
Adding Community

License Agreement

Choosing a Setup Type

Check Requirements

Installation

Product Configuration

Installation Complete

Check Requirements

The following products have failing requirements. The installer will attempt to resolve some of this automatically. Requirements marked as manual cannot be resolved automatically. Click on those items to try and resolve them manually.

For Product	Requirement	Status
<input type="radio"/> Connector/Python (3.4) 2.0.3	Python 3.4 is not installed	Manual

< Back Next > Cancel

Installation

MySQL Installer

MySQL. Installer

Adding Community

License Agreement

Choosing a Setup Type

Check Requirements

Installation

Product Configuration

Installation Complete

Installation

Press Execute to upgrade the following products.

Product	Status	Progress	Notes
MySQL Server 5.6.24	Ready to Install		
MySQL Workbench 6.2.5	Ready to Install		
MySQL Notifier 1.1.6	Ready to Install		
MySQL For Excel 1.3.4	Ready to Install		
MySQL for Visual Studio 1.2.3	Ready to Install		
MySQL Fabric 1.5.4 & MySQL Utiliti...	Ready to Install		
Connector/ODBC 5.3.4	Ready to Install		
Connector/C++ 1.1.5	Ready to Install		
Connector/J 5.1.35	Ready to Install		
Connector/.NET 6.9.6	Ready to Install		
Connector/Python (3.4) 2.0.3	Ready to Install		
MySQL Connector/C 6.1.6	Ready to Install		
MySQL Documentation 5.6.24	Ready to Install		

Click [Execute] to install or update the following packages

< Back Execute Cancel

Installation (2)

MySQL Installer

MySQL. Installer

Adding Community

License Agreement

Choosing a Setup Type

Check Requirements

Installation

Product Configuration

Installation Complete

Installation

Press Execute to upgrade the following products.

Product	Status	Progress	Notes
MySQL Server 5.6.24	Complete		
MySQL Workbench 6.2.5	Complete		
MySQL Notifier 1.1.6	Complete		
MySQL For Excel 1.3.4	Complete		
MySQL for Visual Studio 1.2.3	Complete		
MySQL Fabric 1.5.4 & MySQL Utiliti...	Complete		
Connector/ODBC 5.3.4	Complete		
Connector/C++ 1.1.5	Complete		
Connector/J 5.1.35	Installing	25%	
Connector/.NET 6.9.6	Ready to Install		
Connector/Python (3.4) 2.0.3	Ready to Install		
MySQL Connector/C 6.1.6	Ready to Install		
MySQL Documentation 5.6.24	Ready to Install		

Show Details >

< Back Execute Cancel

Product Configuration

The screenshot shows the MySQL Installer window titled "MySQL Installer" with a sub-header "Adding Community". On the left, a vertical navigation pane lists several steps: "License Agreement", "Choosing a Setup Type", "Check Requirements", "Installation", "Product Configuration" (which is highlighted in blue), and "Installation Complete". The main content area is titled "Product Configuration" and contains two informational paragraphs. The first paragraph states: "We'll now walk through a configuration wizard for each of the following products." The second paragraph states: "You can cancel at any point if you wish to leave this wizard without configuring all the products." Below this text is a table showing the status of two products:

Product	Status
MySQL Server 5.6.24	Ready to Configure
Samples and Examples 5.6.24	Ready to Configure

At the bottom right of the window are "Next >" and "Cancel" buttons.

Type and Networking

 MySQL Installer

MySQL® Installer

MySQL Server 5.6.24

Type and Networking

Accounts and Roles

Windows Service

Apply Server Configuration

Type and Networking

Server Configuration Type

Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.

Config Type: Development Machine

Connectivity

Use the following controls to select how you would like to connect to this server.

TCP/IP Port Number: 3306
 Open Firewall port for network access

Named Pipe Pipe Name: MYSQL
 Shared Memory Memory Name: MYSQL

Advanced Configuration

Select the checkbox below to get additional configuration page where you can set advanced options for this server instance.

Show Advanced Options

Next > **Cancel**

Accounts and Roles

MySQL Installer

MySQL Server 5.6.24

Type and Networking

Accounts and Roles

Windows Service

Apply Server Configuration

Accounts and Roles

Root Account Password
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password: ······
Repeat Password: ······

Password Strength: **Strong**

MySQL User Accounts

Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL Username	Host	User Role

Add User

Edit User

Delete

< Back

Next >

Cancel

Windows Service

 MySQL Installer

MySQL. Installer
MySQL Server 5.6.24

Type and Networking
Accounts and Roles
Windows Service
Apply Server Configuration

Windows Service

Configure MySQL Server as a Windows Service

Windows Service Details
Please specify a Windows Service name to be used for this MySQL Server instance. A unique name is required for each instance.
Windows Service Name:

Start the MySQL Server at System Startup

Run Windows Service as ...
The MySQL Server needs to run under a given user account. Based on the security requirements of your system you need to pick one of the options below.

Standard System Account
Recommended for most scenarios.

Custom User
An existing user account can be selected for advanced scenarios.

< Back Next > Cancel

Server Configuration

MySQL Installer
MySQL Server 5.6.24

Type and Networking
Accounts and Roles
Windows Service
Apply Server Configuration

Apply Server Configuration

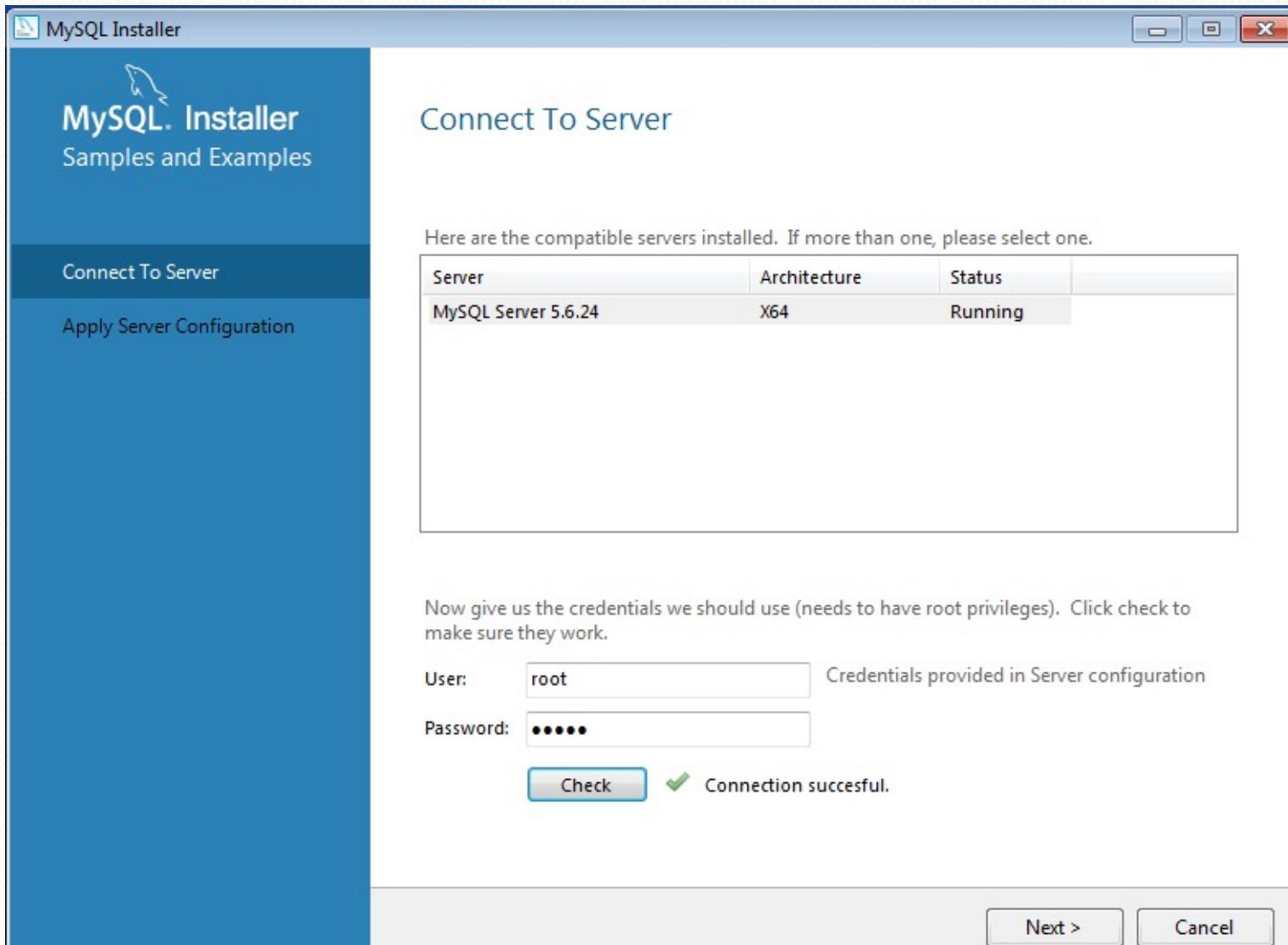
Press [Execute] to apply the changes

Configuration Steps Log

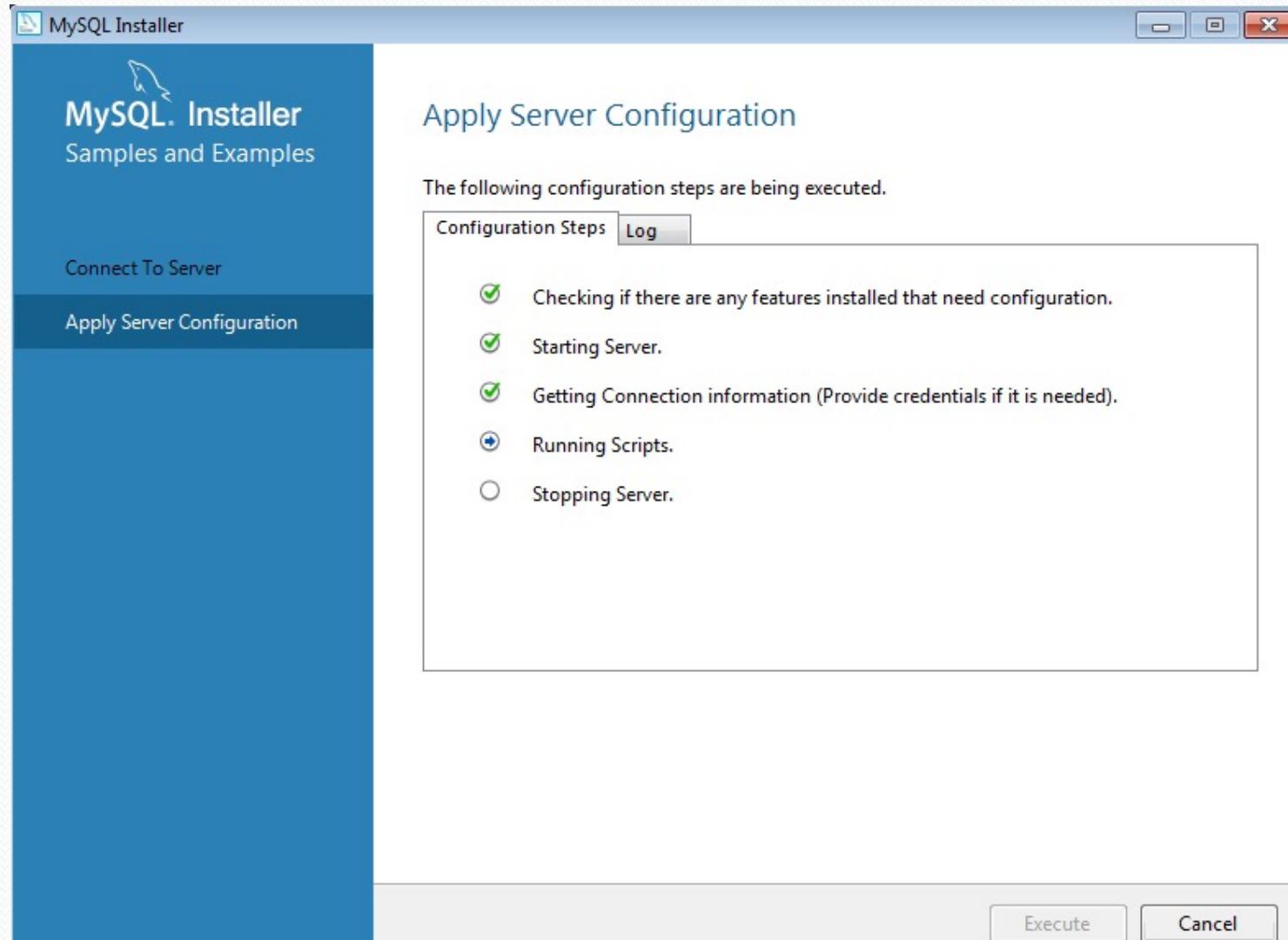
- Stopping Server [if necessary]
- Writing configuration file
- Updating firewall
- Adjusting Windows service [if necessary]
- Starting Server
- Applying security settings
- Creating user accounts
- Updating Start Menu Link

< Back Execute Cancel

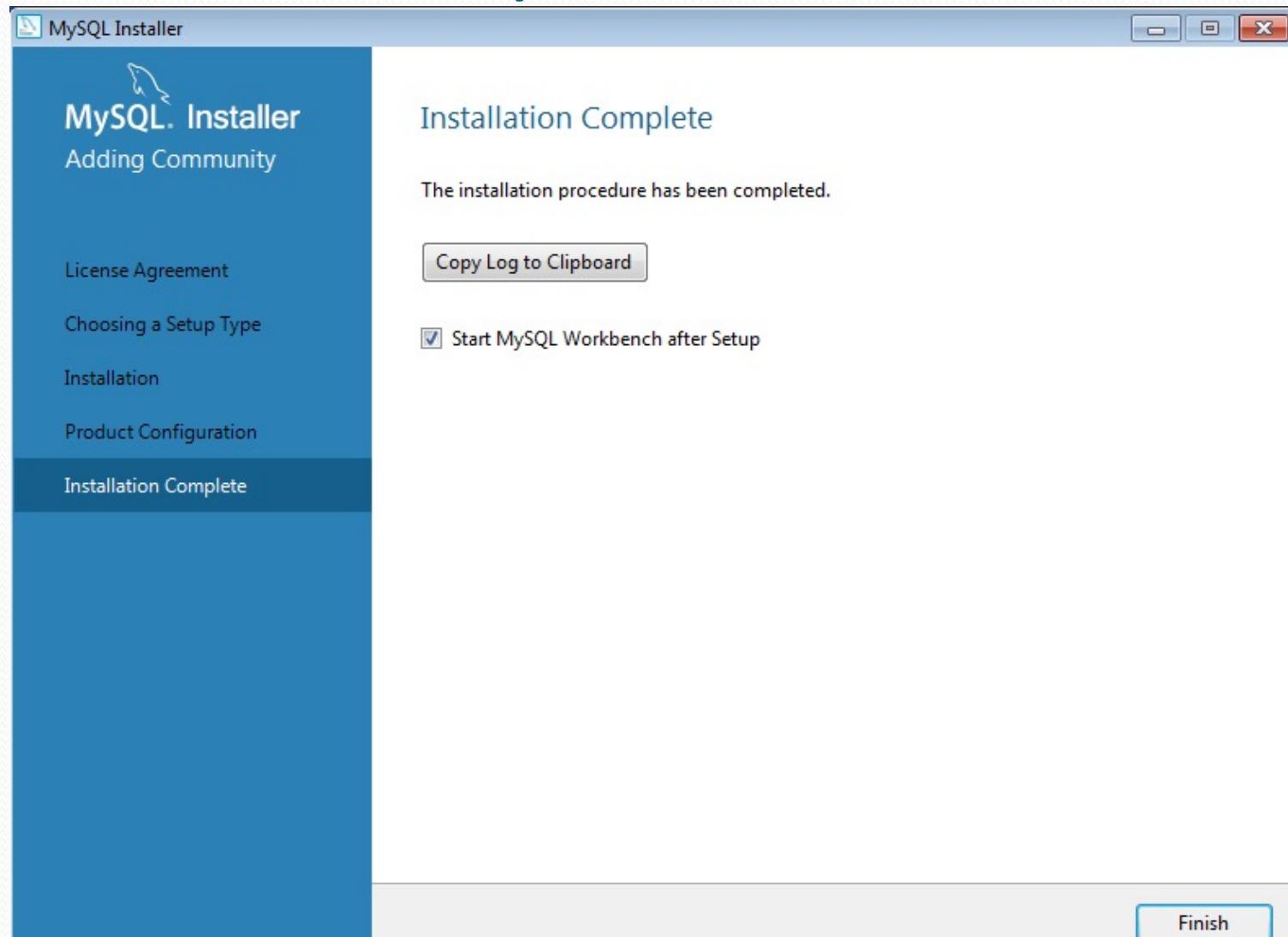
Check the server



Server Configuration



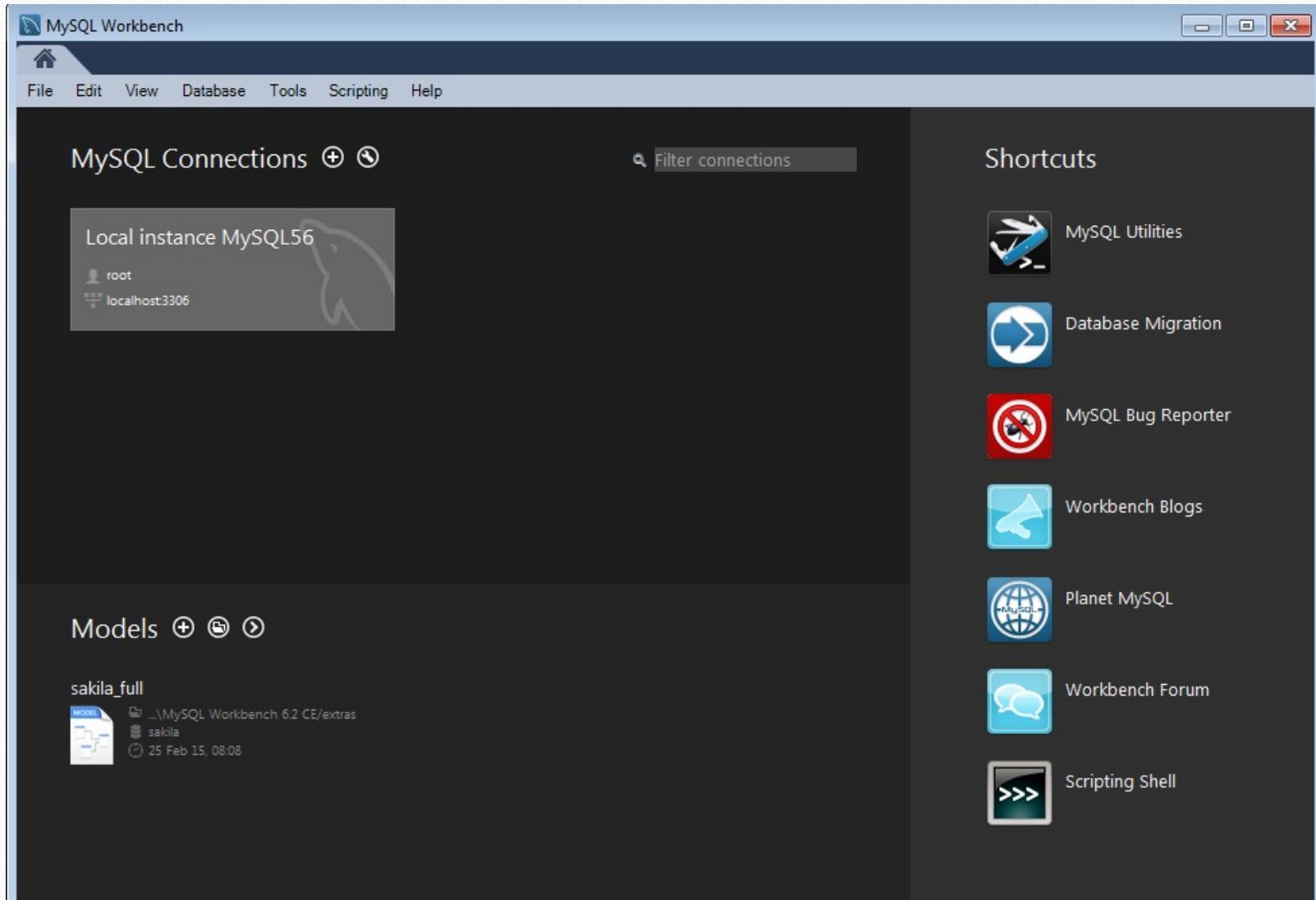
Installation Complete



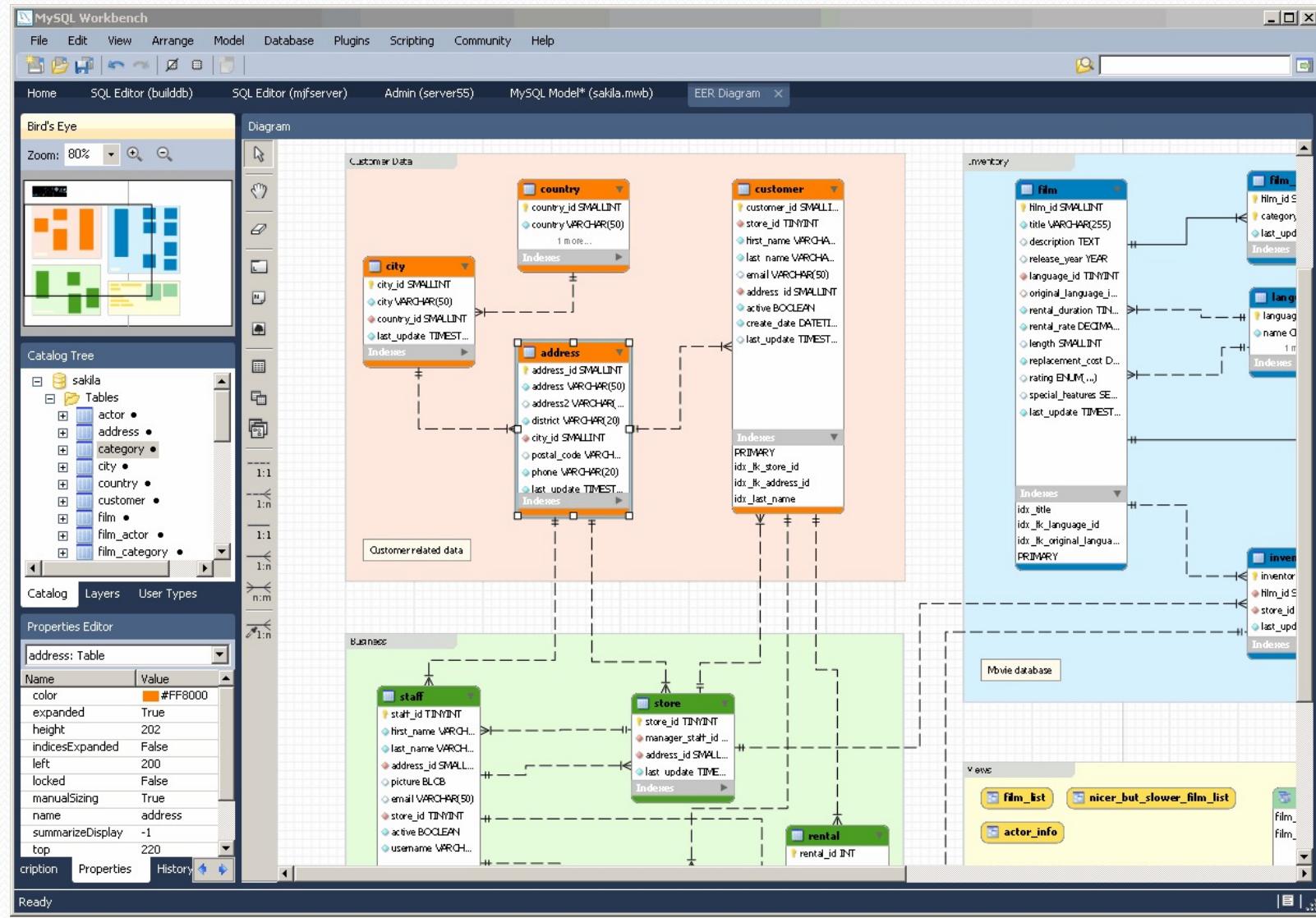
Installazione MySQL – Passi (3)

- MySQL Workbench fornisce ai DBA e gli sviluppatori un tool integrato per:
 - Database Design & Modeling.
 - SQL Development.
 - Database Administration.

MySQL Workbench



Visual Database Design



SQL Editor

MySQL Workbench

File Edit View Query Database Plugins Scripting Community Help

Home SQL Editor (number3server) Admin (webserver) MySQL Model (sakila.mwb) EER Diagram

Object Browser Default: world

- mydb
- sakila
- sakila2
- test
- world
- Tables
- Views
- Routines

SQL Query*

```
123 • COMMIT;
124
125
126 -- Trigger to enforce payment_date during INSERT
127
128
129 • CREATE TRIGGER payment_date BEFORE INSERT ON payment
130   FOR EACH ROW SET NEW.payment_date = NOW();
131
132
133 -- Dumping data for table rental
134
135
136 • SET AUTOCOMMIT=0;
137 • INSERT INTO rental VALUES (1,'2005-05-24 22:53:30',367,130,'2005-05-26 22:04:30',1,'2006-02-15 21:30:53'),(2,'2005-08-18 08:32:33',3049,566,'2005-08-26 03:45:33',2,'2006-02-15 21:30:53'),
138
139 • COMMIT;
140
```

Overview Output History Snippets

mydb MySQL Schema sakila MySQL Schema sakila2 MySQL Schema test MySQL Schema world MySQL Schema

Tables (19 items)

Add Table	actor	address	category	city	country
customer	film	film_actor	film_category	film_text	inventory
language	mynewtable	payment	rental	staff	store
table1	tablename1				

Views (0 items)

Add View

Routines (7 items)

Add Routine	film_in_stock	film_not_in_stock	get_customer_balance	inventory_held_by_c...	inventory_in_stock
	rewards_report	testthat			

Connection Information

Name: number3server
Host: 127.0.0.1:3306
Server: MySQL
Version: 5.5.3-m3-community
User: root

Ready

Administration

MySQL Workbench

Local instance MySQL56

File Edit View Query Database Server Tools Scripting Help

Navigator

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

SCHEMAS

Filter objects

Information

No object selected

Object Info Session

Administration - Server Status × Query 4

Connection Name
Local instance MySQL56

Host: VRlab-PC_02
Socket: MySQL
Port: 3306
Version: 5.6.24-log
Compiled For: MySQL Community Server (GPL)
Configuration File: C:\ProgramData\MySQL\MySQL Server 5.6\my.ini
Running Since: Fri Apr 17 18:52:06 2015 (0:06)

Refresh

Available Server Features

Performance Schema:	<input checked="" type="radio"/> On	SSL Availability:	<input type="radio"/> Off
Thread Pool:	<input type="radio"/> n/a	Windows Authentication:	<input type="radio"/> Off
Memcached Plugin:	<input type="radio"/> n/a	Password Validation:	<input type="radio"/> n/a
Semisync Replication Plugin:	<input type="radio"/> n/a	Audit Log:	<input type="radio"/> n/a

Server Directories

Base Directory:	C:\Program Files\MySQL\MySQL Server 5.6\
Data Directory:	C:\ProgramData\MySQL\MySQL Server 5.6\Data\
Disk Space in Data Dir:	388.00 GB of 452.00 GB available
Plugins Directory:	C:\Program Files\MySQL\MySQL Server 5.6\lib\plugin\
Tmp Directory:	C:\Windows\TEMP\~2\NETWOR~1\AppData\Local\Temp

Server Status

Running

CPU 1%

Connections 4

Traffic 18.20 KB/s

Key Efficiency 100.0%

Selects per Second 0

InnoDB Buffer Usage 8.9%

InnoDB Reads per Second 0

InnoDB Writes per Second 0

Primi passi con MySQL

- In MySQL si può operare contemporaneamente con più di un database.
 - Lo stesso server può essere utilizzato per più di una applicazione.
 - Ogni applicazione opera su un database distinto, e di solito, la gestione di ogni database è affidata a uno o più utenti.
- L'installazione crea un utente di default per connettersi al db, chiamato **root** e con la password inserita durante l'installazione.

Primi passi con MySQL (2)

- Nella configurazione di default, gli utenti hanno accesso soltanto al database **test** tranne un utente particolare, l'utente **root**, che ha accesso totale al server MySQL e può anche creare nuovi database.
- Se vogliamo collegarci col nome utente root, basta dare il comando: **mysql -u root -p**
 - L'opzione **-p** implica l'inserimento della password
 - L'opzione **-h** è usata per indicare la macchina server a cui collegarsi. Ad esempio:
mysql -h mysql.csedu-priv -u sistemi
si collega al computer il cui numero IP è 192.168.7.40 con il nome utente **sistemi**.
- La sintassi completa: **mysql -h <host> -u <utente> -p <password>**

MySQL (sotto Windows)

```
C:\> cd mysql\bin  
C:\mysql\bin> mysql -h host -u user_name -p  
C:\mysql\bin> Enter password: *****
```

- Il messaggio per un utente autorizzato:

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 8.0.13 MySQL Community Server – GPL

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names
may be trademarks of their respective owners.

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

mysql>

- e attende l'immissione dei comandi. I comandi devono esseri comandi SQL validi
(riconosciuti da MySQL) e terminati da un punto e virgola.

Attenzione!

- **mysql>** è il prompt di MySQL! Non confonderlo con il prompt della shell! Dalla shell si possono invocare altri programmi e dare comandi al sistema operativo. Dal prompt di MySQL, invece, si possono inserire dei comandi in SQL che vengono eseguito dal server del database. Se provate un comando come **ls;** (sotto LINUX) il sistema risponde con:

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'ls' at line 1

- Questo è un messaggio di errore standard di MySQL. C'è un numero che identifica il tipo di errore seguito dalla sua descrizione.

Utenti di MySQL

- Un nome utente (**id-utente**) viene specificato con la seguente sintassi:
<nomeutente>@<macchina da cui si collega>

Esempio: per specificare l'utente **pippo**, collegato dalla stessa macchina dove risiede il server, si usa **pippo@localhost**, mentre per **pluto** collegato da **pc-clei-001** si ha **pluto@pc-clei-001**.

- Il comando **GRANT SELECT ON *.* to 'pippo'@'localhost'** consente, all'utente **pippo**, quando si connette da **localhost**, di **accedere in lettura** (**privilegio**) a **qualsiasi** database presente nel computer. Non consente tuttavia alcuna modifica dei dati.
- Per eliminare i diritti già concessi, è possibile usare l'istruzione REVOKE:

**REVOKE <lista privilegi> ON <nome database>.<nome tabella>
FROM <lista utenti>**

- ATTENZIONE! Il diritto **USAGE** ("no privileges") non si può revocare con il comando REVOKE.
- Per elencare i privilegi:

SHOW GRANTS;

Diritti a livello di server

- È possibile concedere diritti ad un utente su tutti i dati di un server, su uno specifico database, su un tabella o su specifiche colonne di una tabella:

GRANT <lista privilegi> ON <nome database>.<nome tabella> TO <lista utenti>

- Alcuni dei privilegi che è possibile concedere sono:
 - **ALTER**: consente l'uso del comando ALTER TABLE (non è possibile creare o eliminare indici);
 - **CREATE**: consente l'uso di CREATE TABLE e CREATE DATABASE;
 - **DELETE**: consente l'uso del comando DELETE;
 - **DROP**: consente l'uso di DROP TABLE e DROP DATABASE;
 - **INSERT**: consente l'uso del comando INSERT;
 - **SELECT**: consente l'uso del comando SELECT;
 - **UPDATE**: consente l'uso del comando UPDATE;
 - **REFERENCES**: consente l'accesso alle chiavi esterne;
 - **INDEX**: consente la creazione ed eliminazione di indici con il comando ALTER TABLE.
 - **ALL o ALL PRIVILEGES**: concede tutti i privilegi possibili.
 - **USAGE**: non concede nessun privilegio specifico tranne il diritto di collegarsi al server. Tutti i privilegi includono automaticamente anche **USAGE**.
 - Dal MySQL 8 non è più possibile (implicitamente) creare un utente utilizzando il comando GRANT. Usare CREATE USER
 - Inoltre, è necessario creare prima il database e eventualmente le tabelle in esso contenute se esplicitamente referenziate

GRANT option

- Se concediamo dei diritti ad un utente, possiamo anche consentirgli, a sua volta, di concedere questi diritti ad altri utenti. Per far ciò basta aggiungere la clausola **WITH GRANT OPTION** alla fine del comando **GRANT**:
 - l'utente **pippo@localhost** deve essere creato prima
 - **airdb** deve essere creato prima
 - non è necessario creare le tabelle in **airdb**

**GRANT ALL PRIVILEGES ON airdb.* TO 'pippo'@'localhost'
WITH GRANT OPTION;**

- In questo modo non solo **pippo@localhost** potrà utilizzare come vuole il database **airdb**, ma potrà a sua volta concedere diritti su questo database ad altri utenti. Notare che un utente non può mai concedere diritti superiori di quelli di cui dispone.
- Per revocare il diritto di grant, si usa REVOKE con il privilegio GRANT OPTION, ovvero:
REVOKE GRANT OPTION ON airdb.* FROM 'pippo'@'localhost';

Diritti a livello di tabella e colonna

- A livello di singola tabella. Sintassi:
**GRANT <lista privilegi> ON <nome db>.<nome tabella>
TO <lista utenti>**
- Stessi privilegi dei diritti a livello di server o di database. L'unica differenza è che i diritti **CREATE** o **DROP** a livello database consentono di eseguire le istruzioni CREATE DATABASE e DROP DATABASE, mentre in un diritto a livello di singola tabella, è possibile solo utilizzare la CREATE TABLE e la DROP TABLE.
- Per fornire i privilegi a livello di colonna. Sintassi:
**GRANT <privilegio>(<lista colonne>) ON <nome db>.<nome tabella>
TO <lista utenti>**
- Gli unici privilegi che si possono usare sono **ALTER**, **DELETE**, **INSERT**, **SELECT** **UPDATE**, e **REFERENCES**.

Diritti a livello di tabella e colonna (2)

- Con il comando
GRANT SELECT(id) ON airdb.aeroporti TO 'pippo'@'localhost';
- l'utente **pippo@localhost** potrà accedere al database **airdb** e dare il comando:
SELECT id FROM aeroporti;
- ma non il comando
SELECT * FROM aeroporti;
- SELECT e UPDATE sono gestiti in maniera indipendente; avere il privilegio di UPDATE non implica la possibilità di eseguire la SELECT (e viceversa):

GRANT UPDATE(modello, produttore) ON airdb.aerei TO 'pluto'@'localhost';
GRANT SELECT(modello, produttore) ON airdb.aerei TO 'pippo'@'localhost';

Autenticazione tramite password

- **ATTENZIONE!** *La prima cosa che un DBA deve fare quando installa MySQL su una macchina condivisa tra più utenti, è assicurarsi che l'accesso al database con l'account root sia protetto da password!*
- Per visualizzare gli utenti creati e relative password (criptate), selezionare il db MySQL e digitare il seguente comando:

USE mysql;

SELECT host, user, authentication_string FROM user;

host	user	authentication_string
%.	sistemi	\$A\$005\$nD~h6[AYuSDP&fC;A0Zh76MEMcjh8Bgt0gf3TC4fUv1kiuM.q1XfWw19qUqpZP9
localhost	mysql.infoschema	\$A\$005\$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBEUSED
localhost	mysql.session	\$A\$005\$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBEUSED
localhost	mysql.sys	\$A\$005\$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBEUSED

Autenticazione tramite password (2)

- Per creare un utente MySQL:

CREATE USER <id-utente> IDENTIFIED BY <password>;

CREATE USER 'pippo'@'localhost' IDENTIFIED BY 'pluto';

CREATE USER 'pippo'@'localhost';

CREATE USER 'pippo'@'%' IDENTIFIED BY 'pluto';

- Per cancellare un utente MySQL:

DROP USER <id-utente>;

DROP USER 'pippo'@'localhost';

DROP USER 'pippo'@'%';

DROP USER IF EXISTS 'pippo'@'localhost';

- Per cambiare la password di un utente si può utilizzare l'istruzione:

SET PASSWORD FOR 'pippo'@'localhost' = 'pluto';

- oppure:

ALTER USER 'pippo'@'localhost' IDENTIFIED BY 'pluto';

Sicurezza in MySQL

- Ogni qualvolta noi ci colleghiamo ad un server MySQL, esso ci identifica sulla base di due informazioni (**id-utente**): ***il nome utente e il computer da cui ci colleghiamo.***
- Possiamo conoscere entrambe queste informazioni con la funzione `USER()`:

```
SELECT user();
```

```
+-----+  
| user() |  
+-----+  
| root@localhost |  
+-----+  
1 row in set (0.00 sec)
```

- Indica che sono stato riconosciuto come l'utente collegato da localhost (un nome speciale che indica lo stesso computer dove risiede il server MySQL).

Esercizio 1

- Usare il comando GRANT per concedere i seguenti diritti di accesso:
 - a) *l'accesso in lettura e scrittura per tutti i database all'utente “studente”, quando si connette da localhost, senza bisogno di immettere nessuna password;*
 - b) *l'accesso in lettura e scrittura per tutti i database all'utente “studente” anche quando si connette da una qualunque altra macchina in rete, purché con una password;*
 - c) *l'accesso in lettura al solo attributo id della tabella aerei, e alle tabelle voli e prenotazioni del db “airdb”; all'utente “pilota”, da qualunque macchina, senza password.*

Soluzione Esercizio 1

Fare attenzione agli apici!
Il simbolo % richiede gli apici!

- Usare il comando GRANT per concedere i seguenti diritti di accesso:

a) l'accesso in lettura e scrittura per tutti i database all'utente studente, quando si connette da localhost, senza bisogno di immettere nessuna password;

CREATE USER 'studente'@'localhost';

GRANT SELECT, INSERT ON *.* TO 'studente'@'localhost';

b) l'accesso in lettura e scrittura per tutti i database all'utente studente anche quando si connette da una qualunque altra macchina in rete, purché con una password;

CREATE USER 'studente'@'%' IDENTIFIED BY 'password';

GRANT SELECT, INSERT ON *.* TO 'studente'@'%';

b) l'accesso in lettura al solo attributo id della tabella aerei, e alle tabelle voli e prenotazioni, all'utente pilota, da qualunque macchina, senza password;

CREATE DATABASE airdb;

CREATE TABLE aerei (id INT, ...

CREATE TABLE voli ...

CREATE TABLE prenotazioni ...

CREATE USER 'pilota'@'%';

GRANT SELECT(id) ON airdb.aerei TO 'pilota'@'%';

GRANT SELECT ON airdb.voli TO 'pilota'@'%';

GRANT SELECT ON airdb.prenotazioni TO 'pilota'@'%';

Vedremo successivamente
come creare le tabelle

Creazione di un semplice database (utente in localhost)

Passo	Descrizione	Comando
1	Lanciare la <i>command shell</i>	Digitare cmd e premere OK
2	Login in MySQL come root	mysql -u root -p
3	Creare un database per gestire i dati	CREATE DATABASE sistemi;
4	Creare l' utente sistemi con password	CREATE USER 'sistemi'@'localhost' IDENTIFIED BY 'sistemi';
5	Assegnare tutti i privilegi sul database sistemi all'utente sistemi	GRANT ALL ON sistemi.* TO 'sistemi'@'localhost';
6	Uscire dal tool MySQL	quit;
7	Login in MySQL come sistemi	mysql -u sistemi -p;
8	Utilizzare il database sistemi	USE sistemi;

Per questioni di sicurezza: non utilizzare l'utente admin per accedere alle funzionalità di un particolare database (di un progetto), creare sempre un utente specifico

Creazione di un semplice database (utente remoto)

Passo	Descrizione	Comando
1	Lanciare la <i>command shell</i>	Digitare cmd e premere OK
2	Login in MySQL come root	mysql -u root -p
3	Creare un database per gestire i dati	CREATE DATABASE sistemi;
4	Creare l' utente sistemi con password	CREATE USER 'sistemi'@'%' IDENTIFIED BY 'sistemi';
5	Assegnare tutti i privilegi sul database sistemi all'utente sistemi	GRANT ALL ON sistemi.* TO 'sistemi'@'%';
6	Uscire dal tool MySQL	quit;
7	Login in MySQL come sistemi	mysql -u sistemi -p;
8	Utilizzare il database sistemi	USE sistemi;

Fare attenzione agli apici!

Il simbolo % richiede necessariamente gli apici!

Alcuni comandi utili per iniziare

- **SELECT version();** -- restituisce il numero di versione del DBMS
- **SELECT now(), current_date();** -- restituisce data-ora e data corrente

now()	current_date()
2019-11-29 13:06:45	2019-11-29

- **SELECT now(), current_date() FROM dual;**
- **SHOW databases;** -- mostra i database presenti nell' installazione corrente si MySQL (inizialmente solo quelli di default)

Comandi MySQL

- **USE <nome_db>;** -- seleziona il database “**nomedb**” come database corrente. Tutte le operazioni SQL da quel momento agiranno all'interno del database selezionato
- **SELECT database();** --mostra il database corrente;
- **SHOW tables [from <nome_db>];** -- mostra le tabelle che fanno parte di un database (opzionale). Se non si specifica nessun database, si considera il database corrente, impostato tramite il comando **USE**

Comandi MySQL (2)

- **SELECT 1+1, 2+2;** -- uso di MySQL come calcolatrice
- **QUIT;** -- per uscire dal client
- Con **enter** si possono continuare i comandi sulle righe successive (fino al punto e virgola: “;”), se si decide di annullare un comando possiamo farlo digitando “\c”.

Comments

SELECT 1+1; # This comment continues to the end of line

SELECT 1+1; -- This comment continues to the end of line

SELECT 1 /* this is an in-line comment ***/ + 1;**

SELECT 1+
/* this is a
multiple-line comment ***/**
1;

Creare e cancellare un DB

- **CREATE DATABASE mydata;** -- crea il database “mydata”
- **GRANT ALL ON mydata.* to <id-utente>;** -- solo un particolare utente può accedere al database
- **USE mydata;** -- per usare il database “mydata”
- **DROP DATABASE mydata;** -- per cancellare il database “mydata”
- **DROP DATABASE IF EXISTS <nome_db>;** -- cancella il database <nomedb> se esiste
 - A differenza di **DROP DATABASE <nome_db>**, la variante con **IF EXISTS** non da errore nel caso il database <nome_db> non esiste.

Operare su una tabella

- Per creare una tabella si utilizza: **CREATE TABLE**
- La sua sintassi è la seguente:

```
CREATE TABLE <nome_tabella> (
    nome_colonna-1 tipo [modificatori]
    [, nome_colonna-2 tipo [modificatori]]
    ...
);
```

- Creiamo quindi una tabella per i dipendenti di un ufficio:

```
CREATE TABLE dipendenti (
    codfiscale      CHAR(16)      PRIMARY KEY,
    nome            CHAR(30),
    cognome         CHAR(30),
    datanascita     DATE,
    tempopieno      ENUM('y','n')  NOT NULL,
    anniserv        INT          DEFAULT 0
);
```

Si possono utilizzare anche chiavi esterne con più attributi di una stessa tabella

Operare su più tabelle

```
DROP TABLE IF EXISTS impiegati; B ordine di cancellazione  
DROP TABLE IF EXISTS dipartimenti; A
```

```
CREATE TABLE dipartimenti A ordine di creazione  
(  
    dept_id  SMALLINT UNSIGNED          NOT NULL,  
    nome     VARCHAR(20)                 NOT NULL,  
    PRIMARY KEY (dept_id)  
);
```

```
CREATE TABLE impiegati B  
(  
    codfiscale      CHAR(16)            NOT NULL,  
    nome            VARCHAR(30)          NOT NULL,  
    cognome         VARCHAR(30)          NOT NULL,  
    superiore_cf    CHAR(16),  
    dept_id         SMALLINT UNSIGNED,  
    datanascita     DATE,  
    tempopieno     ENUM('Y','N')        NOT NULL,  
    anniserv        INT                 DEFAULT 0,  
    FOREIGN KEY (superiore_cf) REFERENCES impiegati (codfiscale),  
    FOREIGN KEY (dept_id) REFERENCES dipartimenti (dept_id),  
    PRIMARY KEY (codfiscale)  
);
```

Composizione di uno script (.sql)

DROP DATABASE IF EXISTS sistemi;

CREATE DATABASE sistemi; -- creazione database

USE sistemi;

DROP USER IF EXISTS 'sistemi@'% ;

CREATE USER 'sistemi'@ '%' **IDENTIFIED BY** 'sistemi'; -- creazione utente

GRANT ALL ON sistemi.* **TO** 'sistemi'@ '%' -- privilegi

DROP TABLE IF EXISTS dipendenti;

CREATE TABLE dipendenti (-- creazione tabella/e

...

);

-- popolamento

INSERT INTO dipendenti (codfiscale, nome, cognome, datanascita, tempopieno, anniserv)

VALUES ('MRCVRD82H25B123Q', 'Marco', 'Verdi', '1982-06-25', 'y', 3);

INSERT INTO dipendenti (codfiscale, nome, cognome, datanascita, tempopieno, anniserv)

VALUES ('ANDBCH87C21G554M', 'Andrea', 'Bianchi', '1987-04-21', 'n', 1);

...

Tipi di Dato

- Particolare attenzione va posta nella scelta del tipo di dato più adatto per ogni campo.
 - I libri su MySQL contengono una trattazione dei vari tipi di dato disponibili.
- Ne vediamo alcuni:
 - Valori interi
 - Valori in virgola mobile
 - Valori stringa
 - Tipi insiemi ed enumerati
 - Valori date e time
 - ...

Valori interi

- Sono usati per rappresentare numeri interi positivi e/o negativi. Ne esistono varie varianti, a seconda del numero di bit destinati alla codifica del numero, e quindi dell'intervallo massimo di valori ammissibili.
 - **TINYINT**: 8 bit, -128...127
 - **SMALLINT**: 16 bit, -32.768 ... 32.767
 - **MEDIUMINT**: 24 bit, -8.338.608 ... 8.338.607
 - **INT**: 32 bit, -2.147.483.648 ... 2.147.483.647
 - **BIGINT**: 64 bit, -9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807
- *Tutti i tipi interi posso essere dichiarati UNSIGNED*. In questo caso il valore massimo dei dati aumenta: TINYINT passa da -128.. 127 a 0..255, e così via. Il modificatore UNSIGNED va messo dopo il tipo e non prima, come di solito si fa in Java e in C.

CREATE TABLE tavola (attributo INT);

INSERT INTO tavola VALUES (-10); -- ok

CREATE TABLE tavola_plus (attributo INT UNSIGNED);

INSERT INTO tavola_plus VALUES (-10); -- out of range value

Valori in virgola mobile

- Sono usati per rappresentare numeri "con la virgola". Anche questi si distinguono in base al numero di bit usati per la rappresentazione, che influisce sia sull'intervallo massimo di valori rappresentabili che sul numero di cifre significative.
 - **FLOAT(p, s)**: 32 bit, singola precisione.
Valori ammessi: da -3.402823466E+38 a -1.175494351E-38, \mathbb{Q} , e 1.175494351E-38 a 3.402823466E+38
 - **DOUBLE(p, s)**: 64 bit, doppia precisione.
Valori ammessi: da -1.7976931348623157E+308 a -2.2250738585072014E-308, \mathbb{Q} , e 2.2250738585072014E-308 to 1.7976931348623157E+308
 - *Tutti i tipi in virgola mobile posso essere dichiarati UNSIGNED.*
 - ‘p’ per precision (numero totale cifre) e ‘s’ per scale (numero cifre decimali).
- CREATE TABLE floating (f FLOAT(4, 2));** -- max 99.99
INSERT INTO floating VALUES (17.8675); -- arrot. 17.87

Valori stringa

- **CHAR(n)**: stringa di n caratteri a lunghezza fissa. Se la stringa è più corta, viene riempita di spazi alla fine.
- **VARCHAR(n)**: stringa di al massimo n caratteri, $n \leq 255$.
- I tipi a lunghezza fissa come CHAR(n) creano tabelle più efficienti di quelli a lunghezza variabile, a fronte di uno spreco di memoria di massa.
- **TINYTEXT(n)**: come VARCHAR(n)
- **TEXT(n)**: come VARCHAR(n) ma n fino a $65535 = 2^{16}$
- **MEDIUMTEXT(n)**: come VARCHAR(n) ma n fino a 2^{24}
- **LONGTEXT(n)**: come VARCHAR(n) ma n fino a 2^{32}

BLOB per archiviare immagini/file

- In MySQL esiste un particolare tipo di dato che si presta ad ospitare grandi volumi di dati:
 - **BLOB** (*Binary Large data OBject*).
 - All'interno di un campo di tipo BLOB, ad esempio, è possibile archiviare oggetti come immagini, PDF o video.

```
CREATE TABLE immagini (
    id UNSIGNED INT NOT NULL,
    dati BLOB DEFAULT NULL,
    PRIMARY KEY(id);
);
```

Non funziona!!!

```
INSERT INTO immagini (id, dati) VALUES (1, LOAD_FILE('foto.jpg'));
```

Tipi insiemi ed enumerati

- **ENUM('val₁', 'val₂',..., 'val_n')**: un valore qualunque tra quelli indicati. In realtà è ammesso anche il valore " (stringa nulla).
- **SET('val₁', 'val₂',..., 'val_n')**: uno o più valori tra quelli indicati.
 - MySQL, per i valori ENUM usa un numero da 1 e 65535 per ogni valore elencato (e usa 0 per la stringa vuota).
 - Per i valori SET usa un numero a 64 bit, dove ogni bit rappresenta un possibile elemento dell'insieme (il bit è a 1 se l'elemento fa parte dell'insieme, a 0 altrimenti).

Tipi SET

```
CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
                    valori ridondanti non considerati
INSERT INTO myset (col) VALUES ('a,b'), ('c,a'), ('a,b,a'), ('a,d,b'), ('d,a,d');
```

- Se si setta una colonna di tipo SET con un valore non supportato, viene mostrato un messaggio di errore:

```
INSERT INTO myset (col) VALUES ('a,d,d,s'); -- data truncated
```

...

SHOW ERRORS;

Level	Code	Message
Error	1265	Data truncated for column 'col' at row 1

- Per selezionare:

```
SELECT * FROM myset WHERE col ='a,b';
```

```
SELECT * FROM myset WHERE col LIKE '%a,b%';
```

```
SELECT * FROM myset WHERE col LIKE '%a%';
```

```
SELECT * FROM myset WHERE col LIKE '%a__';
```

Tipi data

- **DATE**: data nel formato **aaaa-mm-gg** (**YYYY-MM-DD**)
- **DATETIME**: ora e data nel formato **aaaa-mm-gg hh:mi:ss**
- **TIMESTAMP**: ora e data nel formato **aaaa-mm-gg hh:mi:ss**
- **YEAR**: **YYYY** (1901 to 2155)
- **TIME**: **hhh:mi:ss**
- I valori di tipo DATE coprono un range che va dal 1 gennaio 1 DC al 31 dicembre 9999 DC.
- Tuttavia, è possibile specificare tutta una serie di date "*non valide*" come il 0000-1-1 (non esiste l'anno zero). L'idea è che, *non controllando la validità delle date, il server MySQL è più efficiente*:
 - controlli devono essere effettuati da parte del software applicativo che utilizza MySQL come server database.

```
CREATE TABLE mydate (data DATE);
INSERT INTO mydate VALUES ('2020-11-12');
INSERT INTO mydate VALUES ('0000-01-01');
```

- 0000-00-00 viene utilizzata da MySQL automaticamente quando l'utente specifica una stringa che il server non riesce a interpretare come data.
- DATETIME, in più, consente di specificare un'ora da 00:00:00 a 23:59:59.

Esempio con TIMESTAMP (DEFAULT e ON UPDATE)

```
CREATE TABLE time (ts TIMESTAMP);
```

...

```
INSERT INTO time VALUES (NOW());
```

```
INSERT INTO time VALUES (CURRENT_TIMESTAMP);
```

ts
2020-12-11 16:14:18
2020-12-11 16:14:18

```
CREATE TABLE timeup (
    id INTEGER NOT NULL,
    ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        ON UPDATE CURRENT_TIMESTAMP);
```

...

```
INSERT INTO timeup(id) VALUES (1);
```

```
INSERT INTO timeup(id) VALUES (3);
```

id	ts
1	2020-12-11 16:16:31
3	2020-12-11 16:16:31

...

```
UPDATE timeup SET id = 2 WHERE id =1;
```

id	ts
2	2020-12-11 16:17:34
3	2020-12-11 16:16:31

AUTO_INCREMENT

- Per la scelta della chiave primaria:
 - o si usano attributi specifici della tabella come chiave primaria (ad esempio il codice fiscale),
 - oppure conviene usare delle chiavi primarie "sintetiche" (surrogate), di tipo intero.

```
CREATE TABLE persona (
    valint      INT PRIMARY KEY,
    nome        CHAR(20),
    cognome     CHAR(20),
    nascita      DATE
);
```

- Per inserire valori in questa tabella dobbiamo specificare anche il valore di VALINT, con comandi del tipo:

```
INSERT INTO persona VALUES (1,'mario','rossi','2002-12-3');
INSERT INTO persona VALUES (2,'andrea','bianchi','2004-11-9');
INSERT INTO persona VALUES (1,'sandro','verdi','2003-02-02');
-- duplicate entry '1' for key 'PRIMARY'
```

AUTO_INCREMENT (2)

- Il problema è che, visto che VALINT non ha alcun significato, non è facile decidere un valore appropriato, visto che bisogna stare attenti a sceglierne uno che non è già presente nella tabella. Possiamo semplificarcici il lavoro specificando la chiave VALINT come AUTO_INCREMENT.

```
ALTER TABLE persona CHANGE valint valint INT AUTO_INCREMENT;
INSERT INTO persona (nome, cognome, nascita)
VALUES ('sandro','verdi','2003-02-02');
```

valint	nome	cognome	nascita
1	mario	rossi	2002-12-03
2	andrea	bianchi	2004-11-09
3	sandro	verdi	2003-02-02

Lo scopo di InnoDB è quello di associare maggiore sicurezza (intesa soprattutto come consistenza ed integrità dei dati) a performance elevate.

AUTO_INCREMENT (3)

```
CREATE TABLE persona_auto (
    autovalint      INT          NOT NULL AUTO_INCREMENT,
    nome CHAR(20)    DEFAULT NULL,
    cognome        CHAR(20)    DEFAULT NULL,
    nascita         DATE        DEFAULT NULL,
    PRIMARY KEY (`autovalint`)
) ENGINE=InnoDB           AUTO_INCREMENT=3;
```

- Per inserire i dati:

```
INSERT INTO persona_auto (nome, cognome, nascita)
VALUES ('mirco', 'verdi', '2001-06-28');
```

```
INSERT INTO persona_auto (nome, cognome, nascita)
VALUES ('mirco', 'verdi', '2001-06-28');
```

autovalint	nome	cognome	nascita
3	mirco	verdi	2001-06-28
4	mirco	verdi	2001-06-28

Cancellare una tabella

- Vediamo come si può cancellare una tabella o il suo contenuto:

DROP TABLE <nome_tabella>; -- elimina una tabella

DELETE FROM <nome_tabella>; -- elimina il contenuto di una tabella

DROP TABLE dipendenti;

DELETE FROM persona;

DROP TABLE persona;

- **ATTENZIONE !!** Una tabella eliminata con **DROP TABLE** sparisce dal database e se si vuole riutilizzare bisogna crearla di nuovo.
- **DELETE FROM** invece svuota il contenuto della tabella, ma questa rimane nel database ed è possibile in qualunque momento inserire nuovi elementi con il comando **INSERT**.

Insert

- È possibile inserire valori nella tabella con:

```
INSERT INTO dipendenti VALUES ('codfisc1','mario','rossi','1960-12-3','y',1);
```

- **ATTENZIONE!!!** Quando si inseriscono i valori in una tabella con il comando INSERT, dopo la parola chiave VALUES vanno specificati tanti parametri quanti sono i campi nella tabella che si vuole modificare.

```
INSERT INTO dipendenti VALUES ('codfisc1','mario','rossi','1960-12-3',1);
```

ERROR 1136 (21S01): Column count doesn't match value count at row 1

- Alternativa:

```
INSERT INTO dipendenti(codfiscale, nome, cognome)  
VALUES ('codfisc2','mario','bianchi');
```

- Al solito, dopo VALUES bisognerà specificare tanti elementi quanti sono i campi indicati dopo il nome della tabella.

Visualizzare il contenuto delle tabelle

- Possiamo vedere quello che abbiamo inserito con:

```
SELECT * FROM dipendenti;
```

```
+-----+-----+-----+-----+-----+-----+
| codfiscale | nome    | cognome | datanascita | tempopieno | anniserv |
+-----+-----+-----+-----+-----+-----+
| codfisc1   | mario   | rossi   | 1960-12-03  | y          | 1          |
| codfisc2   | mario   | bianchi | NULL        | y          | 0          |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

- Notare i valori di default per i campi *datanascita*, *tempopieno* e *anniserv*.

Descrizione di tabelle

- Se invece si vuole vedere non il contenuto della tabella, ma la descrizione dei campi che la compongono, si può usare il comando:

DESCRIBE <nometabella> ; oppure **DESC <nometabella>** ;

DESCRIBE dipendenti; -- restituisce il seguente risultato

Field	Type	Null	Key	Default	Extra
codfiscale	char(16)	NO	PRI	NULL	
nome	char(30)	YES		NULL	
cognome	char(30)	YES		NULL	
datanascita	date	YES		NULL	
tempopieno	enum('y', 'n')	NO		NULL	
anniserv	int(11)	YES		0	

6 rows in set (0.00 sec)

- Il campo **Null** indica se i corrispondenti attributi possono assumere valori nulli oppure no, il campo **Key** specifica qual è la chiave primaria della tabella, e il campo **Default** indica i valori di default dei vari attributi.

DESCRIBE timeout; -- attributo ts definito con le clausole **DEFAULT** e **ON UPDATE**

Field	Type	Null	Key	Default	Extra
id	int(11)	NO		NULL	
ts	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED on update CURRENT_TIMESTAMP

2 rows in set (0.00 sec)

Esercizio 2

- Iniziamo la creazione della nostra applicazione di prenotazione online. Si vuole creare una tabella che contiene informazioni sugli aerei della nostra compagnia aerea.
- I campi di questa tabella (“aerei”) devono essere:
 - *id*, di tipo *CHAR(20)* che funge da chiave primaria.
 - *produttore*
 - *modello*
 - *dataimm* che contiene la data di immatricolazione (primo volo) dell'aereo.
 - *numposti*, che contiene il numero di posti disponibili sull'aereo.
- I campi produttore e modello non possono essere NULL.

Soluzione

DROP database IF EXISTS airdb;

CREATE database airdb;

USE airdb;

DROP TABLE IF EXISTS aerei;

CREATE TABLE aerei (

id	CHAR(20)	PRIMARY KEY,
produttore	CHAR(20)	NOT NULL,
modello	CHAR(10)	NOT NULL,
dataimm	DATE,	
numposti	INT	

);

Esercizio 3

- Successivamente, riempire la tabella in modo che il comando **SELECT * FROM aerei;** dia il seguente risultato:

id	produttore	modello	dataimm	numposti
superjet	boeing	747	2000-10-10	350
minijet	MDD	Super80	NULL	NULL

- Inserire nella tabella aerei un numero di tuple pari a 15.*

Soluzione

```
INSERT INTO aerei(id, produttore, modello, dataimm, numposti)
VALUES ('superjet', 'boeing', '747', '2000-10-10', 350);
```

```
INSERT INTO aerei(id, produttore, modello)
VALUES ('minijet', 'MDD', 'Super80');
```

```
SELECT * FROM aerei;
```

id	produttore	modello	dataimm	numposti
minijet	MDD	Super80	NULL	NULL
superjet	boeing	747	2000-10-10	350

2 rows in set (0.00 sec)

Query sulle tabelle

- Sintassi:

```
SELECT <what_to_select>
FROM <which_table>
[WHERE <conditions_to_satisfy>]
[LIMIT {[<offset>],} <row_count> | <row_count> OFFSET <offset>]}
```

- Esempi:

```
SELECT * FROM aerei LIMIT 10;
```

-- seleziona le prime dieci tuple di aerei

```
SELECT * FROM aerei LIMIT 5, 6;
```

-- seleziona le successive 6 tuple a partire dalla 5

```
SELECT * FROM aerei WHERE modello = '747';
```

-- seleziona le tuple per cui è verificata la condizione

```
SELECT * FROM aerei WHERE modello = '747' LIMIT 3;
```

-- seleziona le prime 3 tuple per cui è verificata la condizione

Modificare i dati di una tabella

- Sintassi:

```
UPDATE <table_name>
SET      col_name1=expr1 [, col_name2=expr2, ...]
[WHERE <where_definition>]
[LIMIT <row_count>]
```

- Esempi:

```
UPDATE aerei SET dataimm= '2010-02-31'
WHERE produttore = 'MDD';
```

```
UPDATE aerei SET dataimm= '2010-02-27'
WHERE produttore = 'MDD' LIMIT 2;
-- aggiorna solo le prime 2 due tuple per il produttore 'MDD'
```

Cancellare dati da una tabella

- Sintassi:

```
DELETE FROM <table_name>
[WHERE      <where_definition>]
[LIMIT      <row_count>]
```

- Esempi:

```
DELETE FROM aerei LIMIT 5;
-- cancella le prime 5 righe nella tabella
```

```
DELETE FROM aerei WHERE modello= '747';
-- cancella le righe con modello uguale a '747'
```

```
DELETE FROM aerei WHERE modello= '747' LIMIT 2;
-- cancella le prime 2 righe con modello uguale a '747'
```

Esempio di – ON DELETE CASCADE

```
CREATE TABLE parent (
    id      INT      NOT NULL,
    PRIMARY KEY (id)
) ENGINE=INNODB;
```

```
CREATE TABLE child (
    id          INT,
    parent_id   INT          DEFAULT 1,
    birth       TIMESTAMP    DEFAULT current_timestamp,
    PRIMARY KEY(id),
    FOREIGN KEY (parent_id) REFERENCES parent(id) ON DELETE CASCADE
) ENGINE=INNODB;
```

```
INSERT INTO parent VALUES (1);
```

```
INSERT INTO parent VALUES (3);
```

```
INSERT INTO child(id) VALUES (9);      -- al parent_id viene assegnato il valore 1 di default
```

```
INSERT INTO child(id, parent_id) VALUES (10,1);
```

```
INSERT INTO child (id, parent_id) VALUES (11,3);
```

```
INSERT INTO child (id, parent_id) VALUES (12,3);
```

```
...
```

```
DELETE FROM parent WHERE id=3;
```

-- alternativa:
... ON DELETE SET NULL

```
SELECT * FROM child;
```

id	parent_id	birth
9	1	2020-12-04 17:19:04
10	1	2020-12-04 17:19:04

2 rows in set (0.00 sec)

Esempio di – ON DELETE SET NULL

```
CREATE TABLE parent (
    id      INT      NOT NULL,
    PRIMARY KEY (id)
) ENGINE=INNODB;
```

```
CREATE TABLE child (
    id          INT,
    parent_id   INT          DEFAULT 1,
    birth       TIMESTAMP    DEFAULT current_timestamp,
    PRIMARY KEY(id),
    FOREIGN KEY (parent_id) REFERENCES parent(id) ON DELETE SET NULL
) ENGINE=INNODB;
```

```
INSERT INTO parent VALUES (1);
INSERT INTO parent VALUES (3);
INSERT INTO child(id) VALUES (9);
INSERT INTO child(id, parent_id) VALUES (10,1);
INSERT INTO child (id, parent_id) VALUES (11,3);
INSERT INTO child (id, parent_id) VALUES (12,3);
...
DELETE FROM parent WHERE id=3;
```

```
SELECT * FROM child;
```

id	parent_id	birth
9	1	2020-12-11 17:12:56
10	1	2020-12-11 17:13:18
11	NULL	2020-12-11 17:13:18
12	NULL	2020-12-11 17:13:18

4 rows in set (0.00 sec)

ALTER table

- **ALTER TABLE <nome_tabella> DROP PRIMARY KEY**
-- dopo questo comando la tabella non ha più nessuna chiave primaria
- **ALTER TABLE <nome_tabella> ADD PRIMARY KEY (<nome_campo>)**
-- aggiunge una chiave primaria alla tabella
- **ALTER TABLE <nome_tabella> DROP [COLUMN] <nome_colonna>**
-- elimina la colonna “<nome_colonna>”
- **ALTER TABLE <nome_tabella> CHANGE <vecchio_nome_colonna> <nuovo_nome_colonna> <nuovo tipo>**
-- cambia nome e tipo di una colonna
- **ALTER TABLE <nome_tabella> ADD [COLUMN] <nome_colonna> <tipo>**
-- aggiunge un nuovo campo alla tabella
- **ALTER TABLE <nome_tabella> RENAME <nuovo_nome_tabella>**
-- cambia nome a una tabella

ALTER table - esempi

- **ALTER TABLE child CHANGE birth birthday DATE;**
-- cambia la colonna di nome birth in birthday con tipo associato DATE
- **ALTER TABLE child ADD COLUMN age VARCHAR(3);**
-- aggiunge una colonna di nome age di tipo VARCHAR(3)
- **ALTER TABLE child DROP age;**
-- cancella la colonna age

Ricreare una tabella

- **SHOW CREATE TABLE <nome_tabella>;**
-- visualizza il comando necessario per ricreare una tabella già esistente nel database
- Con **SHOW CREATE TABLE <nome_tabella>** si può:
 1. rivisualizzare il comando usato per creare la tabella “aerei”,
 2. copiarlo su un editor di testi,
 3. eliminare la "spazzatura di contorno" (ad esempio, i simboli | - + usati per tracciare le linee nel risultato),
 4. e ricopiarla sul monitor MySQL.

SHOW CREATE TABLE aerei;

```
+-----+  
| Table | Create Table |  
+-----+  
| aerei | CREATE TABLE `aerei` (  
    `id` char(20) NOT NULL,  
    `produttore` char(20) NOT NULL,  
    `modello` char(10) NOT NULL,  
    `dataimm` date DEFAULT NULL,  
    `numposti` int(11) DEFAULT NULL,  
    PRIMARY KEY (`id`)  
 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |  
+-----+  
1 row in set (0.00 sec)
```

Utilizzo degli script

- *Più in generale, se ho un database su un server e voglio copiarlo su un'altra macchina, è possibile farlo senza digitare a mano tutti i comandi usati per la creazione delle tabelle e l'inserimento dei dati?*
- Uno dei modi più semplici per copiare un database da un server a un altro è avere uno script che consiste di **comandi SQL** che creano il database, le tabelle ed eventualmente inseriscono i dati necessari.
(Ad esempio, lo script [airdb.sql](#) esegue i comandi necessari a ricreare il database airdb).
- Uno script è essenzialmente un file di testo composto da vari comandi SQL.

Esecuzione di script in SQL tramite source

- Per eseguire uno script, dal prompt di MySQL eseguire il comando:

SOURCE <nome_file>;

- **ATTENZIONE!!!** Uno script SQL per poter essere eseguito deve essere un file di testo. Questo vuol dire che deve essere privo di qualsiasi tipo di formattazione. Ad esempio un testo in HTML non può essere eseguito. La stessa cosa vale per i file di Microsoft Word (.doc), OpenOffice (.sxw), AbiWord (.abw) e altri.

Importazione dei dati

- Vogliamo adesso aggiungere nuovi elementi. Supponiamo di avere queste informazioni in un file datiaerei.sql.
- Notare che nel file ogni campo è separato da un altro da una **tabulazione** (*attenzione una tabulazione, non uno spazio*), e ogni riga rappresenta una nuova istanza.
- Il comando che utilizziamo per inserire questi dati è:

```
LOAD DATA LOCAL INFILE 'datiaerei.sql' INTO TABLE aerei  
(id, produttore, modello, dataimm, numposti);
```

- È il comando per caricare i dati da un file. Nel comando specifichiamo:
 - il nome di file: **datiaerei.sql**
 - la tabella dove inserire i dati: **aerei**
 - le colonne che intendiamo inserire nella tabella **(id, produttore, modello, dataimm, numposti)**

SET GLOBAL local_infile = true; -- permette di load data in locale

SHOW GLOBAL VARIABLES LIKE 'local_infile'; -- elenca le variabili impostate

e aggiungere il parametro **--local_infile=1** al comando **mysql**. -- mysql -u root -p --local_infile=1

Importazione dei dati (2)

- Ci sono altri parametri che posso essere specificati, dopo il nome della tabella e prima dell'elenco dei campi:
 - **FIELDS ENCLOSED BY '<carattere>'**: per indicare che i valori degli attributi nel file di testo sono racchiusi tra una coppia di caratteri uguali;
 - **FIELD TERMINATED BY '<carattere>'**: per indicare che i valori degli attributi sono separati tra loro dal carattere specificato, piuttosto che dalla tabulazione;
 - **IGNORE <number> LINES**: per ignorare le prime *<number>* righe.

Importazione dei dati (3)

- Supponiamo che il file datiaerei2.sql contenga queste informazioni

produttore,modello,data immatricolazione,numposti
xyz,"columbia shuttle",1999-6-4,5
boeing,B600,1920-2-23

- È possibile importare questi dati usando il comando:

```
LOAD DATA LOCAL INFILE 'datiaerei2.sql' INTO TABLE aerei
FIELDS TERMINATED BY ',' ENCLOSED BY "" IGNORE 1 LINES
(id,produttore,modello, dataimm, numposti);
```

- Notare che FIELDS non va ripetuto in FIELDS ENCLOSED BY poiché è già presente in FIELDS TERMINATED BY.

MySQL Workbench: Data import/export

MySQL Workbench

Local instance 3306

Administration Schemas Query 1 Administration - Data Export

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

Object Info Session

No object selected

Local instance 3306 Data Export

Advanced Options...

Tables to Export

Export	Schema	Objects
<input type="checkbox"/>	HMETHYL_Relational	account
<input type="checkbox"/>	airdb	branch
<input checked="" type="checkbox"/>	bank	business
<input type="checkbox"/>	comune_sicuro_censimento	customer
<input type="checkbox"/>	ofvd_access	department
<input type="checkbox"/>	photo	employee
<input type="checkbox"/>	recchello2010	individual

Refresh 11 tables selected Dump Structure and Data Select Views Select Tables Unselect All

Objects to Export

Dump Stored Procedures and Functions Dump Events Dump Triggers

Export Options

Export to Dump Project Folder /Users/michelerisi/dumps/Dump20201204 Export to Self-Contained File /Users/michelerisi/dumps/Dump20201204.sql

Each table will be exported into a separate file. This allows a selective restore, but may be slower.

All selected database objects will be exported into a single, self-contained file.

Create Dump in a Single Transaction (self-contained file only) Include Create Schema

Press [Start Export] to start... Start Export

SQL Editor Opened.

The Bank schema

- Type:

SOURCE [bank.sql](#); -- premere Invio

USE bank;

SHOW tables;

DESC customer;

Tables_in_bank
account
branch
business
customer
department
employee
individual
officer
product
product_type
transaction

Field	Type	Null	Key	Default	Extra
cust_id	int(10) unsigned	NO	PRI	NULL	auto_increment
fed_id	varchar(12)	NO		NULL	
cust_type_cd	enum('I','B')	NO		NULL	
address	varchar(30)	YES		NULL	
city	varchar(20)	YES		NULL	
state	varchar(20)	YES		NULL	
postal_code	varchar(10)	YES		NULL	

7 rows in set (0.00 sec)

Select

SELECT * FROM department;

SELECT dept_id, name FROM department;

SELECT name FROM department;

SELECT * FROM employee;

SELECT * FROM employee LIMIT 5;

SELECT * FROM employee LIMIT 5,6;

SELECT * FROM employee WHERE emp_id > 10 LIMIT 5;

**SELECT emp_id, 'ACTIVE', emp_id * 3.14159, UPPER(lname)
FROM employee;**

Column aliases

SELECT

emp_id,

'ACTIVE' AS status,

emp_id * 3.14159 AS empid_x_pi,

UPPER(lname) AS last_name_upper

FROM employee;

emp_id	status	empid_x_pi	last_name_upper
1	ACTIVE	3.14159	SMITH
2	ACTIVE	6.28318	BARKER
3	ACTIVE	9.42477	TYLER
4	ACTIVE	12.56636	HAWTHORNE
5	ACTIVE	15.70795	GOODING
6	ACTIVE	18.84954	FLEMING

...

18 rows in set (0.00 sec)

Removing duplicates

SELECT cust_id FROM account;

cust_id
1
1
1
2
2
3
...

24 rows in set (0.00 sec)

SELECT DISTINCT cust_id FROM account;

cust_id
1
2
3
...

13 rows in set (0.00 sec)

Subquery

```
SELECT e.emp_id, e.fname, e.lname FROM  
  (SELECT emp_id, fname, lname,  
         start_date, title FROM employee) AS e;
```

emp_id	fname	lname
1	Michael	Smith
2	Susan	Barker
3	Robert	Tyler
4	Susan	Hawthorne
5	John	Gooding
6	Helen	Fleming
7	Chris	Tucker
8	Sarah	Parker
9	Jane	Grossman
10	Paula	Roberts
11	Thomas	Ziegler
12	Samantha	Jameson
...		
18 rows in set (0.00 sec)		

Views

```
CREATE VIEW employee_vw AS
  SELECT emp_id, fname, lname,
  YEAR(start_date) start_year FROM employee;
```

Query OK, 0 rows affected (0.10 sec)

```
SELECT emp_id, start_year FROM employee_vw;
```

emp_id	start_year
1	2001
2	2002
3	2000
4	2002

...

18 rows in set (0.00 sec)

/* provare con DAY e MONTH */

Table links (join)

```
SELECT employee.emp_id, employee.fname,  
       employee.lname, department.name  
          dept_name  
  
FROM employee INNER JOIN department  
      ON employee.dept_id = department.dept_id;
```

emp_id	fname	lname	dept_name
1	Michael	Smith	Administration
2	Susan	Barker	Administration
3	Robert	Tyler	Administration
4	Susan	Hawthorne	Operations
5	John	Gooding	Loans
6	Helen	Fleming	Operations
7	Chris	Tucker	Operations

...

18 rows in set (0.00 sec)

Where

```
SELECT emp_id, fname, lname, start_date, title  
FROM employee
```

WHERE title = 'Head Teller' AND start_date > '2002-01-01';

```
+-----+-----+-----+-----+  
| emp_id | fname | lname | start_date | title |  
+-----+-----+-----+-----+  
|      6 | Helen | Fleming | 2004-03-17 | Head Teller |  
|     10 | Paula | Roberts | 2002-07-27 | Head Teller |  
+-----+-----+-----+-----+  
2 rows in set (0.00 sec)
```

```
SELECT emp_id, fname, lname, start_date, title  
FROM employee
```

WHERE title = 'Head Teller' OR start_date > '2004-06-01';

```
+-----+-----+-----+-----+-----+  
| emp_id | fname | lname | start_date | title |  
+-----+-----+-----+-----+-----+  
|      6 | Helen | Fleming | 2004-03-17 | Head Teller |  
|      7 | Chris | Tucker | 2004-09-15 | Teller |  
|     10 | Paula | Roberts | 2002-07-27 | Head Teller |  
|     13 | John  | Blake  | 2000-05-11 | Head Teller |  
|     16 | Theresa | Markham | 2001-03-15 | Head Teller |  
+-----+-----+-----+-----+-----+  
5 rows in set (0.00 sec)
```

Group by and Having

```
SELECT d.name, COUNT(e.emp_id) AS num_employees  
FROM department d INNER JOIN employee e  
    ON d.dept_id = e.dept_id  
GROUP BY d.name  
HAVING count(e.emp_id) > 2;
```

```
+-----+-----+  
| name | num_employees |  
+-----+-----+  
| Operations | 14 |  
| Administration | 3 |  
+-----+-----+  
2 rows in set (0.00 sec)
```

```
... HAVING count(e.emp_id) > 3;  
+-----+-----+  
| name | num_employees |  
+-----+-----+  
| Operations | 14 |  
+-----+-----+  
1 rows in set (0.00 sec)
```

-- senza HAVING

```
+-----+-----+  
| name | num_employees |  
+-----+-----+  
| Operations | 14 |  
| Loans | 1 |  
| Administration | 3 |  
+-----+-----+  
3 rows in set (0.00 sec)
```

Order By

```
SELECT open_emp_id, product_cd FROM account;
```

```
SELECT open_emp_id, product_cd  
FROM account  
ORDER BY open_emp_id;
```

```
+-----+-----+  
| open_emp_id | product_cd |  
+-----+-----+  
          1 | CHK  
          1 | SAV  
          1 | MM  
          1 | CHK  
          1 | CD  
          1 | CHK  
          1 | MM  
          1 | CD  
         10 | CHK  
         10 | SAV  
...  
24 rows in set (0.00 sec)
```

```
SELECT open_emp_id, product_cd  
FROM account  
ORDER BY open_emp_id ASC , product_cd DESC;
```

```
+-----+-----+  
| open_emp_id | product_cd |  
+-----+-----+  
          1 | SAV  
          1 | MM  
          1 | MM  
          1 | CHK  
          1 | CHK  
          1 | CHK  
          1 | CD  
          1 | CD  
         10 | SAV  
         10 | SAV  
...  
24 rows in set (0.00 sec)
```

Sorting via Expressions/Placeholders

```
SELECT cust_id, cust_type_cd, city, state, fed_id  
FROM customer  
ORDER BY RIGHT(fed_id, 3);
```

```
SELECT cust_id, cust_type_cd, city, state, fed_id  
FROM customer  
ORDER BY LEFT(fed_id, 3);
```

```
SELECT emp_id, title, start_date, fname, lname  
FROM employee  
ORDER BY 2, 5; -- indice della colonna
```

```
SELECT emp_id, fname, lname  
FROM employee WHERE LEFT(lname, 1) = 'T';
```

emp_id	fname	lname
3	Robert	Tyler
7	Chris	Tucker
18	Rick	Tulman

3 rows in set (0.00 sec)

Inequality conditions

```
SELECT pt.name AS product_type, p.name AS product
FROM product AS p INNER JOIN product_type AS pt
    ON p.product_type_cd = pt.product_type_cd
WHERE pt.name <> 'Customer Accounts';
```

product_type	product
Individual and Business Loans	auto loan
Individual and Business Loans	business line of credit
Individual and Business Loans	home mortgage
Individual and Business Loans	small business loan

4 rows in set (0.00 sec)

si può usare anche ‘!=’

Range conditions

```
SELECT emp_id, fname, lname, start_date  
FROM employee  
WHERE start_date < '2003-04-01';
```

```
SELECT emp_id, fname, lname, start_date  
FROM employee  
WHERE start_date < '2007-01-01' AND start_date >= '2003-02-01';
```

```
SELECT emp_id, fname, lname, start_date  
FROM employee  
WHERE start_date BETWEEN '2003-04-01' AND '2007-01-01'  
ORDER BY start_date;
```

emp_id	fname	lname	start_date
15	Frank	Portman	2003-04-01
5	John	Gooding	2003-11-14
6	Helen	Fleming	2004-03-17
7	Chris	Tucker	2004-09-15

4 rows in set (0.00 sec)

Range conditions (2)

```
SELECT emp_id, fname, lname, start_date  
FROM employee  
WHERE start_date BETWEEN '2007-01-01' AND '2002-04-01';  
Empty set (0.00 sec)
```

```
SELECT account_id, product_cd, cust_id, avail_balance  
FROM account  
WHERE avail_balance BETWEEN 3000 AND 5000;
```

account_id	product_cd	cust_id	avail_balance
3	CD	1	3000.00
17	CD	7	5000.00
18	CHK	8	3487.19

3 rows in set (0.00 sec)

String ranges

```
SELECT cust_id, fed_id  
FROM customer  
WHERE cust_type_cd = 'I' AND  
fed_id BETWEEN '500-00-0000' AND '999-99-9999'; -- lessicografico
```

cust_id	fed_id
5	555-55-5555
6	666-66-6666
7	777-77-7777
8	888-88-8888
9	999-99-9999

5 rows in set (0.00 sec)

Using subqueries

```
SELECT account_id, product_cd, cust_id, avail_balance  
FROM account  
WHERE product_cd IN  
    (SELECT product_cd  
     FROM product  
     WHERE product_type_cd = 'ACCOUNT');
```

product_cd
CD
CHK
MM
SAV

account_id	product_cd	cust_id	avail_balance
3	CD	1	3000.00
15	CD	6	10000.00
17	CD	7	5000.00
23	CD	9	1500.00
1	CHK	1	1057.75
4	CHK	2	2258.02

...

21 rows in set (0.00 sec)

Using subqueries

```
SELECT account_id, product_cd, cust_id, avail_balance  
FROM account  
WHERE product_cd NOT IN ('CHK','SAV','CD','MM');
```

account_id	product_cd	cust_id	avail_balance
25	BUS	10	0.00
27	BUS	11	9345.55
29	SBL	13	50000.00

3 rows in set (0.00 sec)

Matching conditions

```
SELECT lname FROM employee  
WHERE lname LIKE '_a%e%';
```

lname
Barker
Hawthorne
Parker
Jameson

4 rows in set (0.00 sec)

```
SELECT cust_id, fed_id FROM customer  
WHERE fed_id LIKE '___-__-___';
```

cust_id	fed_id
1	111-11-1111
2	222-22-2222
3	333-33-3333

...

9 rows in set (0.00 sec)

```
SELECT emp_id, fname, lname, superior_emp_id  
FROM employee
```

```
WHERE superior_emp_id IS NULL;
```

emp_id	fname	lname	superior_emp_id
1	Michael	Smith	NULL

1 row in set (0.00 sec)

Join e Cross Join

```
SELECT e.fname, e.lname, d.name  
FROM employee AS e CROSS JOIN department AS d;
```

fname	lname	name
Michael	Smith	Operations
Michael	Smith	Loans
Michael	Smith	Administration
Susan	Barker	Operations
Susan	Barker	Loans
Susan	Barker	Administration
Robert	Tyler	Operations
Robert	Tyler	Loans
Robert	Tyler	Administration
Susan	Hawthorne	Operations
Susan	Hawthorne	Loans
Susan	Hawthorne	Administration
John	Gooding	Operations
John	Gooding	Loans
John	Gooding	Administration
...		
54 rows in set (0.00 sec)		

esegue il prodotto cartesiano senza specificare la condizione;
si può usare anche solo la clausula JOIN

Inner Join

```
SELECT e.fname, e.lname, d.name  
FROM employee AS e JOIN department AS d  
ON e.dept_id = d.dept_id;
```

```
SELECT e.fname, e.lname, d.name  
FROM employee AS e INNER JOIN department AS d  
USING (dept_id);
```

```
SELECT a.account_id, c.fed_id, e.fname, e.lname  
FROM account AS a INNER JOIN customer AS c  
ON a.cust_id = c.cust_id  
INNER JOIN employee e  
ON a.open_emp_id = e.emp_id  
WHERE c.cust_type_cd = 'B'; -- inner join multiple
```

Left vs Right Outer Join/Natural Join

```
SELECT c.cust_id, b.name
```

```
FROM customer c LEFT OUTER JOIN business b  
ON c.cust_id = b.cust_id;
```

cust_id	name
1	NULL
2	NULL
3	NULL
4	NULL
5	NULL
6	NULL
7	NULL
8	NULL
9	NULL
10	Chilton Engineering
11	Northeast Cooling Inc.
12	Superior Auto Body
13	AAA Insurance Inc.

13 rows in set (0.00 sec)

```
SELECT c.cust_id, b.name
```

```
FROM customer c RIGHT OUTER JOIN business b  
ON c.cust_id = b.cust_id;
```

```
SELECT a.account_id, a.cust_id, c.cust_type_cd, c.fed_id  
FROM account a NATURAL JOIN customer c; -- cust_id in comune
```

Inner Joins (2)

```
SELECT e1.fname, e1.lname, 'VS' AS vs, e2.fname, e2.lname  
FROM employee AS e1 INNER JOIN employee AS e2  
    ON e1.emp_id != e2.emp_id  
WHERE e1.title = 'Teller' AND e2.title = 'Teller';
```

fname	lname	vs	fname	lname
Sarah	Parker	VS	Chris	Tucker
Jane	Grossman	VS	Chris	Tucker
Thomas	Ziegler	VS	Chris	Tucker
Samantha	Jameson	VS	Chris	Tucker
Cindy	Mason	VS	Chris	Tucker
Frank	Portman	VS	Chris	Tucker
Beth	Fowler	VS	Chris	Tucker
Rick	Tulman	VS	Chris	Tucker
Chris	Tucker	VS	Sarah	Parker
Jane	Grossman	VS	Sarah	Parker
Thomas	Ziegler	VS	Sarah	Parker
...				
72 rows in set (0.00 sec)				

Set operators

```
SELECT 'IND' type_cd, cust_id, lname AS name FROM individual  
UNION ALL
```

```
SELECT 'BUS' type_cd, cust_id, name FROM business;
```

type_cd	cust_id	name
IND	1	Hadley
IND	2	Tingley
IND	3	Tucker
IND	4	Hayward
IND	5	Frasier
IND	6	Spencer
IND	7	Young
IND	8	Blake
IND	9	Farley
BUS	10	Chilton Engineering
BUS	11	Northeast Cooling Inc.
BUS	12	Superior Auto Body
BUS	13	AAA Insurance Inc.

13 rows in set (0.00 sec)

-- in MySQL 8 INTERSECT
e EXCEPT non sono supportati

Create table with strings

```
CREATE TABLE string_tbl (
    char_fld CHAR(30),
    vchar_fld VARCHAR(30),
    text_fld TEXT
);
```

```
INSERT INTO string_tbl (char_fld, vchar_fld, text_fld)
VALUES ('This is char data', 'This is varchar data', 'This is text data');
```

```
SELECT * FROM string_tbl;
```

char_fld	vchar_fld	text_fld
This is char data	This is varchar data	This is text data

1 row in set (0.00 sec)

Including single quotes

```
UPDATE string_tbl
```

```
SET text_fld = 'This string doesn't work'; -- non funziona
```

```
UPDATE string_tbl
```

```
SET text_fld = 'This string didn"t work, but it does now'; -- funziona
```

```
UPDATE string_tbl
```

```
SET text_fld = 'This string didn\'t work, but it does now'; -- funziona
```

```
SELECT text_fld FROM string_tbl;
```

```
SELECT QUOTE(text_fld) FROM string_tbl;
```

```
+-----+  
| QUOTE(text_fld) |  
+-----+  
| 'This string didn\'t work, but it does now' |  
+-----+  
1 row in set (0.00 sec)
```

String functions

```
SELECT CONCAT('danke sch', CHAR(148), 'n');
```

```
+-----+  
| CONCAT('danke sch', CHAR(148), 'n') |  
+-----+  
| danke schön |  
+-----+  
1 row in set (0.00 sec)
```

```
SELECT LENGTH(char_fld) char_length,  
      LENGTH(vchar_fld) varchar_length,  
      LENGTH(text_fld) text_length
```

```
FROM string_tbl;
```

```
+-----+-----+-----+  
| char_length | varchar_length | text_length |  
+-----+-----+-----+  
|          17 |             20 |           40 |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

```
SELECT POSITION('an' IN fname), fname FROM employee;
```

```
+-----+-----+  
| POSITION('an' IN fname) | fname   |  
+-----+-----+  
|                  0 | Michael |  
|                  4 | Susan   |  
...  
18 rows in set (0.00 sec)
```

Performing Arithmetic Functions

SELECT MOD(10,4); -- 2

SELECT POW(2,8); -- 256

SELECT CEIL(72.445),
FLOOR(73.445); -- 73 73

SELECT ROUND(72.4999),
ROUND(72.5),
ROUND(72.5001); -- 72 73 73

SELECT ROUND(72.0909, 1),
ROUND(72.0909, 2),
ROUND(72.0909, 3); -- 72.1 72.09 72.091

Functions for generating dates

```
SELECT STR_TO_DATE('December 05, 2019', '%M %d, %Y');
```

```
+-----+
| STR_TO_DATE('December 05, 2019', '%M %d, %Y') |
+-----+
| 2019-12-05
+-----+
1 row in set (0.00 sec)
```

Format component	Description
%M	Month name (January to December)
%m	Month numeric (01 to 12)
%d	Day numeric (01 to 31)
%j	Day of year (001 to 366)
%W	Weekday name (Sunday to Saturday)
%Y	Year, four-digit numeric
%y	Year, two-digit numeric
%H	Hour (00 to 23)
%h	Hour (01 to 12)
%i	Minutes (00 to 59)
%s	Seconds (00 to 59)
%f	Microseconds (000000 to 999999)
%p	A.M. or P.M.

Temporal functions that return dates

```
SELECT DATE_ADD(CURRENT_DATE(), INTERVAL 5 DAY);
```

```
+-----+  
| DATE_ADD(CURRENT_DATE(), INTERVAL 5 DAY) |  
+-----+  
| 2020-12-17 |  
+-----+  
1 row in set (0.01 sec)
```

Interval name	Description
Second	Number of seconds
Minute	Number of minutes
Hour	Number of hours
Day	Number of days
Month	Number of months
Year	Number of years
Minute_second	Number of minutes and seconds, separated by ":"
Hour_second	Number of hours, minutes, and seconds, separated by ":"
Year_month	Number of years and months, separated by "-"

```
SELECT LAST_DAY('2020-02-05');
```

```
+-----+  
| LAST_DAY('2020-02-05') |  
+-----+  
| 2020-02-29 |  
+-----+  
1 row in set (0.00 sec)
```

Temporal functions

```
SELECT DAYNAME('2019-12-05');
```

```
+-----+  
| DAYNAME('2019-12-05') |  
+-----+  
| Thursday |  
+-----+  
1 row in set (0.01 sec)
```

```
SELECT DATEDIFF('2019-09-03', '2019-06-24');
```

```
+-----+  
| DATEDIFF('2019-09-03', '2019-06-24') |  
+-----+  
| 71 |  
+-----+  
1 row in set (0.00 sec)
```

Aggregate functions

```
SELECT MAX(avail_balance) AS max_balance,  
       MIN(avail_balance) AS min_balance,  
       AVG(avail_balance) AS avg_balance,  
       SUM(avail_balance) AS tot_balance,  
       COUNT(*) AS num_accounts  
FROM account  
WHERE product_cd = 'CHK';
```

max_balance	min_balance	avg_balance	tot_balance	num_accounts
38552.05	122.37	7300.800985	73008.01	10

1 row in set (0.01 sec)

Aggregate functions (2)

```
SELECT product_cd,  
       MAX(avail_balance) AS max_balance,  
       MIN(avail_balance) AS min_balance,  
       AVG(avail_balance) AS avg_balance,  
       SUM(avail_balance) AS tot_balance,  
       COUNT(*) AS num_accounts  
  FROM account  
 GROUP BY product_cd;
```

product_cd	max_balance	min_balance	avg_balance	tot_balance	num_accounts
BUS	9345.55	0.00	4672.774902	9345.55	2
CD	10000.00	1500.00	4875.000000	19500.00	4
CHK	38552.05	122.37	7300.800985	73008.01	10
MM	9345.55	2212.50	5681.713216	17045.14	3
SAV	767.77	200.00	463.940002	1855.76	4
SBL	50000.00	50000.00	50000.000000	50000.00	1

6 rows in set (0.00 sec)

Grouping via expressions

SELECT

EXTRACT(YEAR FROM start_date) AS year,

COUNT(*) AS how_many

FROM employee

GROUP BY EXTRACT(YEAR FROM start_date);

year	how_many
2001	2
2002	8
2000	3
2003	3
2004	2

5 rows in set (0.00 sec)

Save output

```
SELECT emp_id, fname, lname, start_date  
      INTO OUTFILE 'emp_list.txt'  
     FROM employee;
```

-- utilizzare il parametro: **--secure-file-priv=dir_name**

Aggiungere o creare indici

ALTER TABLE department ADD INDEX dept_name_idx (name);
SHOW INDEX FROM department \G

```
***** 1. row *****
    Table: department
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 1
 Column_name: dept_id
  Collation: A
Cardinality: 2
  Sub_part: NULL
    Packed: NULL
      Null:
Index_type: BTREE
  Comment:
Index_comment:
  Visible: YES
 Expression: NULL
***** 2. row *****
    Table: department
  Non_unique: 1
    Key_name: dept_name_idx
Seq_in_index: 1
 Column_name: name
  Collation: A
Cardinality: 2
  Sub_part: NULL
    Packed: NULL
      Null:
Index_type: BTREE
  Comment:
Index_comment:
  Visible: YES
 Expression: NULL
```

DESCRIBE department;

Field	Type	Null	Key	Default	Extra
dept_id	smallint(5) unsigned	NO	PRI	NULL	auto_increment
name	varchar(20)	NO	MUL	NULL	

2 rows in set (0.01 sec)

Aggiungere o creare indici (2)

CREATE INDEX dept_name_idx ON department (name);

- Indice su più attributi:

ALTER TABLE employee ADD INDEX emp_names_idx (lname, fname);

- Esempio:

ALTER TABLE account ADD INDEX acc_bal_idx (cust_id, avail_balance);

```
SELECT cust_id, SUM(avail_balance) tot_bal  
FROM account WHERE cust_id IN (1, 5, 9, 11)  
GROUP BY cust_id;
```

cust_id	tot_bal
1	4557.75
5	2237.97
9	10971.22
11	9345.55

4 rows in set (0.00 sec)

Trigger

- I **trigger** sono un meccanismo attraverso il quale è possibile automatizzare le operazioni INSERT, UPDATE o DELETE al verificarsi di eventi riguardanti i dati.
- Quando definiamo un trigger in MySQL dobbiamo stabilire se questo debba innescarsi prima o dopo un certo evento:
 - BEFORE INSERT **tempistica evento**
 - BEFORE UPDATE
 - BEFORE DELETE
 - AFTER INSERT
 - AFTER UPDATE
 - AFTER DELETE

Sintassi dei Trigger

- La sintassi di base per la creazione di un trigger è la seguente:

```
CREATE TRIGGER nome_trigger
    tempistica evento ON nome_tabella
    FOR EACH ROW
```

```
BEGIN
```

```
...
```

```
END;
```

```
SHOW TRIGGERS;
```

-- elenca i trigger

```
DROP TRIGGER nome_trigger;
```

-- cancella un trigger

```
DROP TRIGGER IF EXISTS nome_trigger;
```

-- cancella se esiste

Esempio

- Creiamo un trigger che viene invocato subito prima (*BEFORE*) dell'aggiornamento (*UPDATE*) della tabella "clienti":

DELIMITER \$\$;

```
CREATE TRIGGER log_upd_clienti
    BEFORE UPDATE ON clienti
    FOR EACH ROW
```

BEGIN

```
    INSERT INTO clienti_log SET
        vecchio_nome = OLD.nome,
        nuovo_nome = NEW.nome,
        vecchia_email = OLD.email,
        nuova_email = NEW.email,
        modificato = NOW();
```

END \$\$

Perché usare DELIMITER?

Una trigger può contenere più istruzioni separate da punto e virgola (;

vecchio_nome, nuovo_nome,
vecchia_email, nuova_email
sono attributi della tabella clienti_log

Utilizziamo le keyword **OLD** e **NEW** per far riferimento, rispettivamente, al valore precedente alla modifica ed a quello, eventualmente differente, successivo all'operazione di UPDATE

DELIMITER ; -- ripristiniamo il delimitatore

OLD e NEW

- In merito alle due keyword **OLD** e **NEW** è importante ricordare che il loro utilizzo non è sempre possibile, precisamente:
 - INSERT: per le operazioni di inserimento dati è possibile utilizzare solo NEW in quanto non esiste alcun vecchio valore a cui far riferimento;
 - UPDATE: è possibile utilizzare sia OLD che NEW a seconda che si desideri far riferimento al valore di un campo, prima o dopo l'operazione di UPDATE;
 - DELETE: è possibile utilizzare solo OLD in quanto non si ha alcun nuovo valore.

Esempio

- Per i trigger scatenati mediante BEFORE sugli eventi INSERT e UPDATE, è possibile modificare il valore NEW di un dato campo prima che tale valore venga scritto nel database:

```
DELIMITER $$;
```

```
CREATE TRIGGER controllo_update_eta
    BEFORE UPDATE ON clienti
    FOR EACH ROW
BEGIN
    IF NEW.anni < 0 THEN
        SET NEW.anni = 0;
    ELSEIF NEW.anni > 120 THEN
        SET NEW.anni = 120;
    END IF;
END $$

DELIMITER ;
```

```
DELIMITER $$;
```

```
CREATE TRIGGER controllo_insert_eta
    BEFORE INSERT ON clienti
    FOR EACH ROW
BEGIN
    IF NEW.anni < 0 THEN
        SET NEW.anni = 0;
    ELSEIF NEW.anni > 120 THEN
        SET NEW.anni = 120;
    END IF;
END $$

DELIMITER ;
```

Esempio

DELIMITER \$\$

CREATE TRIGGER testref

BEFORE INSERT ON test1

FOR EACH ROW

BEGIN

INSERT INTO test2 SET a2 = NEW.a1;

DELETE FROM test3 WHERE a3 = NEW.a1;

UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;

END \$\$

DELIMITER ;

Gestire più Trigger associati allo stesso momento/evento

- È possibile che alla medesima tabella siano associati più trigger attivabili allo stesso momento e per il medesimo evento.
 - In tal caso i diversi trigger saranno eseguiti dal DBMS secondo l'ordine di creazione;
 - oppure è possibile specificare un ordine esplicito con le keyword **FOLLOW**S e **PRECEDES** seguite dal nome del trigger che si vuole seguire o anticipare.

```
CREATE TRIGGER controllo_username  
    BEFORE UPDATE ON clienti  
    FOR EACH ROW  
    PRECEDES controllo_eta
```

controllo_eta è un altro trigger creato sulla stessa tabella clienti

...

Procedure

- È possibile creare/eliminare delle stored procedure.

DROP PROCEDURE accountnum;

DELIMITER \$\$

CREATE PROCEDURE accountnum (**IN** producttype CHAR(3), **OUT** num INT)
BEGIN

 SELECT COUNT(*) INTO num FROM account
 WHERE product_cd = producttype;

END\$\$

DELIMITER ;

Esempio

CALL accountnum ('CHK', @num); -- CHK in account

SELECT @num;

```
+----+  
| @num |  
+----+  
|   10 |  
+----+  
1 row in set (0.00 sec)
```

CALL accountnum ('MM', @num); -- MM in account

SELECT @num;

```
+----+  
| @num |  
+----+  
|   3 |  
+----+  
1 row in set (0.00 sec)
```

È possibile memorizzare un valore in una variabile definita dall'utente in un'istruzione e farvi riferimento in seguito in un'altra istruzione. Ciò consente di passare valori da un'istruzione all'altra

Le variabili utente sono scritte come **@var_name**, dove il nome della variabile **var_name** è composto da caratteri alfanumerici

Un modo per impostare una variabile definita dall'utente consiste nell'emettere un'istruzione SET: **SET @var_name = expr**

Function

- È possibile definire e utilizzare funzioni.

```
DROP FUNCTION hello;
```

```
CREATE FUNCTION hello (s CHAR(20))
RETURNS CHAR(50) DETERMINISTIC
RETURN CONCAT('Hello, ', s, '!');
```

```
SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world! |
+-----+
1 row in set (0.00 sec)
```

Gestire le password con MD5

```
DROP TABLE IF EXISTS accounts;
```

```
CREATE TABLE accounts (
    id INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50), surname VARCHAR(50),
    email VARCHAR(100),
    password VARCHAR(32),
    role ENUM ('admin','user') DEFAULT 'user',
    PRIMARY KEY (id),
    UNIQUE KEY email (email)
);
```

```
INSERT INTO accounts ('name', 'surname', 'email', 'password', 'role')
```

```
VALUES ('Michele', 'Risi', 'mr@unisa.it', MD5('123'), 'admin');
```

```
INSERT INTO `accounts` ('name', 'surname', 'email', 'password', 'role')
```

```
VALUES ('Antonio', 'Giallo', 'ag@unisa.it', MD5('321'), 'user');
```

```
SELECT * FROM accounts;
```

id name surname email password role
1 Michele Risi mr@unisa.it 202cb962ac59075b964b07152d234b70 admin
2 Antonio Giallo ag@unisa.it caf1a3dfb505ffed0d024130f58c5cfa user

```
2 rows in set (0.01 sec)
```

```
SELECT * FROM accounts WHERE email = 'mr@unisa.it' AND password = MD5('123');
```

id name surname email password role
1 Michele Risi mr@unisa.it 202cb962ac59075b964b07152d234b70 admin

```
1 row in set (0.00 sec)
```

Il vincolo UNIQUE garantisce che tutti i valori in una colonna siano diversi