

Corso di fondamenti di informatica – quarta parte

Ingegneria industriale - polo di Frosinone

| | | | |
|--------------------------|--|------|----|
| <input type="checkbox"/> | La tecnologia JDBC..... | pag. | 2 |
| <input type="checkbox"/> | I driver per database..... | pag. | 3 |
| <input type="checkbox"/> | La connessione al database di Access..... | pag. | 4 |
| <input type="checkbox"/> | La connessione ai database..... | pag. | 9 |
| <input type="checkbox"/> | Connessione con manipolazione di dati..... | pag. | 11 |
| <input type="checkbox"/> | Le eccezioni..... | pag. | 12 |
| <input type="checkbox"/> | Operazione di inserimento in Java..... | pag. | 13 |
| <input type="checkbox"/> | La cancellazione | pag. | 17 |
| <input type="checkbox"/> | I Parametri nei comandi SQL..... | pag. | 19 |
| <input type="checkbox"/> | L'aggiornamento..... | pag. | 21 |
| <input type="checkbox"/> | Le interrogazioni – Proiezione..... | pag. | 23 |
| <input type="checkbox"/> | I metadati..... | pag. | 28 |
| <input type="checkbox"/> | L'accesso ai database in rete..... | pag. | 31 |
| <input type="checkbox"/> | Software MySQL Server..... | pag. | 36 |
| <input type="checkbox"/> | Operazioni con MySQL Workbench..... | pag. | 50 |
| <input type="checkbox"/> | L'accesso ai database in rete - varie applicazioni..... | pag. | 56 |
| <input type="checkbox"/> | MySQL – Principali comandi | pag. | 64 |
| <input type="checkbox"/> | MySQL– Administrator | pag. | 81 |
| <input type="checkbox"/> | Esercizi da implementare per connessione database in rete...pag. | | 83 |

La tecnologia JDBC

Il database o base di dati è un insieme di archivi integrati che formano una base di lavoro comune per utenti che svolgono applicazioni diverse. Tutti i database progettati secondo il modello relazionale, denominati database relazionali, organizzano i dati in tabelle, nelle quali a ciascuna tupla (riga) corrisponde un record formato da campi (ogni campo è una colonna della tabella).

I **DBMS** (Database Management System) sono software che consentono la gestione dei database: Microsoft Access ne è un esempio.

JDBC (Java Database Connectivity) è un'interfaccia di programmazione dell'applicazione, chiamate API, per il linguaggio di programmazione Java. JDBC mette a disposizione una libreria di classi Java per interfacciare l'accesso ai database che usano il linguaggio standard SQL. In tal modo si cerca di dare uniformità di accesso ai database relazionali.

I driver per database

Attraverso JDBC è possibile eseguire i comandi SQL denominati MML, MDL, MCL e QUERY. Le Api (Application Programming Interface) del software JDBC forniscono un'interfaccia analoga a quella offerta dall'API per i sistemi operativi moderni (windows, linux, ecc.).

Oracle tramite le JDK distribuisce L'API JDBC.

Cos'è il driver per database?

Il driver per database è un software che permette il trasferimento di dati presenti in un database. Quindi, se ad esempio, un programma in Java vuole accedere ai dati di un database di Access deve utilizzare il driver per quel database.

JDBC interagisce con i driver per i database creati con i più diffusi prodotti DBMS (Oracle, MySql, Informix, Sybase, SQLite, ecc.).

Connessione al database di Microsoft Access

Per sviluppare applicazioni Java su sistemi operativi Windows è possibile utilizzare anche i driver ODBC (Open Database Connectivity), anche se questa modalità è stata utilizzata nelle applicazioni più datate di java.

Dopo la versione Java SE Development Kit 8 si è deciso di utilizzare i driver forniti dalle aziende produttrici di DBMS.

Quali driver utilizzare per database Microsoft Access?

Per accedere a un database di Access si può usare **UCanAccess**, un driver open-source per JDBC e scaricabile al link:<https://sourceforge.net/projects/ucanaccess/>

Il download consente di scaricare un file zip contenente le librerie .jar utili per le connessione ai database di Microsoft Access.

Per chi usa NetBeans ver. 8.2, si consiglia di scaricare la versione di UCanAccess 3.0.7 al seguente link:

https://osdn.net/projects/sfnet_ucanaccess/downloads/archive/v3_archive/UCanAccess-3.0.7-bin.zip/

Nei prossimi esempi utilizzeremo applicazioni Java che si connettono con un database di Access tramite i driver UCanAccess 3.0.7.

Connessione al database di Microsoft Access

Nell'ambiente di sviluppo NetBeans le librerie **UCanAccess 3.0.7-bin.zip** (precedentemente scaricate) possono essere aggiunte al progetto Java in due modi diversi.

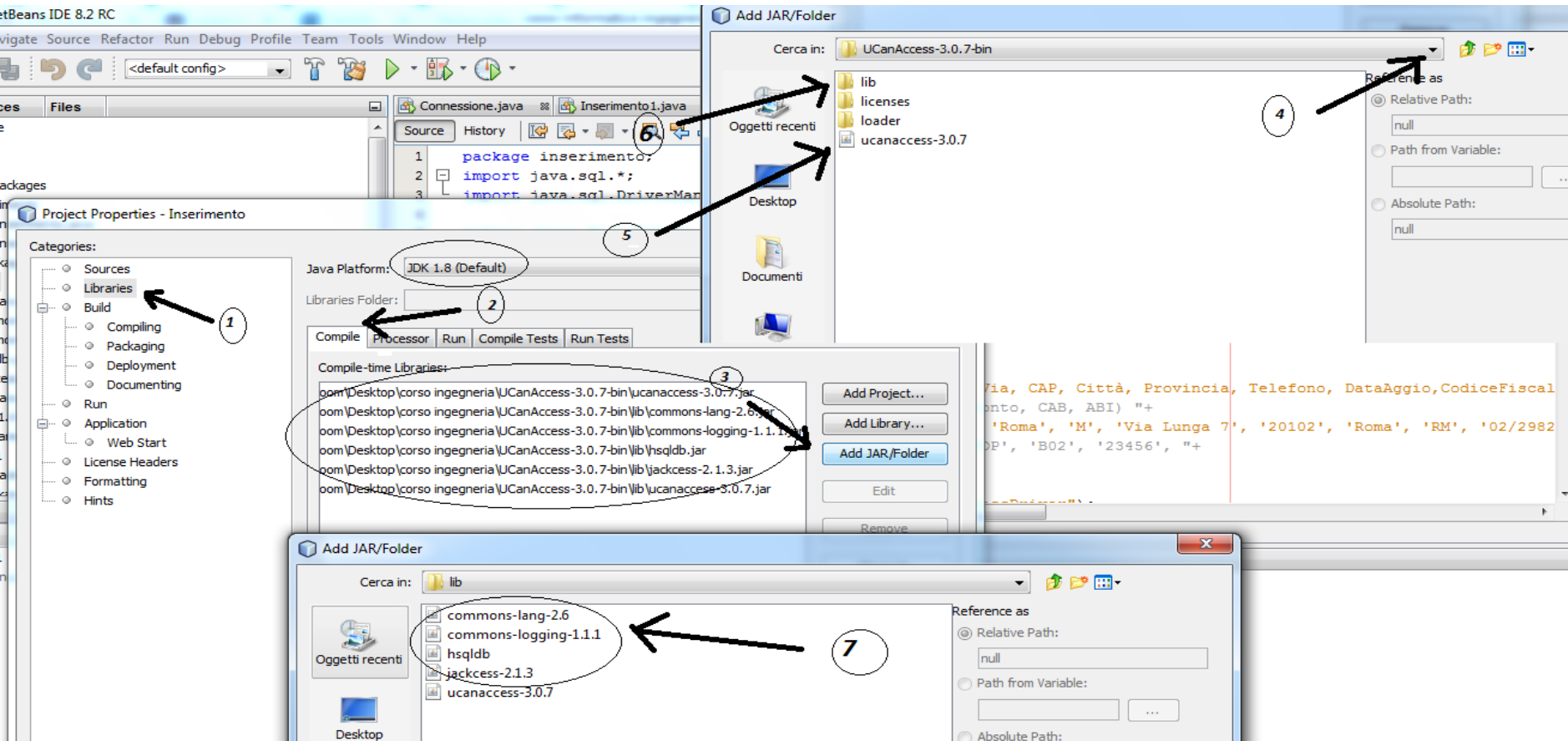
Riportiamo di seguito il **primo metodo**:

1. scompattare il file **UCanAccess 3.0.7-bin.zip** in una cartella;
2. aprire il progetto java e, tramite il tasto destro, selezionare la voce (Projects) sul nome del progetto nel riquadro a sinistra;
3. scegliere la voce *Properties*;
4. selezionare quindi *Libraries*;
5. selezionare la scheda *Compile* ed aggiungere, con il pulsante Add JAR/folder, i file .jar scompattati al punto (1). Si precisa che devono essere aggiunti nella sezione tutti i file contenuti nella cartella lib e il file ucanaccess-3.0.7.jar.

Riportiamo nella prossima pagina la stessa modalità spiegata sopra con l'aiuto di uno screenshot.

Connessione al database di Microsoft Access

Selezionare il progetto desiderato, successivamente cliccare con il tasto destro su **Libraries** e selezionare **Properties**, quindi seguire i passi numerati descritti nella figura.



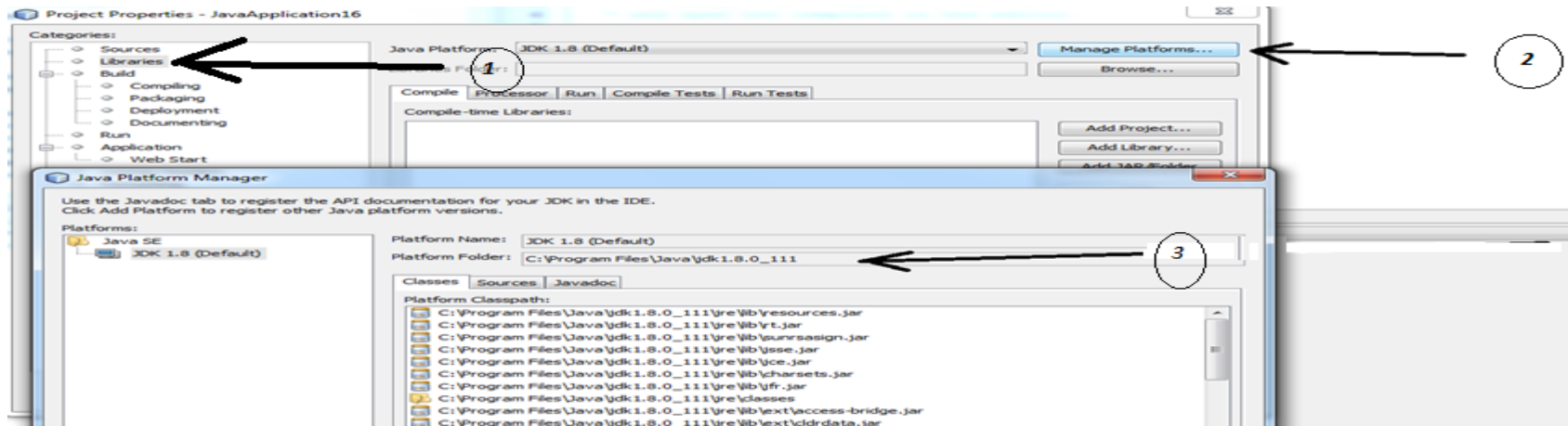
Oltre al metodo riportato, così come già anticipato, vi è un altro modo più efficace per inserire i file .jar nella cartella JDK in modalità permanente. Nella prossima pagina verrà descritto il metodo alternativo.

Connessione al database di Microsoft Access

Secondo metodo.

La modalità di seguito descritta si preferisce alla prima, in quanto assicura l'aggancio delle librerie **Ucanaccess** a tutti i progetti di Netbeans. Tale modalità consente di verificare la cartella di default di Java Platform all'interno del nostro computer. Quindi per individuare la cartella di default Java Platform, contenente i file .jar di java, si procede nel modo seguente:

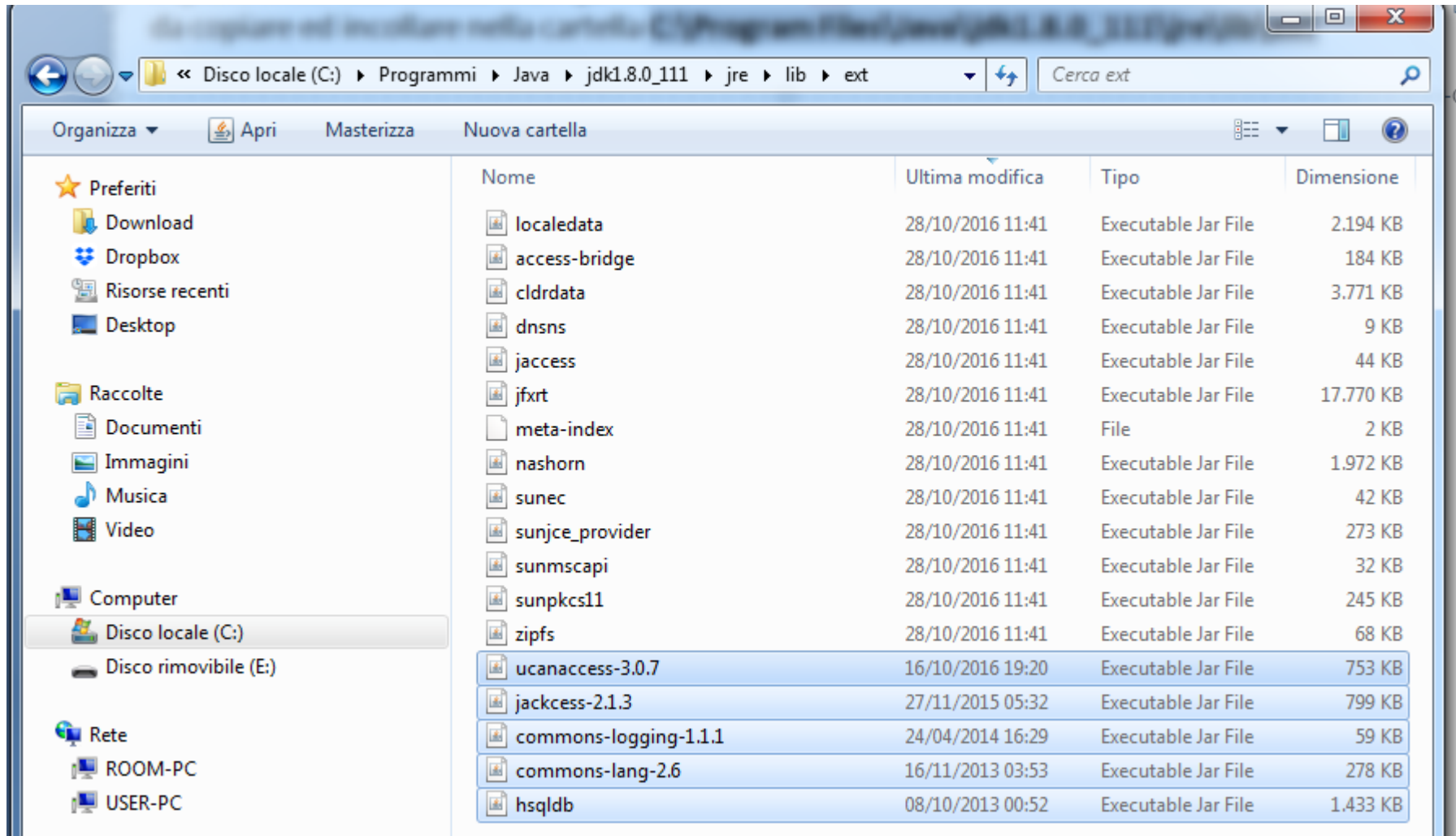
1. selezionare un progetto;
2. cliccare sulla voce *Libraries* con il tasto destro e selezionare *Properties*, così come indicato nella figura sotto (freccia 1).
3. selezionare *Manage Platform* (freccia 2) e successivamente copiare il percorso descritto nella voce Platform Folder (freccia 3) e incollarlo nella barra degli indirizzi di "risorsa del computer"; si precisa che la cartella da individuare è la cartella *ext*, all'interno della quale bisogna incollare i cinque file ucanaccess-3.0.7.jar scompattati e che verranno aggiunti agli altri file già esistenti. Il percorso completo in Windows 7 è **C:\Program Files\Java\jdk1.8.0_111\jre\lib\ext**



Nella prossima slide è illustrata la cartella ext ed i file da copiare

Connessione al database di Microsoft Access

Nuovi File incollati nella cartella C:\Program Files\Java\jdk1.8.0_111\jre\lib\ext



The screenshot shows a Windows Explorer window with the address bar set to C:\Program Files\Java\jdk1.8.0_111\jre\lib\ext. The left sidebar shows the navigation pane with 'Disco locale (C:)' selected. The main pane displays a list of files with columns for Name, Last Modified, Type, and Size.

| Nome | Ultima modifica | Tipo | Dimensione |
|-----------------------|------------------|---------------------|------------|
| localedata | 28/10/2016 11:41 | Executable Jar File | 2.194 KB |
| access-bridge | 28/10/2016 11:41 | Executable Jar File | 184 KB |
| cldrdata | 28/10/2016 11:41 | Executable Jar File | 3.771 KB |
| dnsns | 28/10/2016 11:41 | Executable Jar File | 9 KB |
| jaccess | 28/10/2016 11:41 | Executable Jar File | 44 KB |
| jfxt | 28/10/2016 11:41 | Executable Jar File | 17.770 KB |
| meta-index | 28/10/2016 11:41 | File | 2 KB |
| nashorn | 28/10/2016 11:41 | Executable Jar File | 1.972 KB |
| sunec | 28/10/2016 11:41 | Executable Jar File | 42 KB |
| sunjce_provider | 28/10/2016 11:41 | Executable Jar File | 273 KB |
| sunmscapi | 28/10/2016 11:41 | Executable Jar File | 32 KB |
| sunpkcs11 | 28/10/2016 11:41 | Executable Jar File | 245 KB |
| zipfs | 28/10/2016 11:41 | Executable Jar File | 68 KB |
| ucanaccess-3.0.7 | 16/10/2016 19:20 | Executable Jar File | 753 KB |
| jackcess-2.1.3 | 27/11/2015 05:32 | Executable Jar File | 799 KB |
| commons-logging-1.1.1 | 24/04/2014 16:29 | Executable Jar File | 59 KB |
| commons-lang-2.6 | 16/11/2013 03:53 | Executable Jar File | 278 KB |
| hsqldb | 08/10/2013 00:52 | Executable Jar File | 1.433 KB |

La connessione al database

E' possibile eseguire le operazioni di interrogazione e manipolazione su un **database locale** di Access usando applicazioni java. In alternativa si possono eseguire tali operazioni sul database di un **server di rete** usando le tecnologie Web.

Nel primo caso, le operazioni di interrogazione e manipolazione su un **database locale** possono essere specificate all'interno di una stringa così come sotto riportato:

```
String url = "jdbc:ucanaccess://C:\\Users\\room\\Desktop\\corso ingegneria\\Personale.accdb";
```

In tal caso il database **Personale.accdb** si trova nell'unità

C:\\Users\\room\\Desktop\\corso ingegneria

mentre l'istruzione **jdbc:ucanaccess** indica rispettivamente il protocollo e sottoprotocollo.

Nel secondo caso, per un database presente su un server di rete, si scarica MySQL Connector/J all'indirizzo <http://dev.mysql.com/downloads/connector/j/>

Se il database è presente in locale (server locale), viene definita la seguente stringa di connessione:

```
String url = "jdbc:mysql://localhost/Database";
```

La stringa, generalmente chiamata URL di JDBC, indica l'indirizzo internet dove si trova il database.

La connessione al database

I programmi che usano JDBC devono specificare l'istruzione

```
import java.sql.*;
```

Inoltre la classe dei driver è chiamata **DriverManager** e stabilisce la connessione tra un driver ed un database. Ogni progetto deve contenere all'inizio, tramite la classe DriverManager, l'istruzione che permette di creare la connessione a un particolare database, indicando il nome del database ed eventualmente username e password di autorizzazione.

Il Java per effettuare la connessione con il database utilizza il metodo getConnection della classe DriverManager. Più precisamente:

```
Connection conn =DriverManager.getConnection(url, "username","password");
```

Nel caso di connessioni a database che non hanno username e password, l'istruzione usata è:

```
Connection conn =DriverManager.getConnection(url, "", "");
```

Connessione con manipolazione di dati

Così come indicato precedentemente, la prima operazione da eseguire è la connessione al database. Successivamente si possono inviare al database i comandi SQL.

Prima di tutto utilizziamo una stringa che accolga l'istruzione SQL :

```
String sql="SELECT * FROM Personale WHERE Prov="MI";
```

Dopodichè carichiamo in modo esplicito il driver tramite il metodo **Class.forName**.

Quindi si predispone il comando (Statement) con il metodo **createStatement** che crea un'istanza della classe **Statement**.

L'istruzione seguente usa la connessione (oggetto conn) per costruire un comando SQL da inviare successivamente al database:

```
Statement st=conn.createStatement();
```

I comandi SQL sono poi attivati con i due principali metodi applicati allo statement:

- Il metodo **executeUpdate** che consente di restituire il numero di righe sottoposte all'operazione di manipolazione della tabella. Questo metodo viene utilizzato con i comandi di inserimento (*insert*), aggiornamento (*Update*) e cancellazione (*Delete*). Viene anche usato per i comandi SQL relativi alla creazione (*Create Table*), cancellazione (*Drop Table*) e modifica della struttura (*Alter Table*) delle tabelle nel database.
- Il metodo **executeQuery** è invece usato per le interrogazioni QUERY ed è in grado di restituire un insieme di risultati (*ResultSet*) riportando le righe della tabella che soddisfano la condizione descritta nella query.

Le eccezioni

Anche per le connessioni si utilizza una struttura `try...catch...finally` per gestire le eccezioni. Di seguito vediamo una struttura `try...catch.. finally`:

```
try
{
Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
conn=DriverManager.getConnection(url, "", "");
st=conn.createStatement();
st.executeUpdate(sql);
}
catch(SQLException ex)
{
System.err.println(ex.getMessage());
}
finally
{
if (st !=null)
{
try {st.close();} catch(SQLException ex){}
}
if (conn!=null)
{
try {conn.close();} catch(SQLException ex){}
}
}
}
```

Se JDBC incontra un errore di connessione con il database, allora viene generata l'eccezione **SQLException**. Il metodo **getMessage** descrive completamente l'eccezione che si verifica.

Se non ci sono errori nella gestione della connessione al database entra nel **try**, altrimenti va nel **catch** che stampa il tipo di errore. Il blocco **finally**, che verrà comunque sempre eseguito, sarà utilizzato per la corretta chiusura della connessione al database.

Anche se di solito si implementano istruzioni specifiche per il rilascio delle risorse, il blocco **finally** rilascia gli oggetti di tipo **Statement** e **Connection** con il metodo **close** e più precisamente:

- 1) con l'istruzione **if (st!=null)** viene verificato se la risorsa `st` è stata effettivamente creata e nel caso, tramite la `try{st.close()}`, la chiude; diversamente esegue l'eccezione(`SQLException ex`) tramite la `catch`;
- 2) Successivamente con l'istruzione **if (conn!=null)** verifica se la risorsa `conn` è stata effettivamente creata e nel caso, tramite la `try{conn.close()}`, la chiude; diversamente esegue l'eccezione (`SQLException ex`) tramite la `catch`.

Creazione di un database Access ed operazione di inserimento in Java

Nella prossima sessione di lavoro andremo a creare, tramite Microsoft Access, un database di nome **Personale**. Quindi si andrà ad implementare una tabella, di nome **Anagrafica**, la cui struttura è quella riportata nella foto sotto. Il database **Personale.accdb** è contenuto nella cartella **C:\\Users\\room\\Desktop\\corso ingegneria**

| Anagrafica | | |
|---------------|-----------|---------------------|
| Nome campo | Tipo dati | Dimensione |
| ID | Contatore | <i>Intero lungo</i> |
| Cognome | Testo | 40 |
| Nome | Testo | 40 |
| DataNascita | Data/ora | 8 |
| Luogo | Testo | 40 |
| Sesso | Testo | 1 |
| Via | Testo | 50 |
| CAP | Testo | 5 |
| Città | Testo | 30 |
| Provincia | Testo | 2 |
| Telefono | Testo | 12 |
| DataAggio | Data/ora | 8 |
| CodiceFiscale | Testo | 16 |
| Banca | Testo | 30 |
| Agenzia | Testo | 30 |
| NumeroConto | Testo | 15 |
| CAB | Testo | 6 |
| ABI | Testo | 6 |

Creazione di un database Access ed operazione di inserimento in Java

Dopo aver creato il database Personale e la tabella Anagrafica, così come indicato nel tracciato record della pagina precedente, si implementa il comando SQL costruito con una stringa di nome **sql**.

La stringa **sql** è formata da diversi parti concatenate con l'operatore **+**. Durante la digitazione della stringa, bisogna fare attenzione all'uso corretto dei doppi apici e dell'apice singolo: i doppi apici vengono utilizzati per separare le varie stringhe che formano la query, mentre gli apici singoli vengono utilizzati per delimitare i valori alfanumerici dei campi della tabella. E' riportato di seguito un esempio:

```
sql="INSERT INTO Anagrafica "+  
    " (Cognome, Nome, DataNascita, Luogo, Sesso, Via, CAP, Città, Provincia, Telefono,  
    DataAggio,CodiceFiscale, Banca, Agenzia, NumeroConto, CAB, ABI) "+  
    " VALUES ('Verdi', 'Franco', #09/03/2019#, 'Roma', 'M', 'Via Lunga 7', '00100', 'Roma', 'RM',  
    '02/2982828', #12/12/2019#, 'BNCBGL75B12R342Z', 'BIPOP', 'B02', '23456', '077785', '024211')";
```

Nelle prossime pagine vengono rappresentate le varie operazioni sul database:

- Operazione di inserimento (Inserimento.java)
- Operazione di cancellazione (Cancellazione.java)
- Operazione di aggiornamento (Aggiornamento.java)
- Interrogazioni (Selezione.java)
- Interrogazioni (Selezione_param.java)
- Visualizzazione della struttura dati – Metadati (Struttura.java)

Inserimento.java

```
package inserimento;
import java.sql.*;
public class Inserimento
{
    public static void main(String[] args)
    {
        Connection conn=null;
        Statement st=null;
        String sql;
        String url= "jdbc:ucanaccess://C:\\Users\\room\\Desktop\\corso ingegneria\\Personale.accdb";
        sql="INSERT INTO Anagrafica "+
            "(Cognome, Nome, DataNascita, Luogo, Sesso, Via, CAP, Città, Provincia, Telefono, DataAggio,CodiceFiscale, Banca,
            Agenzia, NumeroConto, CAB, ABI) "+
            "VALUES ('Verdi', 'Franco', #09/03/2019#, 'Roma', 'M', 'Via Lunga 7', '00100', 'Roma', 'RM', '02/2982828', #12/12/2019#,
            'BNCBGL75B12R342Z', 'BIPOP', 'B02', '23456', '077785', '024211')";
        try
        {
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
            conn=DriverManager.getConnection(url, "", "");
            st=conn.createStatement();
            st.executeUpdate(sql);
            System.out.println("RECORD REGISTRATO CORRETTAMENTE");
        }
        catch (ClassNotFoundException | SQLException e)
        {
            System.out.println("SQLException:");
            System.out.println(e.getMessage());
        }
        finally
        {
            if (st !=null)
            { //metodo close per l'oggetto st:rilascia le risorse
              //usate dal comando SQL
              try {st.close();} catch(SQLException e){}
            }
            if (conn!=null)
            { //metodo close per l'oggetto conn: interrompe
              //la connessione al database
              try {conn.close();} catch(SQLException e){}
            }
        }
    }
}
```

Inserimento.java

Si riporta di seguito lo screenshot dell'output di NetBeans, con relativa visualizzazione della nuova riga aggiunta alla tabella Anagrafica, tramite la query **INSERT INTO Anagrafica +.....**

```
32         try {st.close();} catch(SQLException e){
33     }
34     if (conn!=null)
35     { //metodo close per l'oggetto conn: interrompe
36         //la connessione al database
37         try {conn.close();} catch(SQLException e){
38     }
39     }
40 }}
41
42
```

inserimento.Inserimento > main > try > finally > if (st != null) >

Output - Inserimento (run) ✖

```
run:
RECORD REGISTRATO CORRETTAMENTE
BUILD SUCCESSFUL (total time: 2 seconds)
```

Strumenti tabella | Personale : Database (Access 2007) - Microsoft Access

Home | Crea | Dati esterni | Strumenti database | Foglio dati

Visualizza | Incolla | Calibri | 11 | Nuovo | Totali | A-Z | Selezione | Trova

Visualizzazioni | Appunti | G C S | Carattere | Formato RTF | Aggiorna tutto | Salva | Controllo ortografia | Z-A | Avanzate | Trova

Tutte le tabelle | Anagrafica

| ID | Cognome | Nome | DataNascita | Luogo | Sesso | Via | CAP | Città |
|----|---------|--------|-------------|-------|-------|-------------|-------|-------|
| | Verdi | Franco | 03/09/2019 | Roma | M | Via Lunga 7 | 00100 | Roma |

Connessione al database - linguaggio Java

Cancellazione.java

```
package cancellazione;
import java.sql.*;
public class Cancellazione
{
    public static void main(String[] args)
    {
        Connection conn=null;
        Statement st=null;
        String sql;
        String url= "jdbc:ucanaccess://C:\\Users\\room\\Desktop\\corso ingegneria\\Personale.accdb";
        sql="DELETE FROM Anagrafica WHERE ID=1";
        try
        {
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
            conn=DriverManager.getConnection(url, "", "");
            st=conn.createStatement();
            int righe=st.executeUpdate(sql); //metodo executeUpdate per l'oggetto st: esegue il comando sql di cancellazione
            if (righe>0) //il valore di ritorno del metodo executeUpdate consente di controllare se il record è stato
            { //effettivamente eliminato
                System.out.println("Recod eliminato");
            }
            else
            {
                System.out.println("Recod da eliminare non trovato");
            }
        }
        catch (ClassNotFoundException | SQLException e)
        {
            System.out.println("SQLException:");
            System.out.println(e.getMessage());
        }
        finally
        {
            if (st !=null)
            { //metodo close per l'oggetto st:rilascia le risorse
              //usate dal comando SQL
              try {st.close();} catch(SQLException e){}
            }
            if (conn!=null)
            { //metodo close per l'oggetto conn: interrompe
              //la connessione al database
              try {conn.close();} catch(SQLException e){}
            }
        }
    }
}
```

Cancellazione.java

Nella pagina precedente è riportato il codice per cancellare dall'archivio Anagrafica i dati della persona con ID=1.

L'istruzione Java per eliminare una persona dall'archivio Anagrafica è formata dal comando SQL **DELETE FROM Anagrafica WHERE ID=1**. Si riporta di seguito lo screenshot dell'output di NetBeans e la nuova tabella Anagrafica con il campo ID=1 cancellato.

```
33     {
34         if (st !=null)
35             {
36                 //metodo close per l'oggetto st:rilascia le risorse
37                 //usate dal comando SQL
38                 try {st.close();} catch(SQLException e){}
39             }
40         if (conn!=null)
41             {
42                 //metodo close per l'oggetto conn: interrompe
43                 //la connessione al database
44                 try {conn.close();} catch(SQLException e){}
45             }
46     }
47 }
```

Output - Cancellazione (run) ✖

```
run:
Recod eliminato
BUILD SUCCESSFUL (total time: 1 second)
```

| ID | Cognome | Nome | DataNascita | Luogo | Sesso | Via | CAP | Città |
|----|---------|-----------|-------------|-----------|-------|-------------|-------|-----------|
| 1 | Bianchi | Roberto | 05/09/2017 | Cassino | M | Via Lunga 7 | 03040 | Cassino |
| 6 | Rossi | Antonella | 03/09/2001 | Frosinone | F | Via Lunga 7 | 03040 | Frosinone |
| 7 | Carbone | Bernardo | 03/09/2019 | Milano | M | Via Lunga 7 | 20019 | Milano |
| 8 | Bianchi | Anna | 03/09/2018 | Padova | F | Via Lunga 7 | 35100 | Padova |

I PARAMETRI NEI COMANDI SQL

Nei programmi precedenti abbiamo utilizzato il metodo `createStatement` proprio per attivare i comandi SQL che non contengono parametri.

Quando si devono usare i parametri nelle istruzioni SQL, il software API JDBC mette a disposizione il metodo `prepareStatement`. Tale metodo crea un'istanza della classe `PreparedStatement`, la quale, essendo sottoclasse di `Statement`, eredita proprio dalla `Statement` tutte le funzionalità. Si precisa che il metodo `prepareStatement` è utile quando si devono eseguire istruzioni SQL con parametri.

I parametri sono indicati nel comando SQL con il punto interrogativo. Vediamo un esempio di utilizzo di parametri:

```
PreparedStatement st=conn.prepareStatement("UPDATE Anagrafica SET Telefono = ? WHERE ID = ?);  
Successivamente, dopo l'istanza dell'oggetto st, si devono usare i metodi setString o setInt. Si indica in generale il metodo con setXXX, dove XXX sarà sostituito dal tipo di dato usato. Tale metodo deve contenere nei parametri i rispettivi valori numerici in base alla posizione occupata nella query (da sinistra verso destra a partire dal numero 1).
```

Esempio:

```
st.setString(1,"02/9993432"); //nuovo telefonico (parametro 1)  
st.setInt(2,300); //codice (parametro 2)
```

A questo punto si può eseguire il comando SQL con il metodo `executeUpdate`, già usato nella classe Cancellazione, contenente anche il valore di ritorno. Tale valore consente di verificare se il record è stato effettivamente aggiornato :

```
int righe=st.executeUpdate();
```

Infine si ricorda che i metodi `setXXX` possono utilizzare costanti o dati inseriti da tastiera.

I PARAMETRI NEI COMANDI SQL

L'utilizzo di parametri nei comandi SQL, anche tramite *PreparedStatement*, consente una sorta di protezione contro possibili minacce legate alla sicurezza dei dati. In quanto durante le operazione di input di dati, anche tramite i parametri, si può verificare, all'interno di un'applicazione web di java o tramite un form di un sito web, un'aggiunta di informazione indesiderate da parte di utenti non autorizzati.

Tale operazione è definita con il termine **SQL Injection**, nel gergo informatico malicious SQL, in italiano codice malevolo o codice doloso.

L'operazione di aggiornamento

*Creare un file **Aggiornamento.java** che svolga le seguenti operazioni.*

Dato l'archivio Anagrafica, implementare un programma Java che permetta di aggiornare il numero di telefono di una persona. Si precisa che il numero di telefono dovrà essere inserito dall'utente.

Aggiornamento.java

```
package aggiornamento; import java.sql.*; import java.io.*;
public class Aggiornamento
{ public static void main(String[] args) {
    int codice = 0; String telef;
    InputStreamReader input=new InputStreamReader(System.in); //impostazione dello standard input
    BufferedReader tastiera=new BufferedReader(input);
    String sql="UPDATE Anagrafica SET Telefono = ? WHERE ID = ?"; //stringa per il dirver del database
    String url= "jdbc:ucanaccess://C:\Users\room\Desktop\corso ingegneria\Personale.accdb";
    Connection conn=null;
    PreparedStatement st=null;
    try
    { Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
      conn=DriverManager.getConnection(url, "", ""); //connessione al database
      st=conn.prepareStatement(sql);
      //inserire da tastiera dei dati
      try { System.out.print("Codice da aggiornare: ");
          String numeroLetto=tastiera.readLine();
          codice =Integer.parseInt(numeroLetto);
          System.out.print("Nuovo numero telefonico ");
          telef=tastiera.readLine();
        }
      catch(Exception e) { System.err.print("\nInserimento errato ");
                          System.err.println(e.getMessage());
                          return; }
      st.setString(1,telef); //esegue il comando SQL
      st.setInt(2,codice);
      int righe =st.executeUpdate();
      if (righe>0) { System.out.println("Righe aggiornate: "+righe); }
      else { System.out.println("Codice "+codice+" non trovato"); }
    }
    catch (ClassNotFoundException | SQLException e)
    {
      System.out.println("SQLException:");
      System.out.println(e.getMessage());
    }
    finally {
      if (st !=null) { //metodo close per l'oggetto stmt:rilascia le risorse usate dal comando SQL
                    try {st.close();} catch(SQLException e){}
                    }
      if (conn!=null) { //metodo close per l'oggetto con: interrompe la connessione al database
                      try {conn.close();} catch(SQLException e){}
                      }
    }
  }
}
```

Aggiornamento.java

```
37     if (st !=null)          { //metodo close per l'oggetto stmt:rilascia le risorse usate dal comando S
38                             try {st.close();} catch(SQLException e){
39                                 }
40     if (conn!=null)         { //metodo close per l'oggetto con: interrompe la connessione al data
41                             try {conn.close();} catch(SQLException e){
42                                 }
43     } }
44
```

Output - Aggiornamento (run) %

```
run:
Codice da aggiornare: 4
Nuovo numero telefonico 07763024438
Righe aggiornate: 1
BUILD SUCCESSFUL (total time: 18 seconds)
```

Strumenti tabella Personale : Database (Access 2007) - Microsoft Access

Home Crea Dati esterni Strumenti database Foglio dati

Visualizza Incolla Taglia Copia Copia formato Calibri 11 Carattere Formato RTT Record Ordina e filtra Trova

Tutte le tabelle Anagrafica

| ID | Cognome | Nome | DataNascita | Luogo | Sesso | Via | CAP | Città | Provincia | Telefono |
|----|----------|-----------|-------------|-----------|-------|-------------|-------|-----------|-----------|-------------|
| 1 | Bianchi | Roberto | 05/09/1917 | Cassino | M | Via Lunga 7 | 03040 | Cassino | FR | 07763024438 |
| 6 | Rossi | Antonella | 03/09/2001 | Frosinone | F | Via Lunga 7 | 03040 | Frosinone | FR | 0775/30122 |
| 7 | Carbone | Bernardo | 03/09/1919 | Milano | M | Via Lunga 7 | 20019 | Milano | MI | 02/2982828 |
| 8 | Buccheri | Anna | 03/09/1919 | Padova | F | Via Lunga 7 | 35100 | Padova | PD | 049/2982828 |

Le interrogazioni - Proiezione

Per eseguire le interrogazioni al database si utilizza il metodo **executeQuery**. Si vuole, ad esempio, effettuare una **proiezione** della tabella **Anagrafica** sulle colonne **ID, Cognome e Nome**.

In tal caso, occorre scrivere la seguente istruzione:

```
ResultSet rs=st.executeQuery("SELECT id,Cognome,Nome FROM Anagrafica");
```

Diversamente si può passare al metodo `executeQuery`, come parametro, una stringa sql che contiene il comando SQL:

```
String sql= "SELECT id,Cognome,Nome FROM Anagrafica";
```

.....

```
ResultSet rs=st.executeQuery(sql);
```

Si precisa che il valore di ritorno del metodo `executeQuery` è un singolo **ResultSet**, ossia un insieme di righe della tabella **Anagrafica** che soddisfano la condizione specificata nell'interrogazione SQL. Dopo la query verrà generata una nuova tabella formata dalle righe selezionate e dalle colonne scelte. Per scorre nella tabella generata si potrà usare il metodo **next**. Il metodo *next* si trova automaticamente posizionato sulla prima riga. Quindi la visualizzazione dei risultati contenuti nel *ResultSet* si ottengono con il ciclo iterativo di lettura usando i metodi `getXXX`, dove *XXX* indica il tipo di dato.

Vediamo nella pagina successiva il programma **Selezione.java** che esegue una interrogazione, con operazione di proiezione, sulla tabella **Anagrafica**.

Selezione.java

```
package selezione; import java.sql.*;
public class Selezione
{   public static void main(String[] args)   {
    String sql; //stringa sql per il driver del database
    sql = "SELECT ID, Cognome, Nome, DataNascita "+
        "FROM Anagrafica "+
        "WHERE Provincia ='MI'";
    String url= "jdbc:ucanaccess://C:\\Users\\room\\Desktop\\corso ingegneria\\Personale.accdb";
    Connection conn=null;
    Statement st=null;
    ResultSet rs=null;
    try
    {
        Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
        conn=DriverManager.getConnection(url, "", ""); //connessione al database
        //esecuzione della query
        st=conn.createStatement();
        rs=st.executeQuery(sql);
        //righe della tabella dei risultati
        while (rs.next())
        {
            int cod=rs.getInt(1);
            String cogn=rs.getString(2);
            String nom=rs.getString(3);
            String datan=rs.getString(4);
            //visualizza campi
            System.out.println("Codice   = "+cod);
            System.out.println("Cognomee = "+cogn);
            System.out.println("Nome     = "+nom);
            System.out.println("Data Nascita= "+datan);
            System.out.print("\n");
        }
    }
    catch(ClassNotFoundException | SQLException e)   { System.err.print("SQLException: ");
        System.err.println(e.getMessage());}
    finally   {
        if (st !=null)   { //metodo close per l'oggetto st:rilascia le risorse usate dal comando SQL
            try {st.close();} catch(SQLException e){}
        }
        if (conn!=null)   { //metodo close per l'oggetto conn: interrompe la connessione al database
            try {conn.close();} catch(SQLException e){}
        }
    }
} } }
```


Le interrogazioni – Proiezione tramite input da tastiera della provincia

Si vuole visualizzare il codice, il cognome, il nome e la data di nascita delle persone residenti in una provincia digitata da tastiera.

Si utilizza lo stesso codice del programma Selezione.java, modificando però alcune righe di programma. In questo caso si utilizza l'istruzione:

```
PreparedStatement st=null
```

al posto di **Statement st=null**, in quanto abbiamo un comando select con il parametro.

In aggiunta bisogna utilizzare nella try l'istruzione

```
st=conn.prepareStatement(query);
```

al posto di **st=conn.createStatement()**

Infine bisogna togliere l'istruzione **rs=st.executeQuery(sql);**
ed aggiungere le seguenti istruzioni

```
st.setString(1,prov);  
rs=st.executeQuery();
```

Si consideri anche l'impostazione dello standard input.

Riportiamo di seguito il codice sorgente della classe **Selezione_param.java**

Proiezione tramite input da tastiera – file Selezione_param.java

```
package selezione_param; import java.sql.*; import java.io.*;
public class Selezione_param
{   public static void main(String[] args)   {
    String prov;
    //impostazione dello standard input
    InputStreamReader input=new InputStreamReader(System.in); //impostazione dello standard input
    BufferedReader tastiera=new BufferedReader(input);

    String sql; //stringa sql per il driver del database
    sql = "SELECT ID, Cognome, Nome, DataNascita "+
        "FROM Anagrafica "+
        "WHERE Provincia = ?";
    String url= "jdbc:ucanaccess://C:\\Users\\room\\Desktop\\corso Ingegneria\\Personale.accdb";
    Connection conn=null;
    PreparedStatement st=null;
    ResultSet rs=null;
    try
    {   Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
        conn=DriverManager.getConnection(url, "", ""); //connessione al database
        st=conn.prepareStatement(sql); //statement di preparazione all'aggiornamento con un parametro
        //inserimento da tastiera della provincia
        try
        {   System.out.print("Provincia richiesta :"); prov=tastiera.readLine(); }
        catch(Exception e) {System.err.print("\nInserimento errato "); System.err.println(e.getMessage());return; }
        //esegui il comando sql
        st.setString(1,prov);
        rs=st.executeQuery();
        //righe della tabella dei risultati
        while (rs.next())
        {   int cod=rs.getInt(1);
            String cogn=rs.getString(2);
            String nom=rs.getString(3);
            String datan=rs.getString(4);
            //visualizza campi
            System.out.println("Codice   = "+cod);
            System.out.println("Cognomee = "+cogn);
            System.out.println("Nome     = "+nom);
            System.out.println("Data Nascita= "+datan);
            System.out.print("\n");
        }
        }
        catch(ClassNotFoundException | SQLException e)   {   System.err.print("SQLException: ");
            System.err.println(e.getMessage());}
    finally   {   if (st !=null)   { //metodo close per l'oggetto st:rilascia le risorse usate dal comando
        SQL
            try {st.close();} catch(SQLException e){}
        }
        if (conn!=null)
        { //metodo close per l'oggetto conn: interrompe la connessione al database
            try {conn.close();} catch(SQLException e){}
        } } } }
```

Proiezione tramite input da tastiera della provincia (File Selezione_param.java)

The screenshot displays an IDE environment. The top pane shows Java code with a try-catch block for closing a database statement. The middle pane shows the output of a program run, displaying user input and database results. The bottom pane shows a database table named 'Anagrafica' with columns for personal and contact information.

```
47 | 11 (st !=null) | //metodo close per l'oggetto st:rilascia le risorse usate dal comando SQL  
48 | | try {st.close();} catch (SQLException e){}
```

selezione_tastiera.Selezione_tastiera > main > try > finally > if (st != null) >

Output - Selezione_tastiera (run) >

```
run:  
Provincia richiesta :RM  
Codice = 14  
Cognomee = Bianchi  
Nome = Roberto  
Data Nascita= 2019-09-03 00:00:00.000000  
  
Codice = 15  
Cognomee = Bianchi  
Nome = Roberto  
Data Nascita= 2019-09-03 00:00:00.000000  
  
Codice = 16  
Cognomee = Bianchi  
Nome = Roberto  
Data Nascita= 2019-09-03 00:00:00.000000  
  
Codice = 17  
Cognomee = Bianchi
```

| ID | Cognome | Nome | DataNascita | Luogo | Sesso | Via | CAP | Città | Provincia | Telefono | DataAggio | CodiceFisc |
|----|---------|-----------|-------------|-----------|-------|-------------|-------|-----------|-----------|-------------|------------|------------|
| | Bianchi | Roberto | 05/09/2017 | Cassino | M | Via Lunga 7 | 03040 | Cassino | FR | 07763024438 | 12/12/2019 | BNCBGL75E |
| 6 | Rossi | Antonella | 03/09/2001 | Frosinone | F | Via Lunga 7 | 03040 | Frosinone | FR | 0775/30122 | 12/12/2019 | BNCBGL75E |
| 14 | Bianchi | Roberto | 03/09/2019 | Roma | M | Via Lunga 7 | 20102 | Roma | RM | 02/2982828 | 12/12/2019 | BNCBGL75E |
| 15 | Bianchi | Roberto | 03/09/2019 | Roma | M | Via Lunga 7 | 20102 | Roma | RM | 02/2982828 | 12/12/2019 | BNCBGL75E |
| 16 | Bianchi | Roberto | 03/09/2019 | Roma | M | Via Lunga 7 | 20102 | Roma | RM | 02/2982828 | 12/12/2019 | BNCBGL75E |
| 17 | Bianchi | Roberto | 03/09/2019 | Roma | M | Via Lunga 7 | 20102 | Roma | RM | 02/2982828 | 12/12/2019 | BNCBGL75E |
| 18 | Bianchi | Roberto | 03/09/2019 | Roma | M | Via Lunga 7 | 20102 | Roma | RM | 02/2982828 | 12/12/2019 | BNCBGL75E |
| 19 | Bianchi | Roberto | 03/09/2019 | Roma | M | Via Lunga 7 | 20102 | Roma | RM | 02/2982828 | 12/12/2019 | BNCBGL75E |
| * | (Nuovo) | | | | | | | | | | | |

I metadati

In Java è possibile conoscere tutte le informazioni sulla struttura del database (cioè i **metadati**). Il JDBC mette a disposizione una classe chiamata **DataBaseMetaData**, che tramite i propri metodi permette di conoscere tutte le informazioni sulla struttura del database. Il tutto avviene con un unico insieme, restituendo un oggetto **ResultSetMetaData** per ogni *ResultSet* ottenuto con una query.

L'istruzione **ResultSet.getMetaData** restituisce un oggetto *ResultSetMetaData* che fornisce tutte le informazioni riguardanti il numero, il tipo e le proprietà delle colonne della tabella.

Vediamo nella prossima pagina il programma che consente di visualizzare il nome e il tipo per ciascuna colonna di una tabella del database.

I metodi utilizzati della classe *DatabaseMetaData*, applicati all'oggetto *ResultSetMetaData*, possono visualizzare come già detto la struttura della tabella, con l'elenco dei campi, il numero, il nome e tipo di ciascuna colonna.

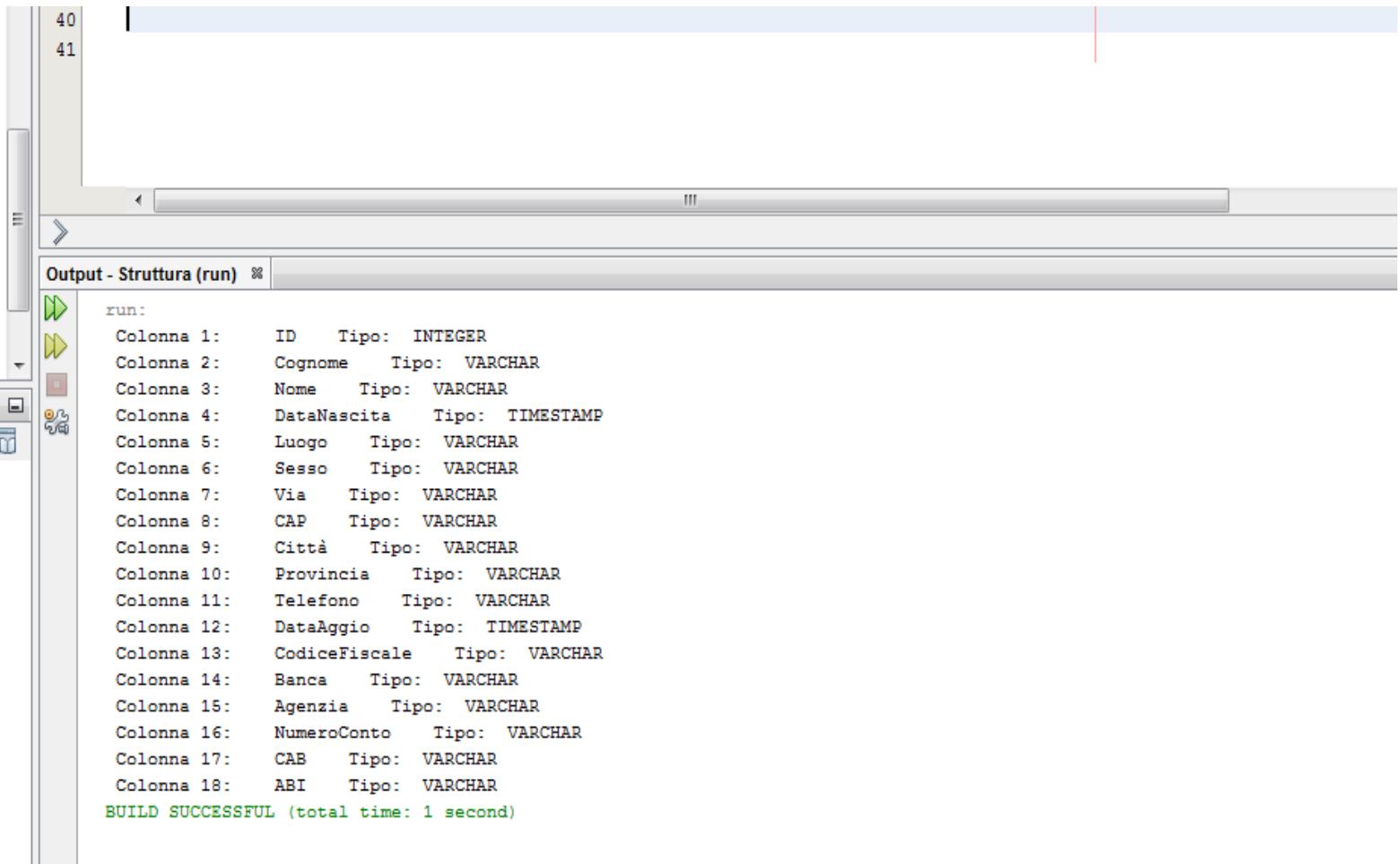
Il file che andremo ad implementare nella pagina successiva è denominato **Struttura.java**

I metadati – file Struttura.java

```
package struttura; import java.sql.*;
public class Struttura
{   public static void main(String[] args)   {
    String sql; //stringa sql per il driver del database
    sql = "SELECT * FROM Anagrafica ";
    String url= "jdbc:ucanaccess://C:\\Users\\room\\Desktop\\corso ingegneria\\Personale.accdb";
    Connection conn=null;
    Statement st=null;
    ResultSet rs=null;
    try
    {   Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
        conn=DriverManager.getConnection(url, "", ""); //connessione al database
        st=conn.createStatement();//statement di preparazione all'aggiornamento con un parametro
        rs=st.executeQuery(sql);
        //metadati
        ResultSetMetaData metadati=rs.getMetaData();
        int numberOfColumns=metadati.getColumnCount();
        for (int i=1;i<=numberOfColumns;i++)
        {
            String n=metadati.getColumnName(i);
            String t=metadati.getColumnTypeName(i);
            System.out.print(" Colonna "+i+":\t"+n);
            System.out.println(" Tipo: "+t);
        }
    }
    catch(ClassNotFoundException | SQLException e)
    {
        System.err.print("SQLException: ");
        System.err.println(e.getMessage());
    }
    finally
    {
        if (st !=null)
        { //metodo close per l'oggetto st:rilascia le risorse usate dal comando SQL
            try { st.close(); }
            catch(SQLException e) { }
        }
        if (conn!=null) { //metodo close per l'oggetto conn: interrompe la connessione al database
            try { conn.close(); }
            catch(SQLException e) { } } } } }
```

I metadati – file Struttura.java

Struttura della tabella Anagrafica (output della struttura)



The screenshot shows an IDE window with a code editor at the top and an output console at the bottom. The code editor shows lines 40 and 41, which are currently blank. The output console, titled "Output - Struttura (run)", displays the following text:

```
run:  
Colonna 1: ID Tipo: INTEGER  
Colonna 2: Cognome Tipo: VARCHAR  
Colonna 3: Nome Tipo: VARCHAR  
Colonna 4: DataNascita Tipo: TIMESTAMP  
Colonna 5: Luogo Tipo: VARCHAR  
Colonna 6: Sesso Tipo: VARCHAR  
Colonna 7: Via Tipo: VARCHAR  
Colonna 8: CAP Tipo: VARCHAR  
Colonna 9: Città Tipo: VARCHAR  
Colonna 10: Provincia Tipo: VARCHAR  
Colonna 11: Telefono Tipo: VARCHAR  
Colonna 12: DataAggio Tipo: TIMESTAMP  
Colonna 13: CodiceFiscale Tipo: VARCHAR  
Colonna 14: Banca Tipo: VARCHAR  
Colonna 15: Agenzia Tipo: VARCHAR  
Colonna 16: NumeroConto Tipo: VARCHAR  
Colonna 17: CAB Tipo: VARCHAR  
Colonna 18: ABI Tipo: VARCHAR  
BUILD SUCCESSFUL (total time: 1 second)
```

L'accesso ai database in rete

Le applicazioni implementate finora utilizzano connessioni, tramite il driver UCanAccess, a database locali creati con Microsoft Access. Abbiamo anche accennato alla possibilità di eseguire le operazioni di manipolazione e di interrogazione su database presenti su un server di rete . Nelle prossime pagine andremo a sviluppare classi in java che permettono il collegamento ad un database di MySQL sul server locale (localhost).

Riportiamo di seguito le installazioni necessarie per implementare le classi in Java.

Prima di tutto scarichiamo ed installiamo il software **MySQL Community Edition** compatibile con il nostro hardware.

Il software **MySQL Community Edition** utilizzato durante le esercitazioni è:

mysql-installer-community-8.0.19.0.msi

L'accesso ai database in rete

Durante l'installazione si può scegliere di includere pacchetti opzionali.

Nel nostro caso abbiamo installato:

-**MySQL Server 5.7.29** (si ricorda che esistono piattaforme server sia LAMP che WAMP)

-**MySQL Workbench 6.3.8** (è uno strumento visuale di progettazione per database in grado di creare tabelle, eseguire operazioni di manutenzione sui database e svolgere altre utili operazioni sulla propria postazione di lavoro)

-**MySQL Connector/J 8.0.19** (MySQL mette a disposizione un connector per il linguaggio Java, in quanto bisogna far interagire i DBMS con JDBC)

Nella prossima pagina si riportano i vari passaggi necessari per la configurazione di **MySQL Community Edition**.

Quindi andare alla pagina <https://dev.mysql.com/downloads>

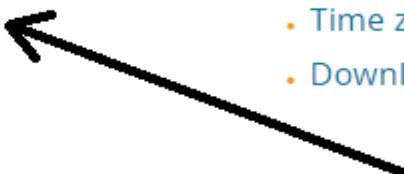
L'accesso ai database in rete

Ⓣ MySQL Community Downloads

- MySQL Yum Repository
 - MySQL APT Repository
 - MySQL SUSE Repository

 - MySQL Community Server
 - MySQL Cluster
 - MySQL Router
 - MySQL Shell
 - MySQL Workbench

 - MySQL Installer for Windows
 - MySQL for Excel
 - MySQL for Visual Studio
 - MySQL Notifier
- C API (libmysqlclient)
 - Connector/C++
 - Connector/J
 - Connector/NET
 - Connector/Node.js
 - Connector/ODBC
 - Connector/Python
 - MySQL Native Driver for PHP

 - MySQL Benchmark Tool
 - Time zone description tables
 - Download Archives
- 

ORACLE © 2020, Oracle Corporation and/or its affiliates

[Legal Policies](#) | [Your Privacy Rights](#) | [Terms of Use](#) | [Trademark Policy](#) | [Contributor Agreement](#) | [Cookie Preferences](#)

L'accesso ai database in rete

MySQL Community Downloads

MySQL Installer

General Availability (GA) Releases

Archives



MySQL Installer 8.0.19

Select Operating System:

Microsoft Windows

Looking for previous GA versions?

Windows (x86, 32-bit), MSI Installer

8.0.19

18.6M

Download

(mysql-installer-web-community-8.0.19.0.msi)

MD5: 32043776cb2239db45fddaa86dc0ad61 | Signature

Windows (x86, 32-bit), MSI Installer

8.0.19

398.9M

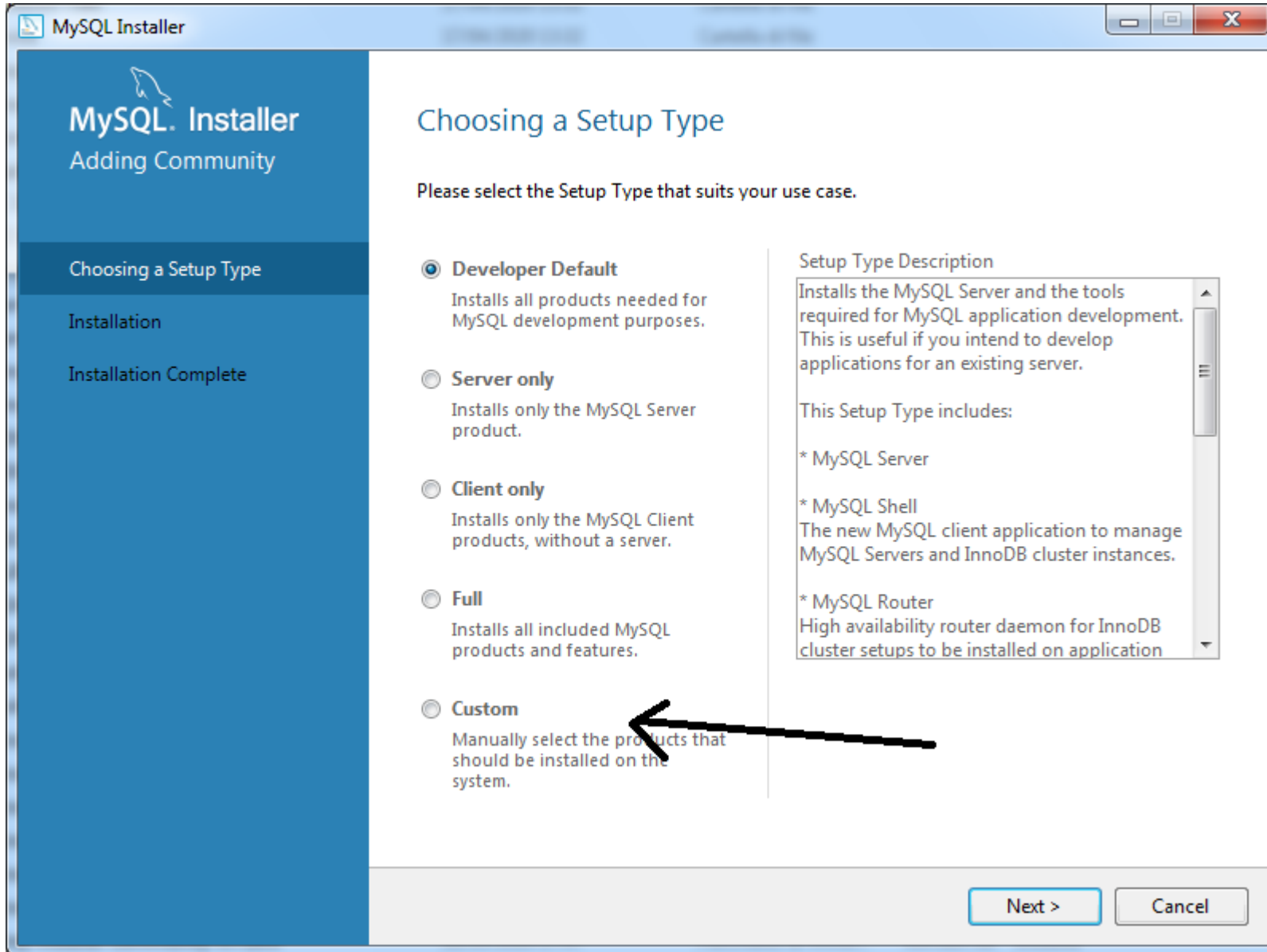
Download

(mysql-installer-community-8.0.19.0.msi)

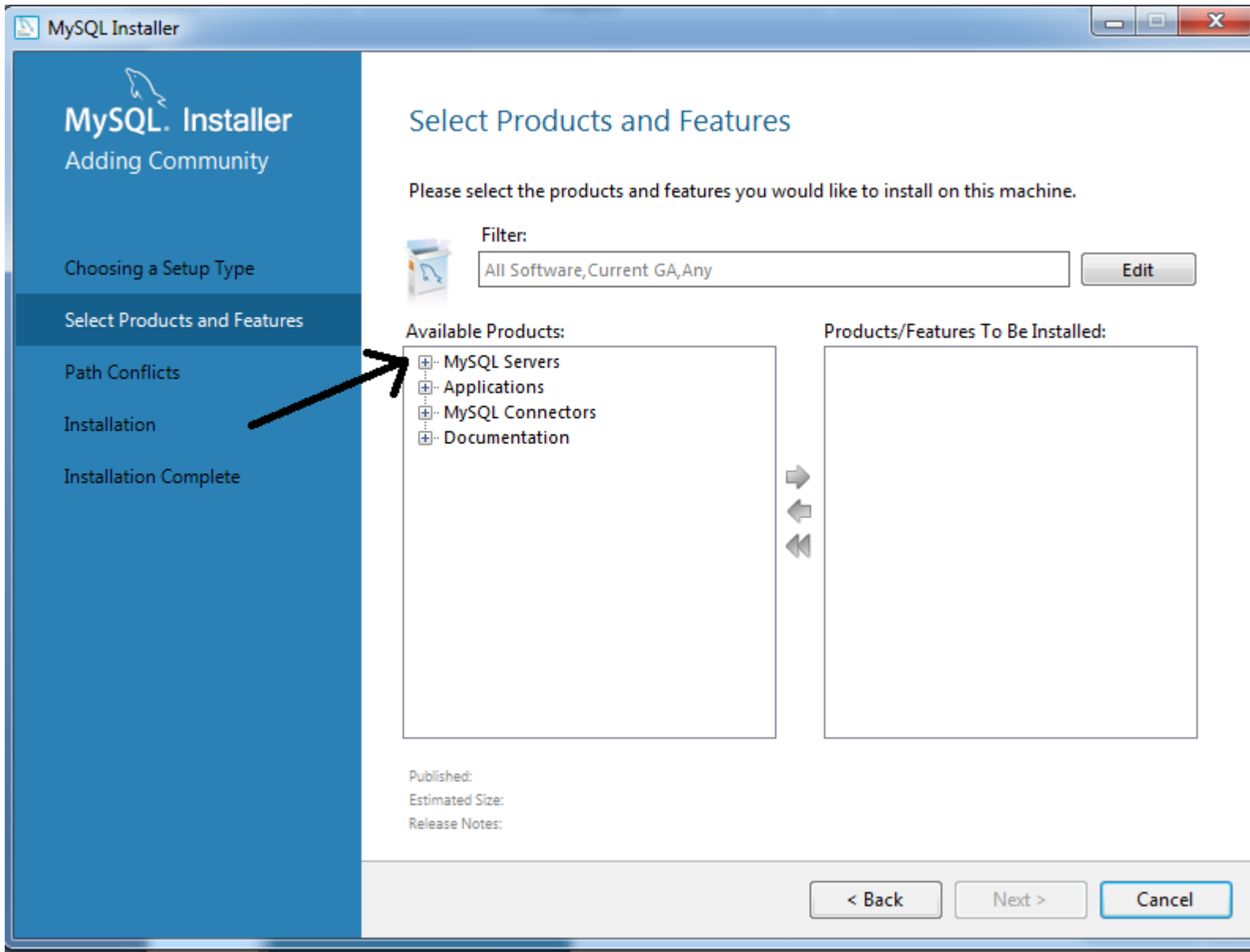
MD5: 1a882015da7fb93f20c4717e63b6817c | Signature

 We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

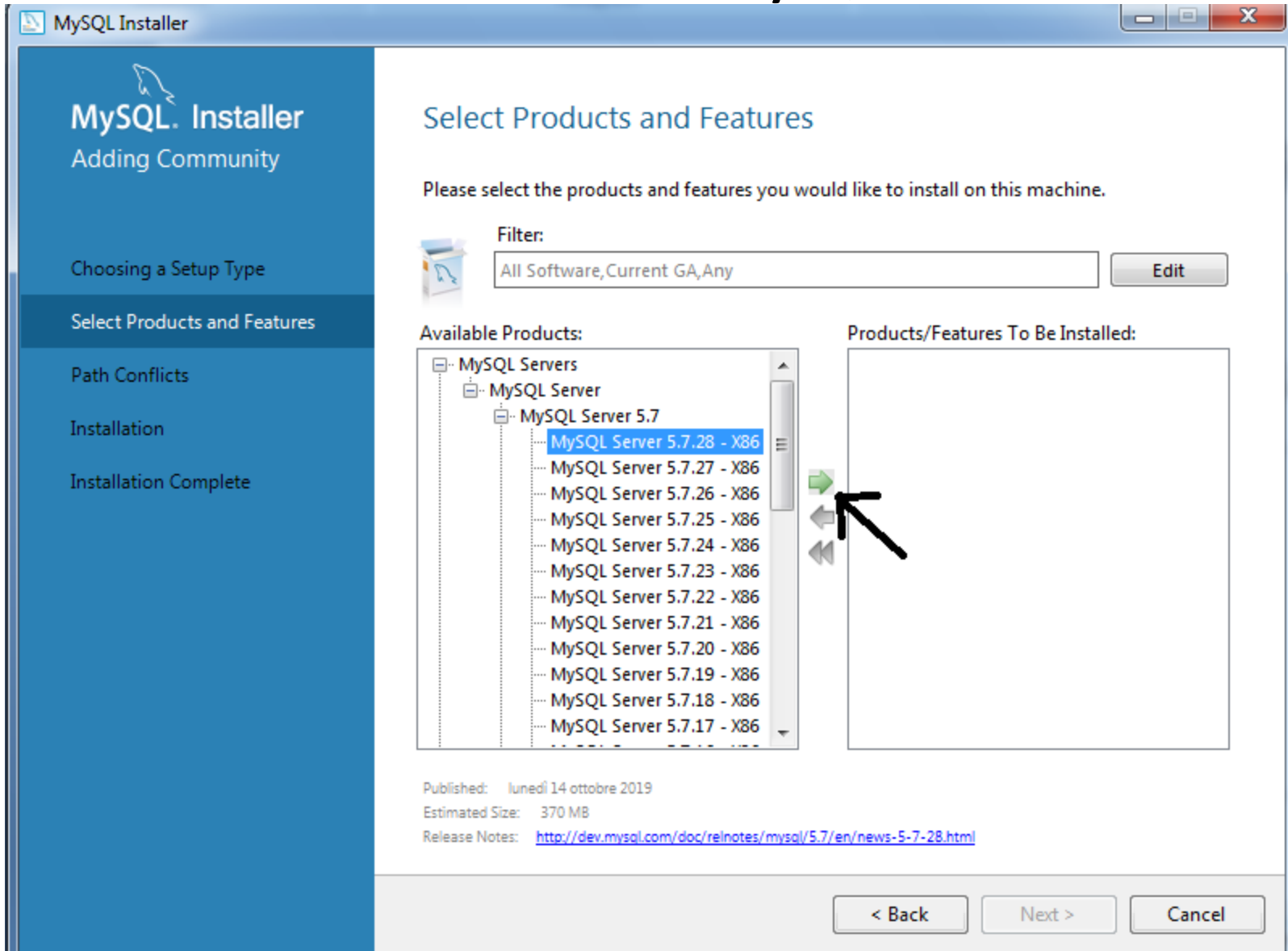
L'accesso ai database in rete



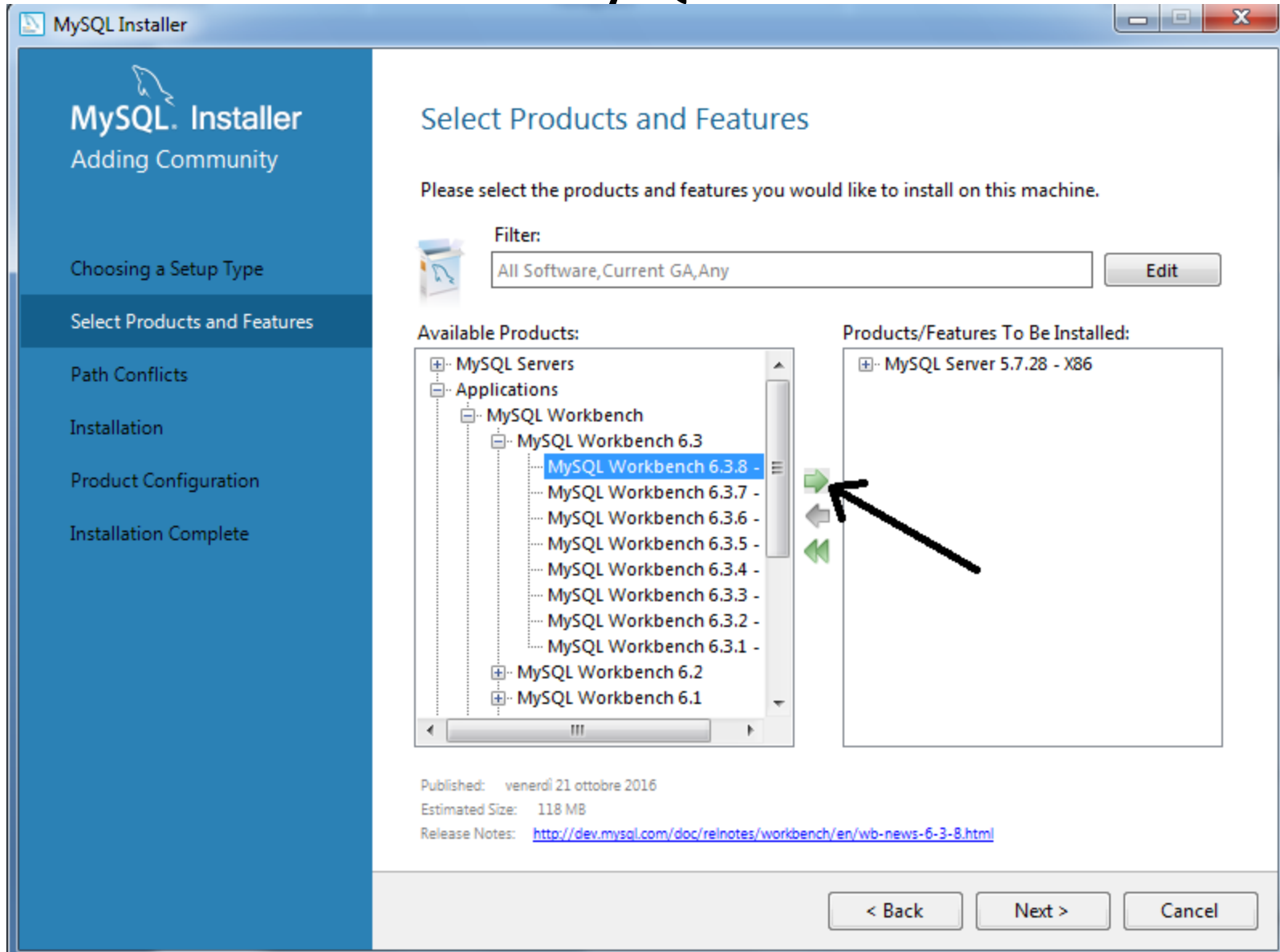
Software MySQL Server



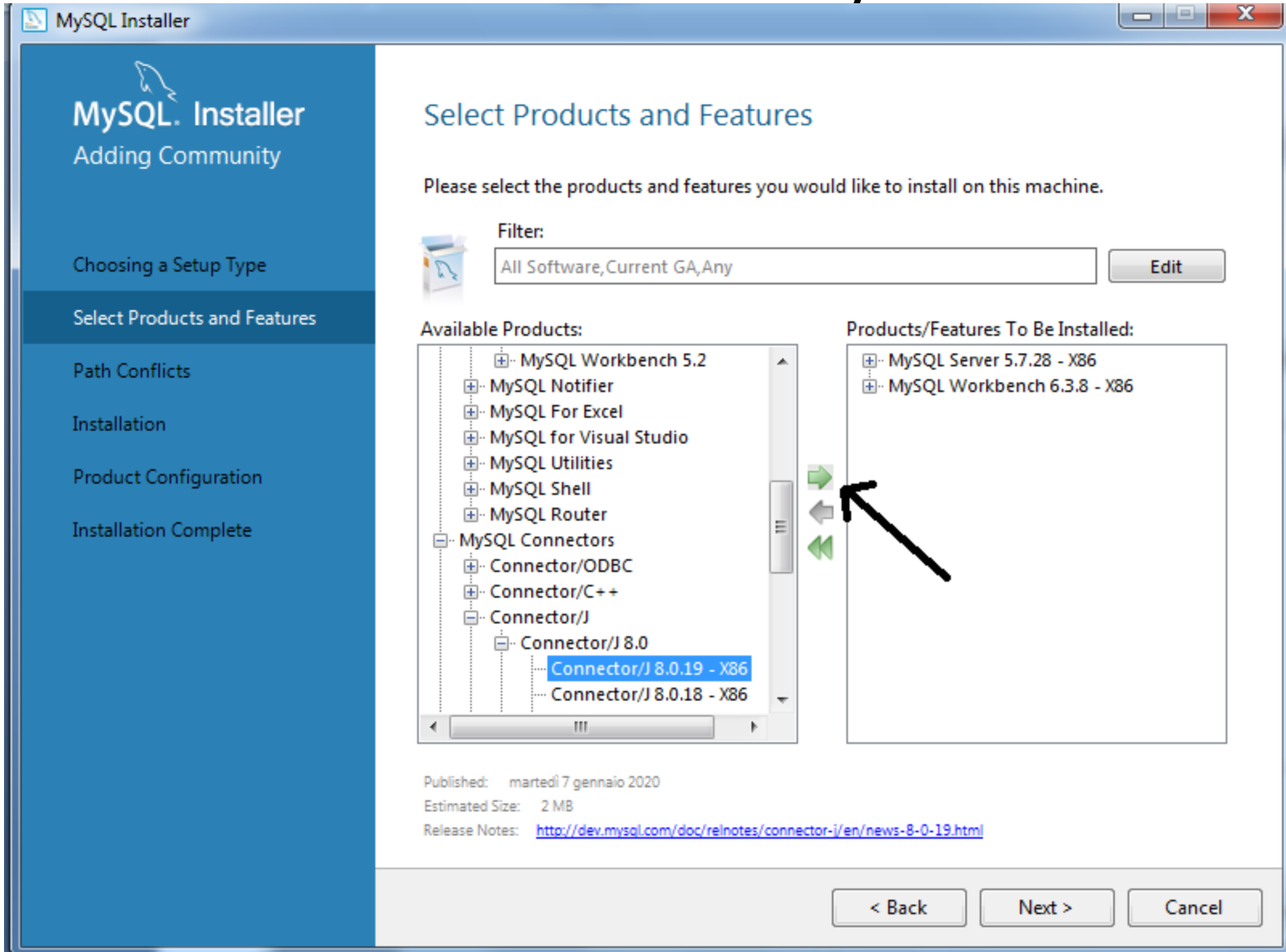
Selezione software MySQL Server



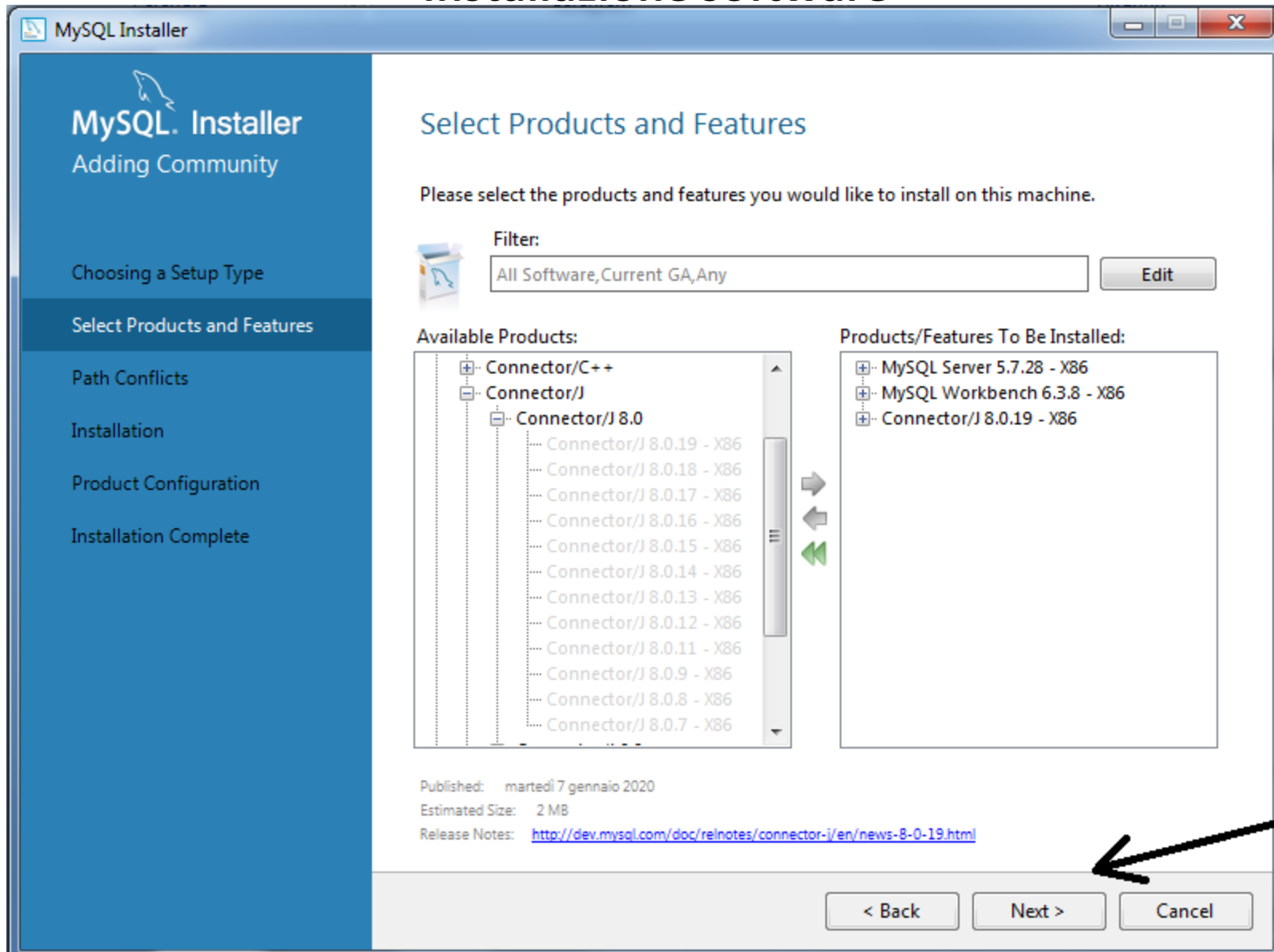
Software MySQL Workbench



Software Connector/J

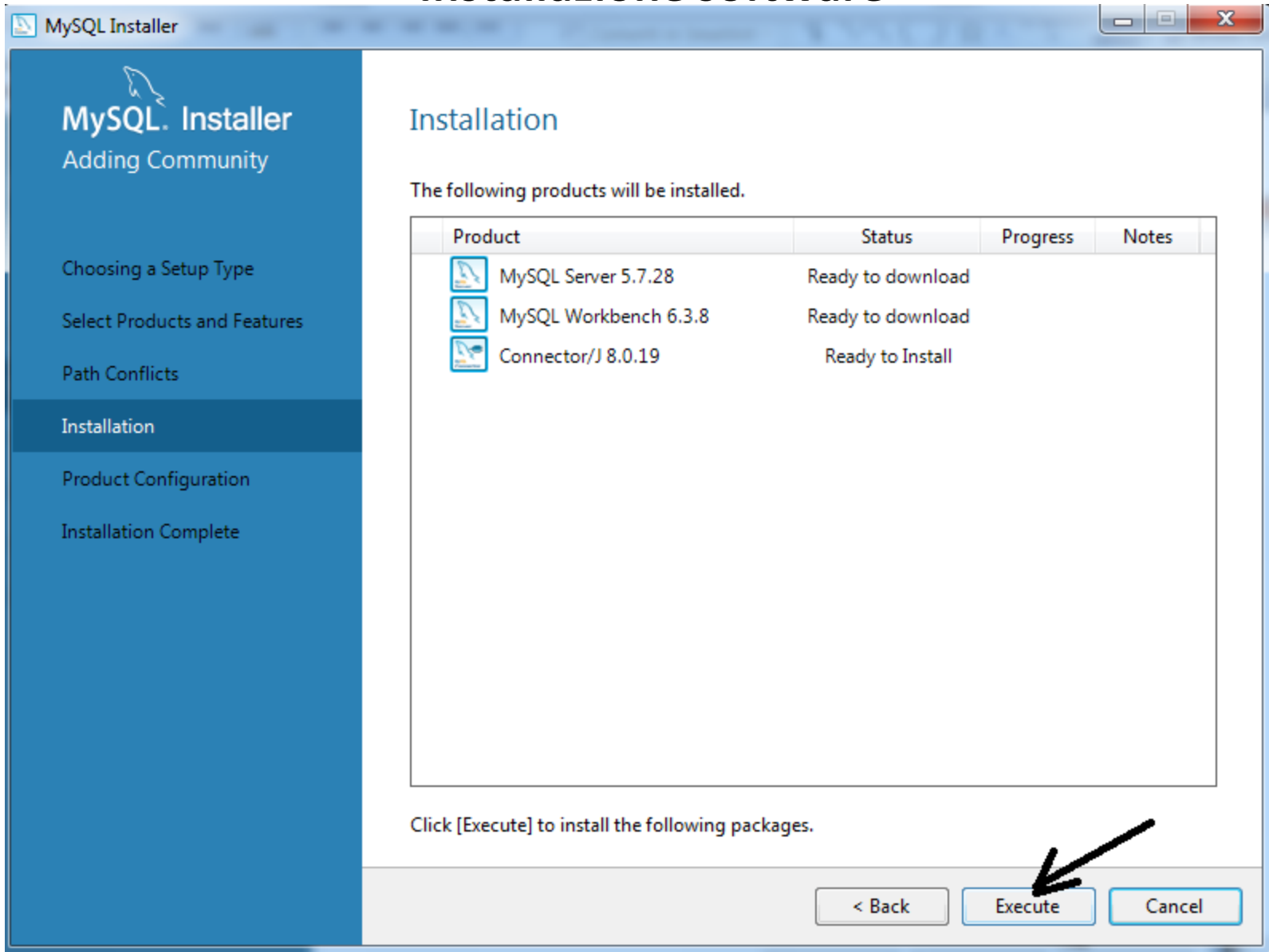


Installazione software

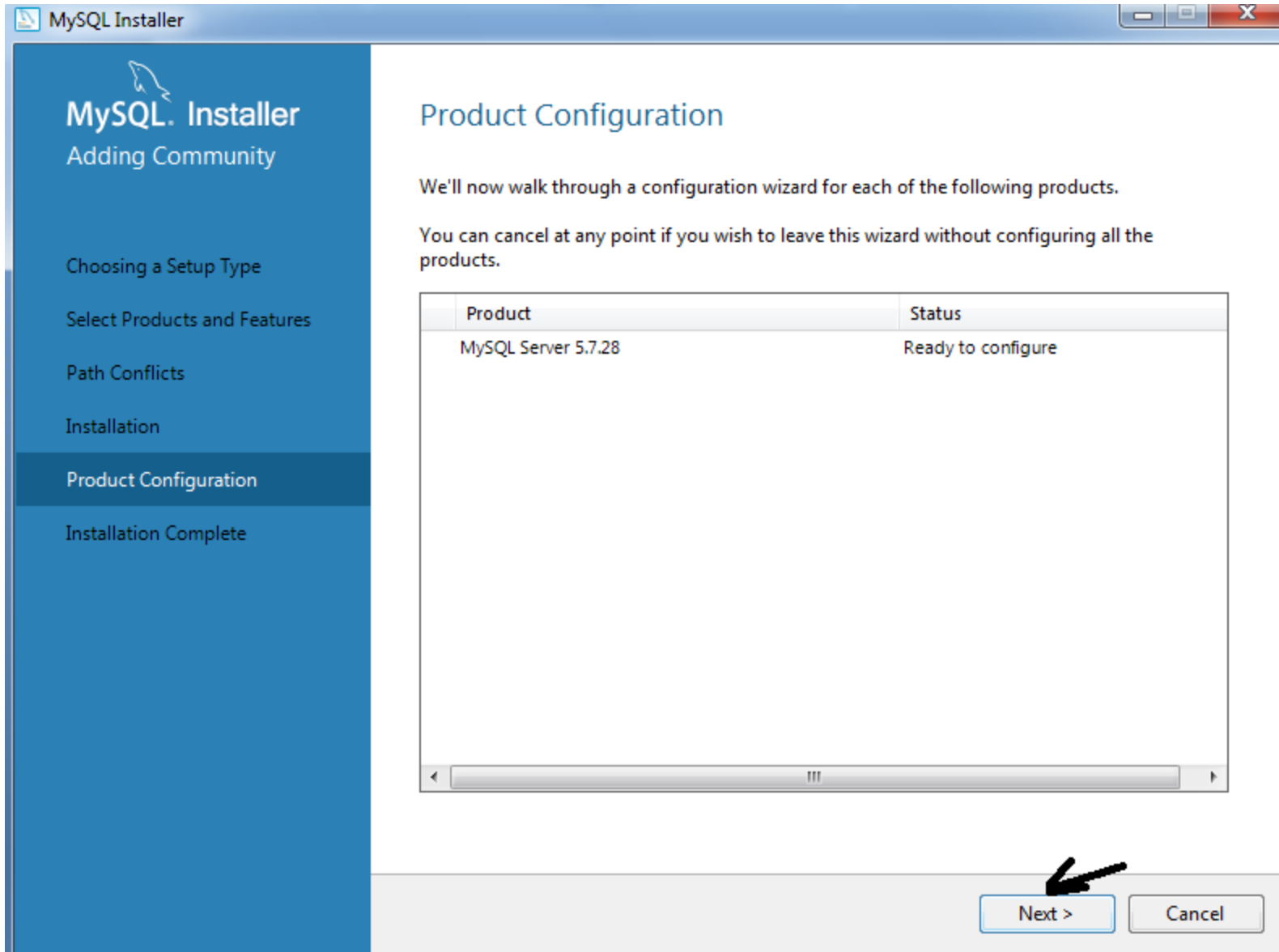


Connessione al database - linguaggio Java

Installazione software



Installazione software



Configurazione MySQL Server

MySQL Installer
MySQL Server 5.7.28

High Availability

Type and Networking

Accounts and Roles

Windows Service

Apply Configuration

High Availability

- Standalone MySQL Server / Classic MySQL Replication**
Choose this option to run the MySQL instance as a standalone database server with the opportunity to configure classic replication later. With this option, you can provide your own high-availability solution, if required.
- Sandbox InnoDB Cluster Setup (for testing only)** [Why is this option disabled?](#)
The [InnoDB cluster](#) technology provides an out-of-the-box HA (high availability) solution for MySQL using Group Replication.

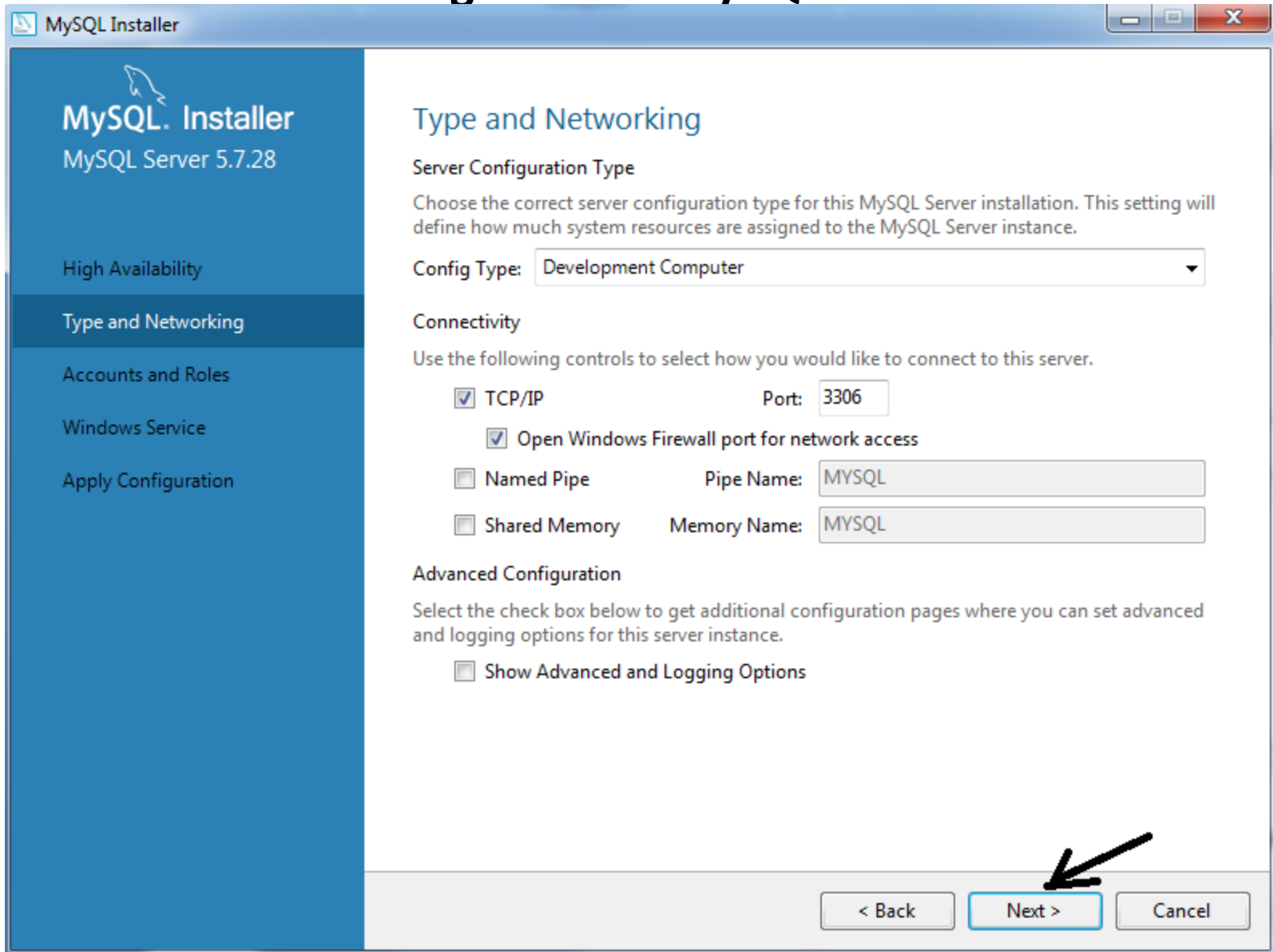
This option enables you to test an InnoDB cluster setup on your local computer using several MySQL Server sandbox instances. Read about [Sandbox Deployment of InnoDB Cluster](#) to learn more.

The diagram illustrates the InnoDB Cluster architecture. On the left, a 'Client App' connects to a 'MySQL Router'. The 'MySQL Router' connects to a 'MySQL Shell'. The 'MySQL Shell' connects to an 'InnoDB Cluster', which is represented by three database nodes within a dashed circle. Arrows indicate the flow of data and control between these components.

To setup and configure a real-world production InnoDB cluster use the Create a New InnoDB Cluster or Add Local MySQL Server Instance to an InnoDB Cluster options available for MySQL Server 8.0.

InnoDB Cluster Log Verbosity Level
The log level for InnoDB cluster configuration.

Configurazione MySQL Server



Configurazione accounts e ruoli

The image shows the MySQL Installer window at the 'Accounts and Roles' step. The main window has a sidebar with options: High Availability, Type and Networking, Accounts and Roles (selected), and Windows Service. The main content area is titled 'Accounts and Roles' and contains the following text: 'Root Account Password' (underlined), 'Enter the password for the root account. Please remember to store this password in a secure place.' Below this are two password input fields. The first is labeled 'MySQL Root Password:' and the second is labeled 'Repeat Password:'. Both fields contain the password 'roberto'. Below the second field, it says 'Password strength: Weak'. There are two arrows pointing to the password fields.

Overlaid on top of the main window is a smaller dialog box titled 'MySQL User Account'. It contains the text: 'Please specify the user name, password, and database role.' Below this is a MySQL logo with a padlock icon. The dialog has three input fields: 'User Name:' with 'admin', 'Host:' with '<All Hosts (%)>', and 'Role:' with 'DB Admin'. Below these is the 'Authentication:' section with 'MySQL' selected. Underneath is a section for 'MySQL user credentials' with two password input fields: 'Password:' and 'Confirm Password:', both containing 'roberto'. Below these fields, it says 'Password strength: Weak'. There are two arrows pointing to the password fields. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

In the background, behind the dialog box, a table is visible with columns 'User' and 'User Role'. To the right of the table are three buttons: 'Add User', 'Edit User', and 'Delete'. An arrow points to the 'Add User' button. At the bottom of the main window are three buttons: '< Back', 'Next >', and 'Cancel'.

Configurazione accounts e ruoli

MySQL Installer

MySQL Server 5.7.28

High Availability

Type and Networking

Accounts and Roles

Windows Service

Apply 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:

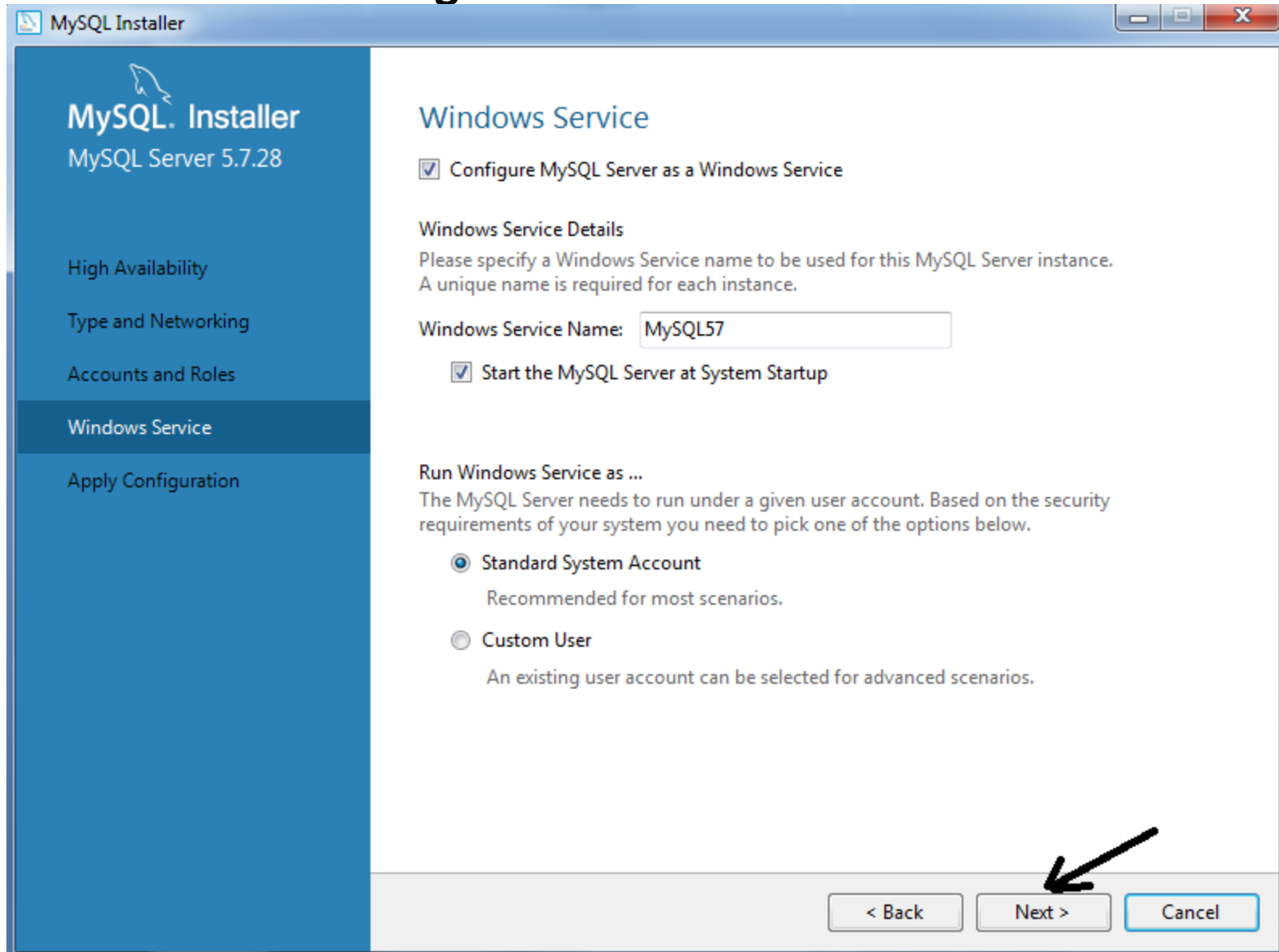
Repeat Password:

Password strength: **Weak**

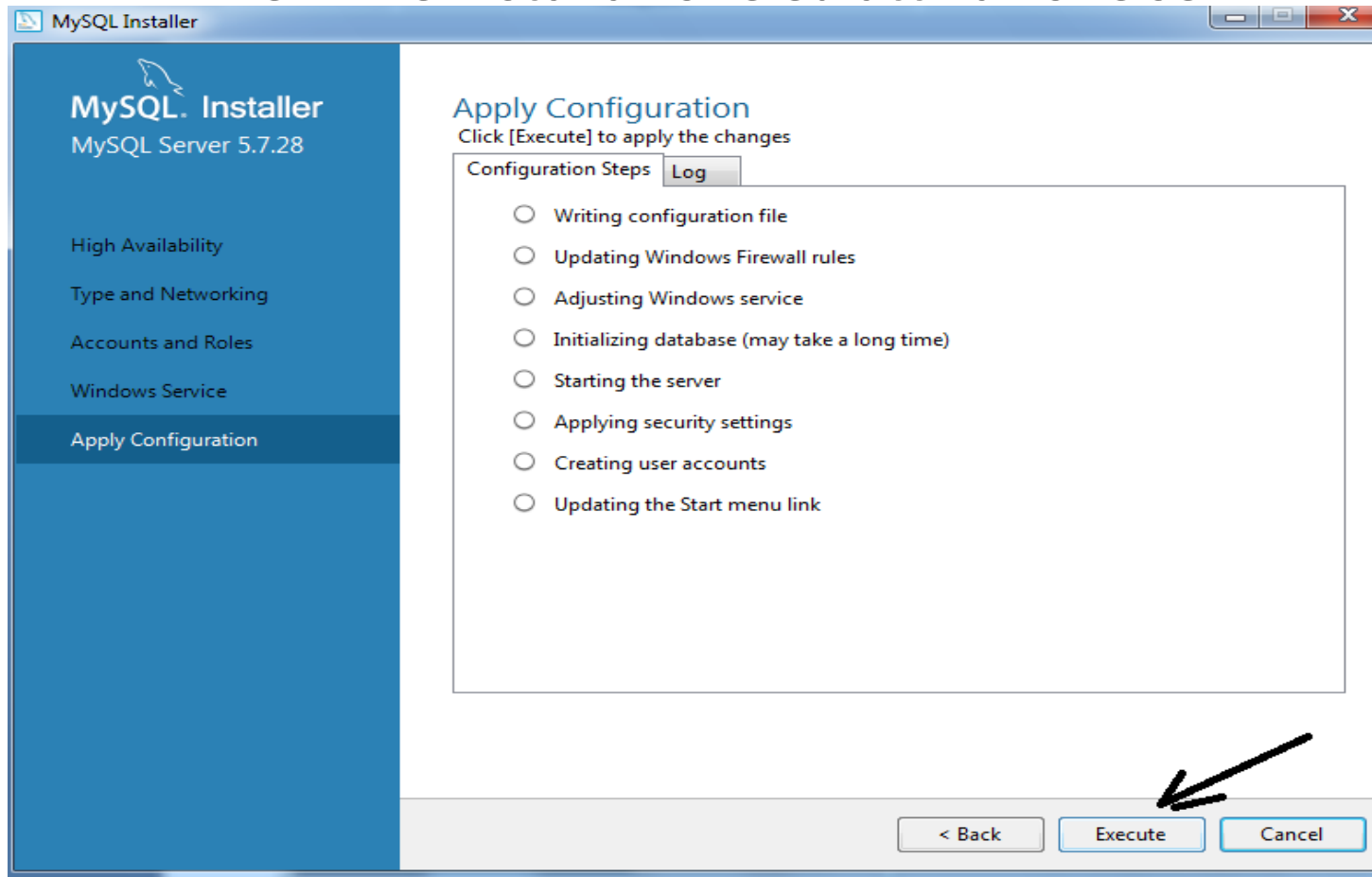
MySQL User Accounts
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

| MySQL User Name | Host | User Role | |
|---|------|-----------|--|
|  admin | % | DB Admin | |

Configurazione accounts e ruoli

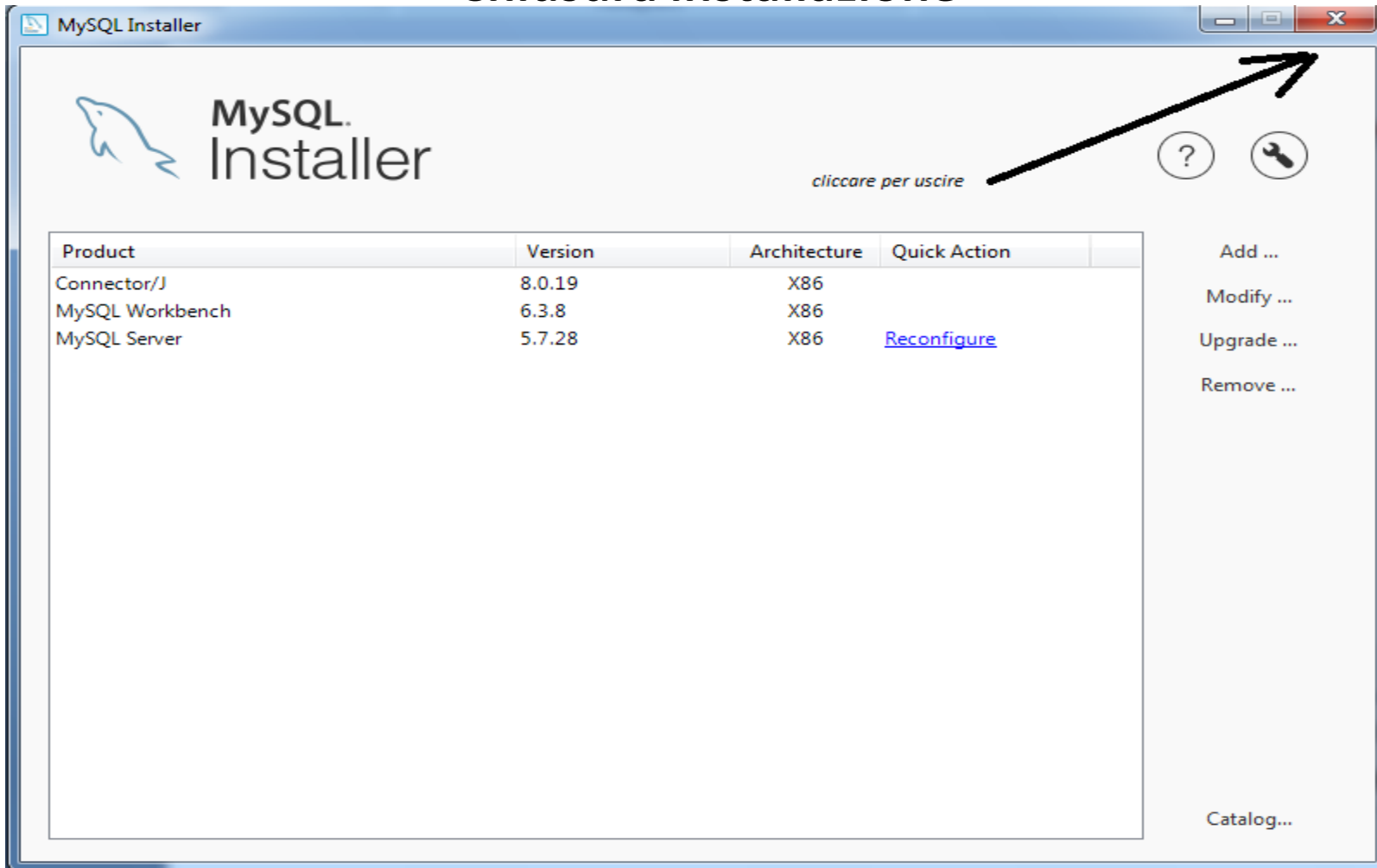


Termine installazione ed attivazione servizi



Terminata l'installazione, chiudere tutto ed iniziamo subito a gestire il primo progetto di database tramite l'applicazione WorkBench.

Chiusura installazione



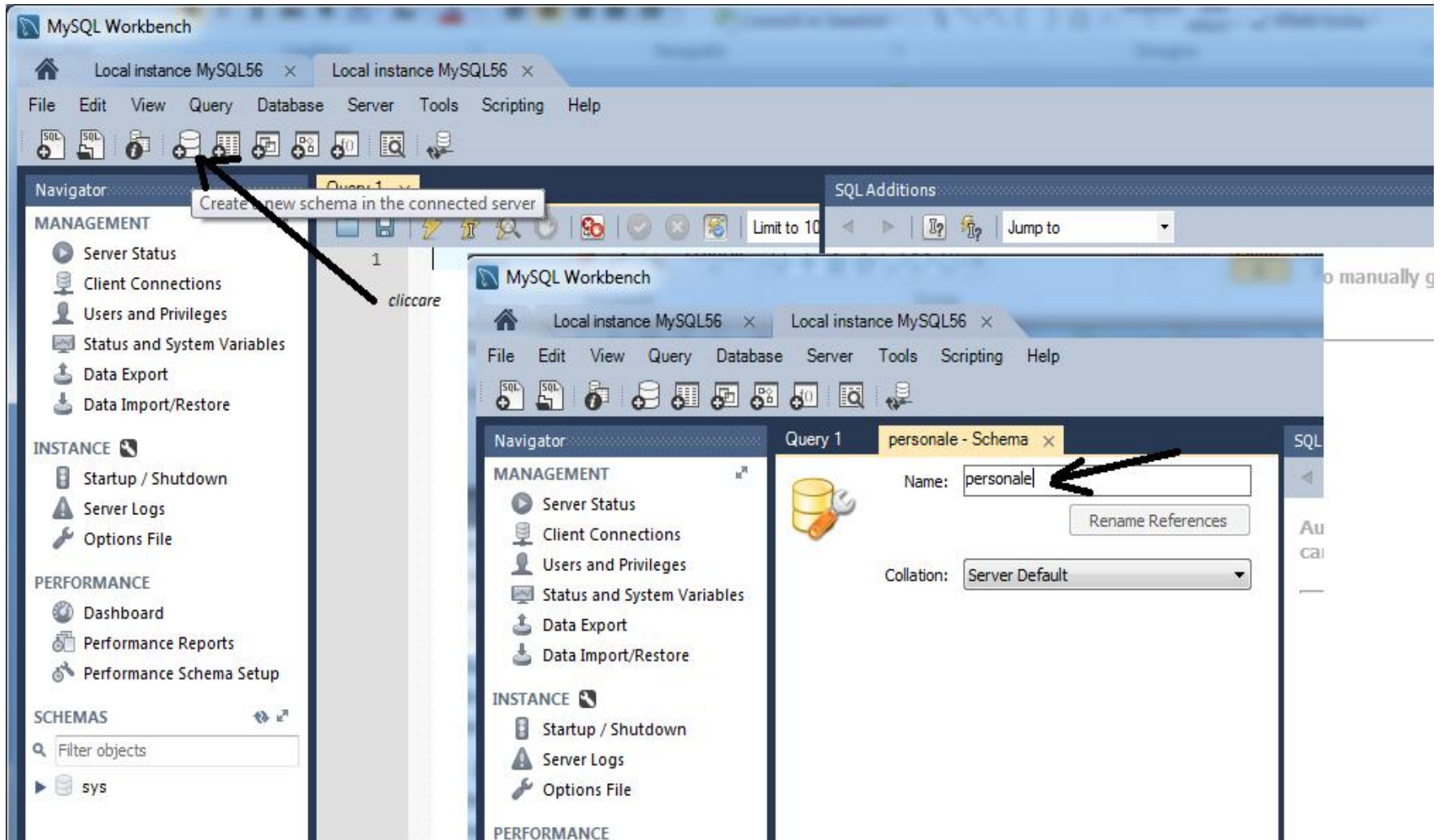
Terminata l'installazione, chiudere tutto ed iniziamo subito a gestire il primo progetto di database tramite l'applicazione WorkBench.

Operazioni con MySQL Workbench

The screenshot displays the MySQL Workbench application window. The main interface is dark-themed and shows a list of MySQL connections under the heading "MySQL Connections". A connection named "Local instance MySQL56" is highlighted, with a black arrow pointing to it and the word "cliccare" (click) written next to it. Below this, a "Connect to MySQL Server" dialog box is open. The dialog box contains the following text: "Please enter password for the following service:", "Service: Mysql@localhost:3306", "User: root", and "Password: *****". A black arrow points to the password input field. Below the password field, there is a checkbox labeled "Save password in vault" and two buttons: "OK" and "Cancel". On the right side of the main interface, there is a "Shortcuts" section with several icons and labels: "MySQL Utilities", "Database Migration", "MySQL Bug Reporter", "Workbench Blogs", "Planet MySQL", "Workbench Forum", and "Scripting Shell".

Operazioni con MySQL Workbench

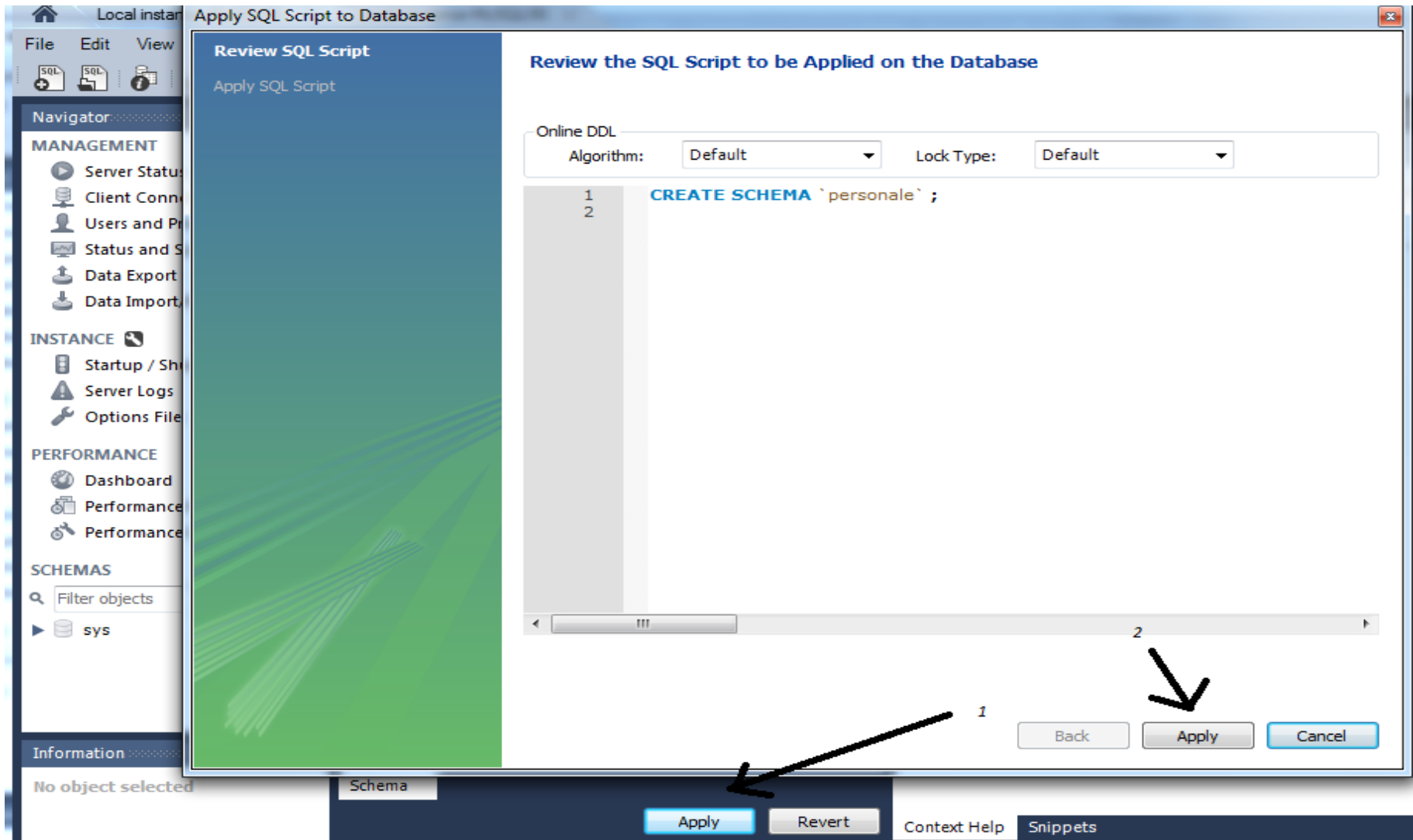
Costruiamo il primo database di nome **personale**, con username **root** e password **roberto**.



Connessione al database - linguaggio Java

Operazioni con MySQL Workbench

Infine selezionare **Apply (1)** e verrà visualizzata lo script corrispondente in SQL. Quindi cliccare ancora su **Apply (2)** per confermare.



Creazione tabella con MySQL Workbench

A questo punto creiamo la prima tabella di nome anagrafica con i vari attributi riportati nella figura sotto.

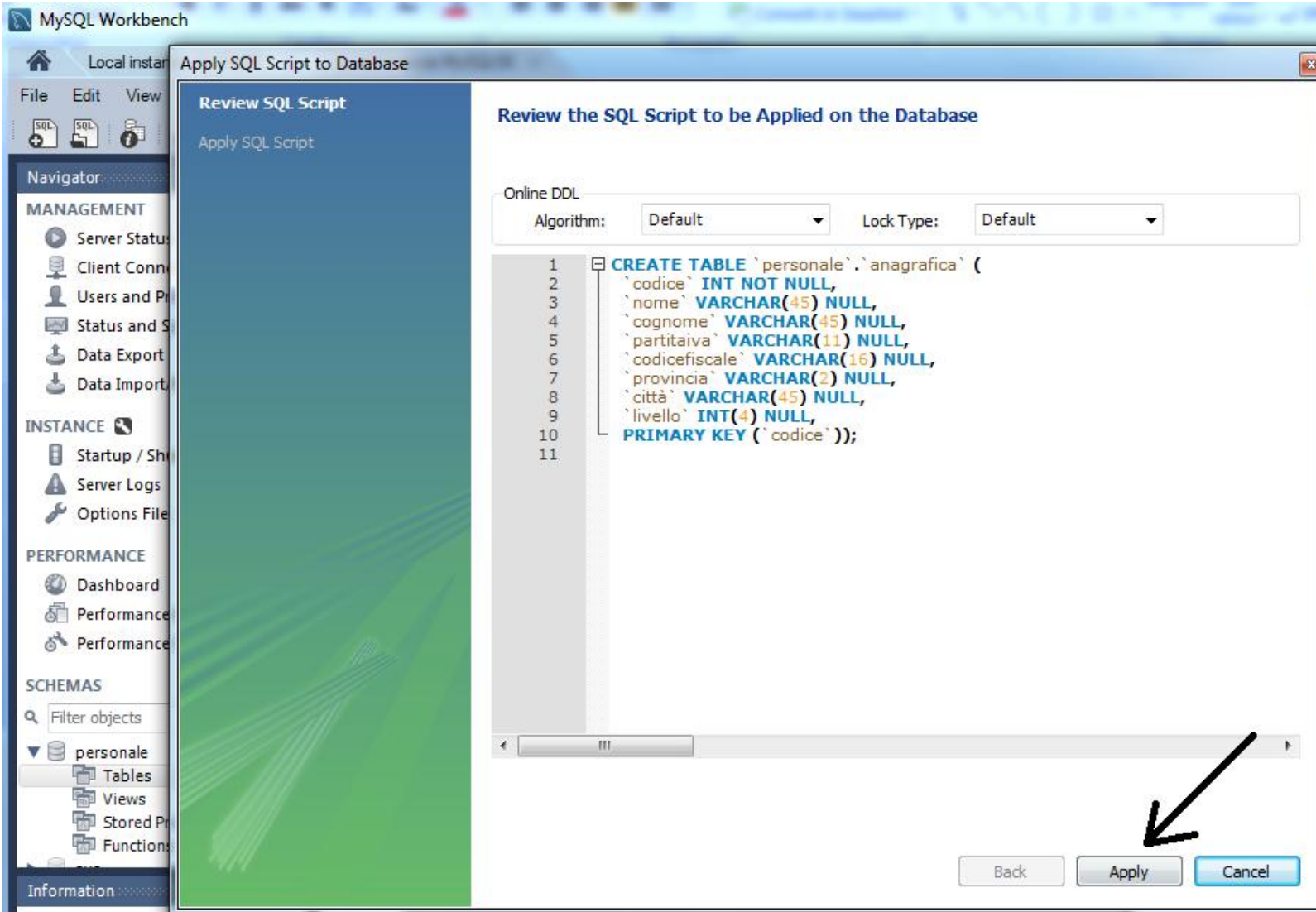
The screenshot shows the MySQL Workbench interface for creating a table named 'anagrafica' in the 'personale' schema. The 'Columns' tab is active, displaying a list of columns with their datatypes and options. The 'codice' column is highlighted as the primary key. The 'Apply' button is visible at the bottom right.

| Datatype | PK | NN | UQ | B | UN | ZF | AI | G | Default/Expression |
|-------------|-------------------------------------|-------------------------------------|-------------------------------------|--------------------------|--------------------------|--------------------------|-------------------------------------|--------------------------|--------------------|
| INT(11) | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |
| VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| VARCHAR(11) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| VARCHAR(16) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| VARCHAR(2) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |
| VARCHAR(45) | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL |

Connessione al database - linguaggio Java

Visualizzazione query di creazione tabella

Verrà visualizzata il nuovo script SQL e selezioniamo Apply per terminare.



Connessione al database - linguaggio Java

MySQL Workbench – Costruire uno schema E/R

Tramite MySQL Workbench è possibile implementare il diagramma E/R

The screenshot displays the MySQL Workbench interface for creating an Entity-Relationship (E/R) diagram. The main workspace shows three tables: 'studente', 'corsi', and 'anagrafica'. The 'studente' table has attributes: Codice INT(11), Nome CHAR(50), Cognome CHAR(50), DataNascita DATE, LuogoNascita CHAR(50), Sesso CHAR(1), Corso INT(11), and corsi_Codice INT(11). The 'corsi' table has attributes: Codice INT(11), Descrizionecorso CHAR(50), and Sede CHAR(50). The 'anagrafica' table has attributes: codice INT(11), nome VARCHAR(45), cognome VARCHAR(45), and partiva VARCHAR(11). A relationship named 'fk_studente_corsi' is defined between the 'corsi_Codice' attribute of the 'studente' table and the 'Codice' attribute of the 'corsi' table. The relationship is shown as a line connecting the two attributes, with a crow's foot notation on the 'studente' side and a one-to-one notation on the 'corsi' side. The 'Modeling Additions' panel on the right lists 'timestamps', 'user', and 'category' as available additions. The 'Catalog Tree' on the left shows the database structure, including 'mydb', 'personale', and 'anagrafica'. The 'Description Editor' at the bottom shows the relationship details, including the caption 'fk_studente_corsi', the relationship definition, and visibility settings.

MySQL Workbench

Local instance MySQL56 (persona.x) MySQL Model* x EER Diagram1 x

File Edit View Arrange Model Database Tools Scripting Help

Bird's Eye

Zoom: 100%

Diagram

Modeling Additions

- timestamps
create_time, update_time
- user
username, email, password, create_time
- category
category_id, name

studente

- Codice INT(11)
- Nome CHAR(50)
- Cognome CHAR(50)
- DataNascita DATE
- LuogoNascita CHAR(50)
- Sesso CHAR(1)
- Corso INT(11)
- corsi_Codice INT(11)

corsi

- Codice INT(11)
- Descrizionecorso CHAR(50)
- Sede CHAR(50)

anagrafica

- codice INT(11)
- nome VARCHAR(45)
- cognome VARCHAR(45)
- partiva VARCHAR(11)

personale - Schema Relationship x

Caption: fk_studente_corsi 'studente' (fk_studente_corsi) 'corsi'

2nd Capt.: 'corsi' () 'studente'

Comments:

Visibility Settings

- Fully Visible
- Draw Split
- Hide

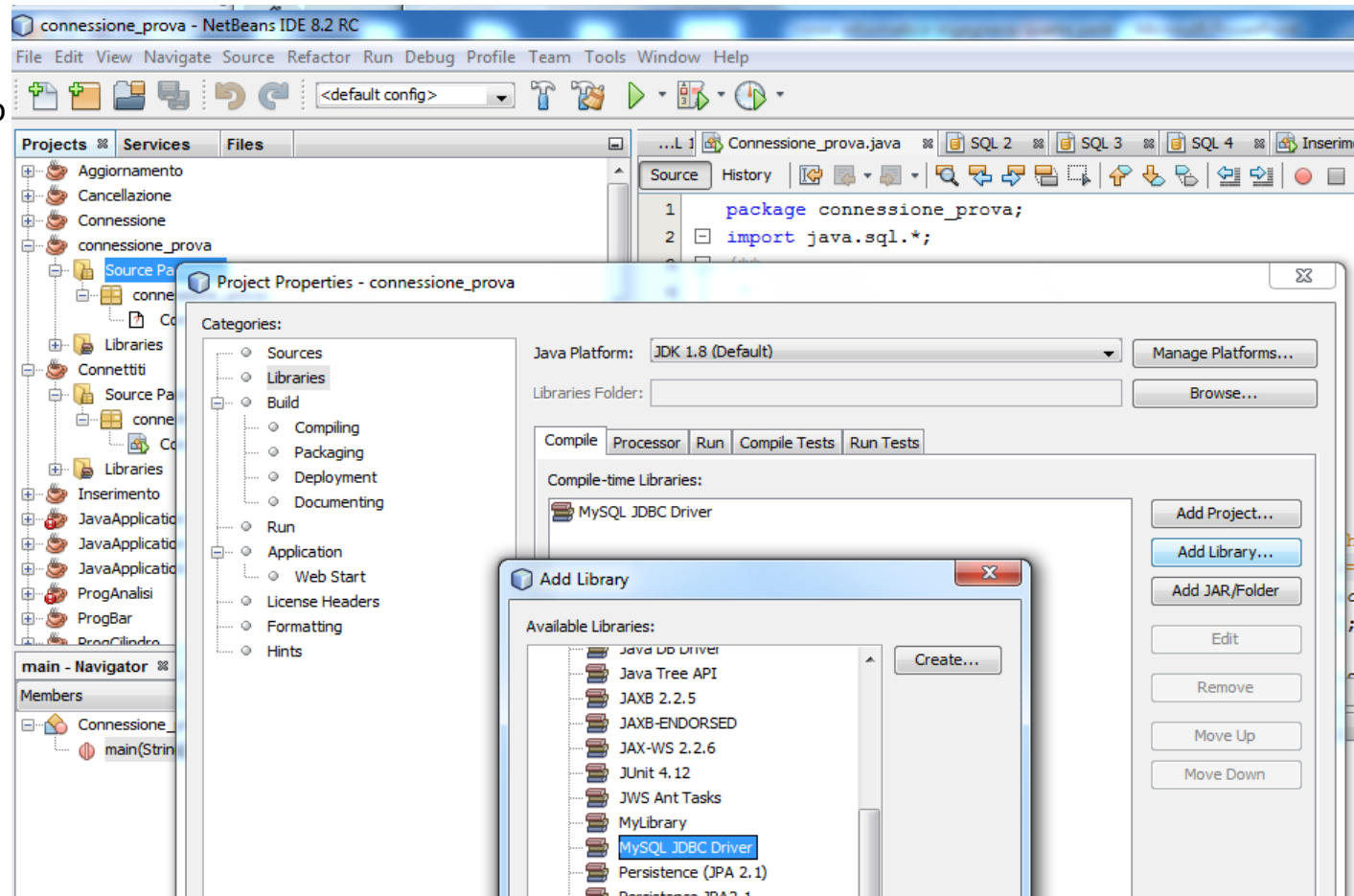
L'accesso ai database in rete - varie applicazioni

Prima di passare al programma in Java, illustriamo alcune istruzioni utili per una corretta connessione con il database **personale** (creato precedentemente).

Il metodo **Class.forName** ricerca la classe **com.mysql.jdbc.Driver** necessaria per il driver jdbc:

Class.forName("com.mysql.jdbc.Driver");

Il metodo **Class.forName** potrà essere riconosciuto solo se aggiungiamo la libreria **MySQL JDBC Driver** nel progetto di NetBens. Quindi creiamo il nuovo programma in Java **Connessione_prova.java** e aggiungiamo la libreria MySQL JDBC Driver.



L'accesso ai database in rete - varie applicazioni

La seconda istruzione necessaria è la dichiarazione dell'oggetto **con** di tipo **Connection**, il metodo **getConnection** dell'interfaccia **DriverManager** attiva la connessione con la risorsa identificata nell'URL. L'URL in JDBC è formato dalla seguente struttura:

protocollo:sottoprotocollo:nomedatabase

- sostituiamo *protocollo* con **jdbc**,
- sostituiamo sottoprotocollo con **mysql**,
- il nomedatabase è formato **dall'indirizzoIP**, il numero della **porta**, il **nome del database** e **credenziali** di accesso ed altri parametri.

Si precisa che il **sottoprotocollo** è il vero driver specifico di quel database, quindi la sintassi di **nomedatabase** dipende dal tipo di sottoprotocollo.

Nel nostro esempio usiamo il sottoprotocollo specifico di MySQL che si chiama appunto **mysql** e per questo tipo di driver si usa nel nomedatabase la seguente sintassi:

indirizzoIP:numeroporta/nome_istanza_database?user=nome_utente&password=password

Per comodità inseriamo in una stringa di nome **Url** il protocollo, sottoprotocollo e nome database così come sotto riportato:

```
String Url = "jdbc:mysql://localhost/personale?" +  
            "user=root&password=roberto";
```

A seguire va indicato l'oggetto **con** di tipo *Connection* per attivare la connessione con la risorsa identificata nella Stringa **Url**.

```
Connection con = DriverManager.getConnection(Url);
```

Riportiamo nella prossima pagina la classe che consente di verificare la connessione al database **personale**.

L'accesso ai database in rete - Connessione_prova.java

```
package connessione_prova;
import java.sql.*;
public class Connessione_prova {
    public static void main(String[] args) {
        // TODO code application logic here
        try {
            Class.forName("com.mysql.jdbc.Driver");
            String Url = "jdbc:mysql://localhost/personale?" +
                "user=root&password=roberto";
            Connection con = DriverManager.getConnection(Url);
            System.out.println("connessione riuscita");
        }
        catch (SQLException e)
        {
            System.out.println("SQL Exception: "+ e.toString());
        }
        catch (ClassNotFoundException cE)
        {
            System.out.println("Class Not Found Exception: "+ cE.toString());
        }
    }
}
```

L'accesso ai database in rete

Una volta verificata la connessione, si passa all'inserimento dei record all'interno della tabella **anagrafica**.

Dopo aver creato, tramite MySQL Workbench, il database personale e la tabella anagrafica, andiamo ad implementare il comando SQL per inserire un nuovo record all'interno della tabella anagrafica. Inseriamo nella stringa **sql** la query seguente:

```
sql="INSERT INTO Anagrafica "+  
    "(nome,cognome,partitaiva,codicefiscale,provincia,città,livello) "+  
    "VALUES ('Roberto', 'Bianchi', '01212342343', 'BNCRBT67A26C034I', 'RM', 'Roma', 1)";
```

Nella prossima pagina è riportato il programma per aggiungere una nuova riga all'interno della tabella anagrafica (file Aggiungi_righe.java).

L'accesso ai database in rete – Aggiungi_righe.java

```
package aggiungi_righe;
import java.sql.*;
public class Aggiungi_righe
{
    public static void main(String[] args)
    {
        Connection con=null;
        Statement st=null;
        String sql;
        String Url="jdbc:mysql://localhost:3306/personale?user=root&password=roberto";
        sql="INSERT INTO Anagrafica "+
            " (nome,cognome,partitaiva,codicefiscale,provincia,città,livello) "+
            " VALUES ('Roberto', 'Bianchi', '01212342343', 'BNCRBT67A26C034I', 'RM', 'Roma', 1)";
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            con=DriverManager.getConnection(Url);
            st=con.createStatement();
            st.executeUpdate(sql);
            System.out.println("RECORD REGISTRATO CORRETTAMENTE");
        }
        catch (ClassNotFoundException | SQLException e)
        {
            System.out.println("SQLException:");
            System.out.println(e.getMessage());
        }
        finally
        {
            if (st !=null)
            { //metodo close per l'oggetto st:rilascia le risorse
              //usate dal comando SQL
              try {st.close();} catch(SQLException e){}
            }
            if (con!=null)
            { //metodo close per l'oggetto conn: interrompe
              //la connessione al database
              try {con.close();} catch(SQLException e){}
            }
        }
    }
}
```

L'accesso ai database in rete- stampa record di una tabella

Implementiamo il file che permette di stampare tutti i record della tabella anagrafica.

Vediamo le istruzioni più importanti:

Si dichiara la **Url** = "jdbc:mysql://localhost:3306/personale?user=root&password=roberto";

Il metodo **Class.forName** ricerca la classe **com.mysql.jdbc.Driver**, quindi viene registrata come driver nell'interfaccia **DriverManager**

```
Class.forName("com.mysql.jdbc.Driver");
```

Si dichiara un oggetto **con** di tipo **Connection**

```
Connection con=null;
```

Il metodo **getConnection** dell'interfaccia **DriverManager** attiva la connessione con la risorsa identificata dell'Url

```
con=DriverManager.getConnection(Url);
```

Dopo la connessione **con** aperta, si crea l'oggetto **st** di tipo **Statement** tramite il metodo **createStatement**

```
Statement st = con.createStatement();
```

Tramite il metodo **executeQuery**, si esegue la query per leggere tutte le tuple della tabella anagrafica, il risultato della query verrà salvato nell'oggetto **rs** di tipo **ResultSet**

```
ResultSet rs = st.executeQuery("SELECT * from Anagrafica");
```

Viene usato il metodo **next** dell'oggetto **rs** per muovere il puntatore alla riga successiva del **resultset**, restituisce quindi un valore falso se ci troviamo alla fine dell'ultima riga.

Con il metodo **getString()** preleviamo i valori contenuti nei campi della riga del **resultset** corrente e, tramite la **System.out.println()**, stampiamo il risultato.

L'accesso ai database in rete- stampa record di una tabella

Infine, tramite il costrutto while, stampiamo tutte le righe della tabella anagrafica:

```
while (rs.next()) {  
    System.out.println("CODICE:"+rs.getString("codice"));  
    System.out.println("NOME:"+rs.getString("nome"));  
    System.out.println("COGNOME:"+rs.getString("cognome"));  
    System.out.println("PARTITA IVA:"+rs.getString("partitaiva"));  
    System.out.println("CODICE FISCLAE:"+rs.getString("codicefiscale"));  
    System.out.println("PROVINCIA:"+rs.getString("provincia"));  
    System.out.println("CITTA':" +rs.getString("città"));  
    System.out.println("LIVELLO:"+rs.getString("livello"));  
    System.out.println("_____");  
}
```

Riportiamo il programma **Prova_conn.java** nella pagina successiva.

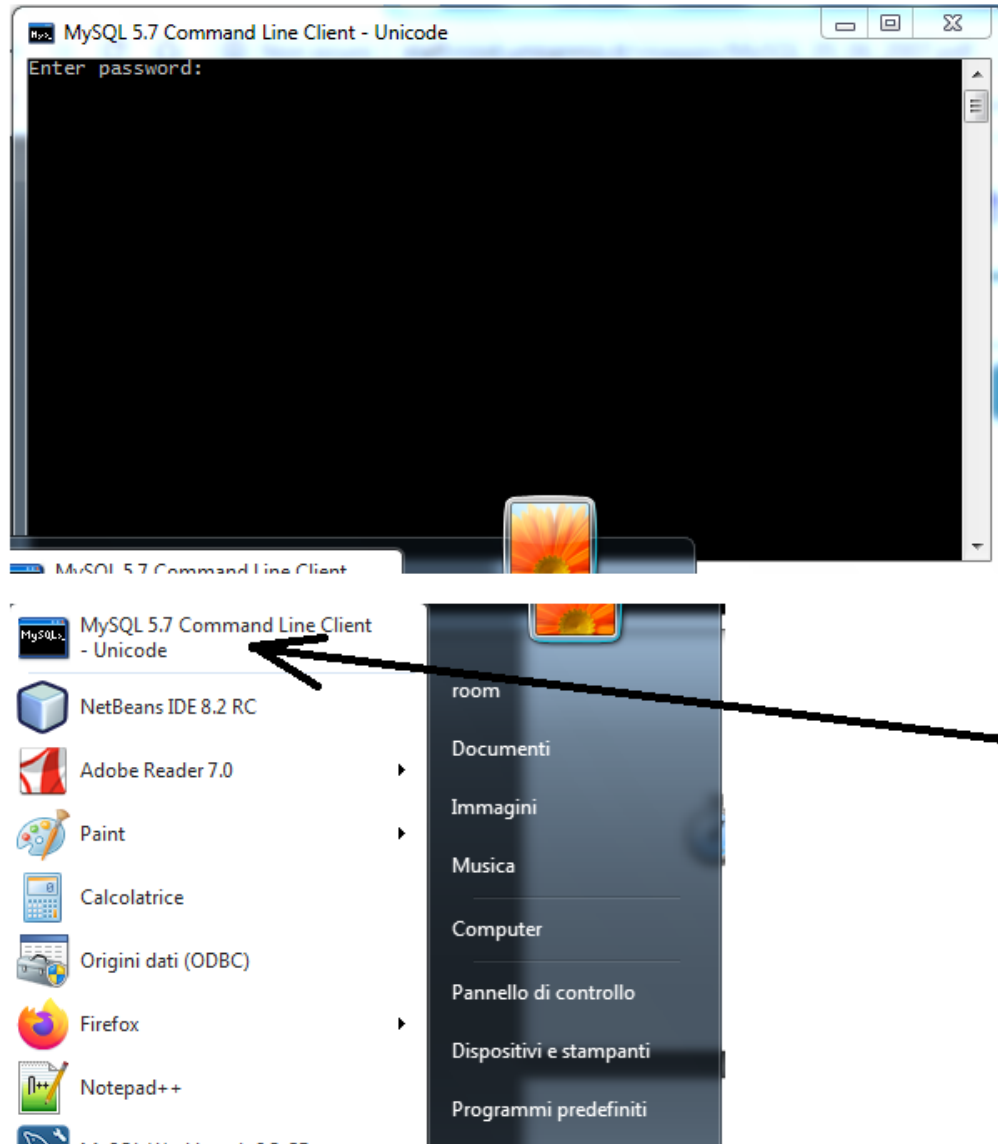
L'accesso ai database in rete – Prova_conn.java

```
package prova_conn;
import java.sql.*;
class Prova_conn {
    public static void main(String[] Args) {
        String Url="jdbc:mysql://localhost:3306/personale?user=root&password=roberto";
        try {
            Class.forName("com.mysql.jdbc.Driver");
        }
        catch (Exception E) {
            System.err.println("Non trovo il driver ");
            E.printStackTrace();
        }
        try {
            Connection con=null;
            con=DriverManager.getConnection(Url);
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery("SELECT * from Anagrafica");
            while (rs.next()) {
                System.out.println("CODICE:"+rs.getString("codice"));
                System.out.println("NOME:"+rs.getString("nome"));
                System.out.println("COGNOME:"+rs.getString("cognome"));
                System.out.println("PARTITA IVA:"+rs.getString("partitaiva"));
                System.out.println("CODICE FISCLAE:"+rs.getString("codicefiscale"));
                System.out.println("PROVINCIA:"+rs.getString("provincia"));
                System.out.println("CITTA':" +rs.getString("città"));
                System.out.println("LIVELLO:"+rs.getString("livello"));
                System.out.println("_____");
            }
            rs.close();//Il resultset rs viene chiuso.
            st.close();//l'oggetto st viene deallocato
            con.close();// a questo punto viene chiusa la connessione al database
        }
        catch (SQLException E) {
            System.out.println("SQLException: " + E.getMessage());
            System.out.println("SQLState: " + E.getSQLState());
            System.out.println("VendorError: " + E.getErrorCode());
        }
    }
}
```

Connessione al database - linguaggio Java

MySQL – Principali comandi

Riportiamo di seguito i comandi principali per interagire con un server MySQL. Prima di tutto apriamo il Command Line Client e vediamo di interagire con i comandi principali.



MySQL– Principali comandi

Premesso che l'utente base del database è root e che il servizio server è chiamato MySQL

Per eseguire la connessione con il server, bisogna inserire login e password

Quindi l'istruzione completa è:

```
shell>mysql -h host -u user -p
```

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

Dove **host** e **user** rappresentano rispettivamente il nome dell'host dove risiede MySQL ed user è lo username dell'utente che possiede un account server; -p specifica al server la richiesta della password dell'utente.

Il comando completo è:**C:\>mysql -u nomeuser -p**

dove al posto di nomeuser si può inserire root e come password, nel nostro caso, si può digitare roberto.

Se l'inserimento è corretto, verrà visualizzato

Il messaggio di benvenuto

“Welcome to the MySQL monitor...”

Se digitiamo il comando \h, **help**, verrà

visualizzato un elenco di comandi utili

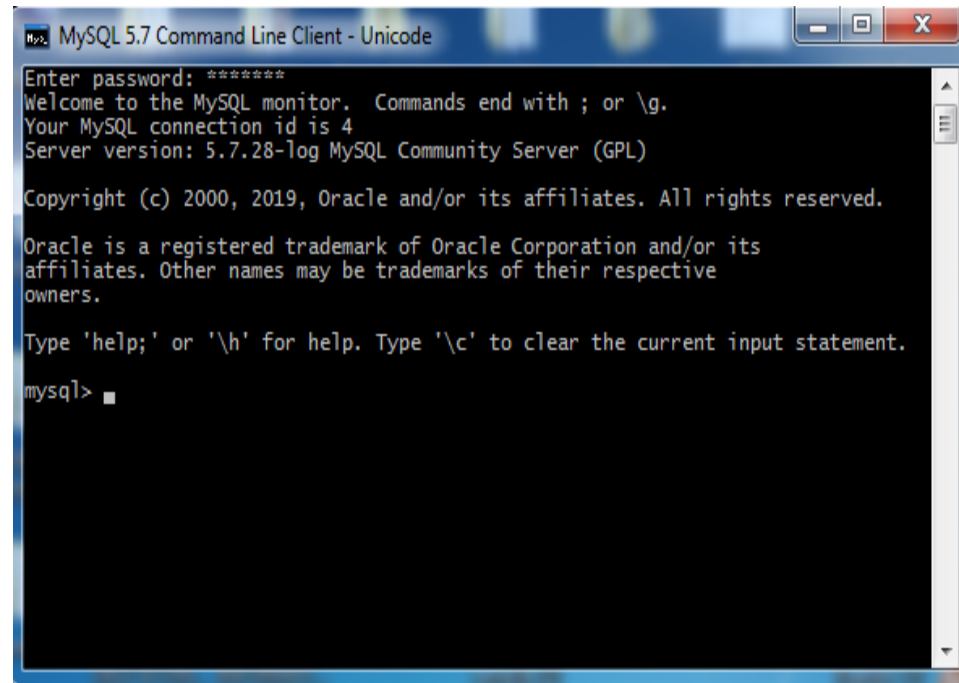
per gestire gli utenti, il database ed altro

ancora. E' possibile chiedere un aiuto per il

comando da digitare, la sintassi è:

\h nomecomando, ad esempio:**\h use**

Si può scrivere anche **? use** Connessione al database - linguaggio Java



MySQL – Principali comandi

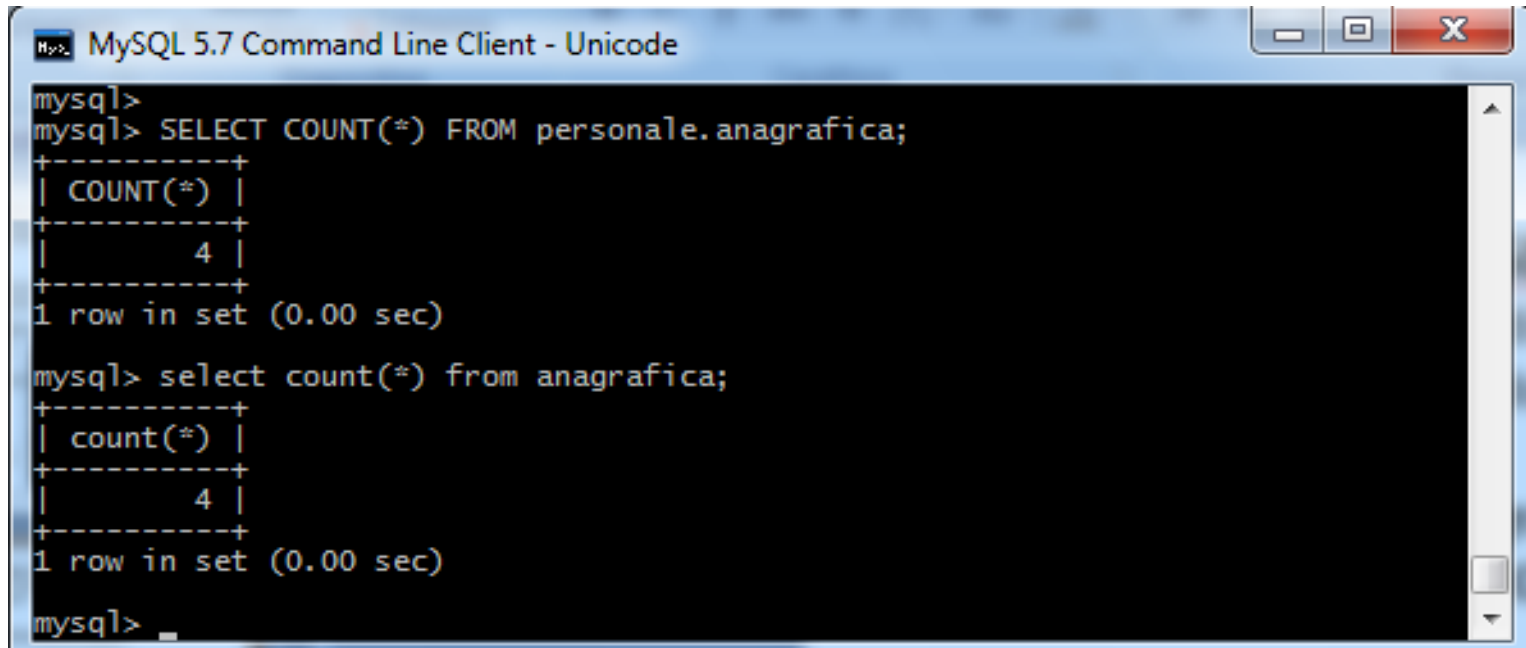
Si può chiedere di contare le righe della tabella anagrafica appartenente al database personale.

Il comando è il seguente:

use personale;

SELECT COUNT(*) FROM anagrafica;

Si può anche scrivere: **SELECT COUNT(*) FROM personale.anagrafica;**



```
MySQL 5.7 Command Line Client - Unicode
mysql>
mysql> SELECT COUNT(*) FROM personale.anagrafica;
+-----+
| COUNT(*) |
+-----+
|         4 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from anagrafica;
+-----+
| count(*) |
+-----+
|         4 |
+-----+
1 row in set (0.00 sec)

mysql> _
```

MySQL – Principali comandi

Vediamo ora come è possibile eseguire operazioni di aggiunta nuovo utente, modifica password ed altre operazioni sugli account. Descriviamo prima di tutto le varie operazioni da implementare:

- 1) Creare un utente antonio senza password
- 2) Creare un utente marco con password 'ciao'
- 3) Eliminare l'utente antonio
- 4) Impostare la password 'robby' per l'utente collegato (attualmente è roberto)
- 5) Impostare la password 'prova' per l'utente marco@localhost

Quindi le istruzioni sono:

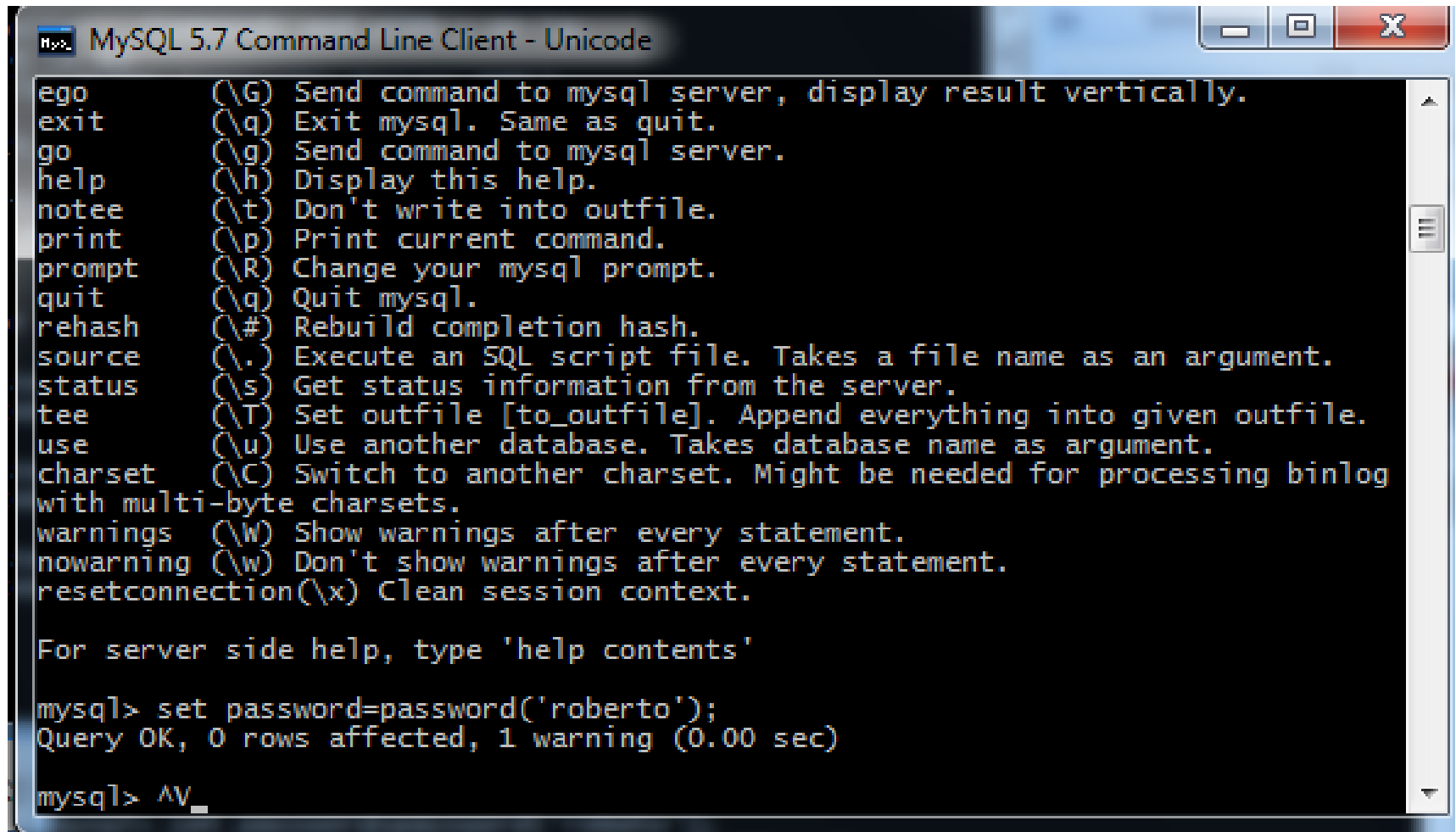
- 1) `CREATE USER antonio@localhost;`
- 2) `CREATE USER marco@localhost;`
- 3) `DROP USER antonio@localhost;`
- 4) `SET PASSWORD=PASSWORD('robby');`
- 5) `SET PASSWORD FOR marco@localhost =PASSWORD('prova');`

Al termine avremo l'utente root con la password roby e l'utente antonio con la password prova, mentre l'utente antonio è stato cancellato dopo la creazione.

Per consentire ai programmi creati precedentemente di funzionare, bisogna reimpostare la password di root con roberto.

Quindi digitiamo nella linea di comando:`SET PASSWORD=PASSWORD('roberto');`

MySQL– Principali comandi



```
MySQL 5.7 Command Line Client - Unicode
ego      (\G) Send command to mysql server, display result vertically.
exit     (\q) Exit mysql. Same as quit.
go       (\g) Send command to mysql server.
help     (\h) Display this help.
notee    (\t) Don't write into outfile.
print    (\p) Print current command.
prompt   (\R) Change your mysql prompt.
quit     (\q) Quit mysql.
rehash   (\#) Rebuild completion hash.
source   (\.) Execute an SQL script file. Takes a file name as an argument.
status   (\s) Get status information from the server.
tee       (\T) Set outfile [to_outfile]. Append everything into given outfile.
use      (\u) Use another database. Takes database name as argument.
charset  (\C) Switch to another charset. Might be needed for processing binlog
with multi-byte charsets.
warnings (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
resetconnection (\x) Clean session context.

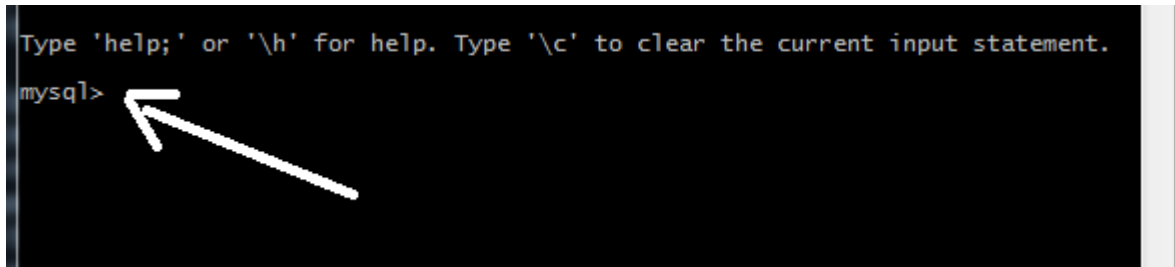
For server side help, type 'help contents'

mysql> set password=password('roberto');
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> ^V_
```

MySQL – Principali comandi

Vediamo di seguito il prompt dei comandi di MySQL:



```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql>
```

mysql> pronto per ricevere un comando

-> si tratta di un comando multiple-line, quindi è in attesa di un nuovo comando;

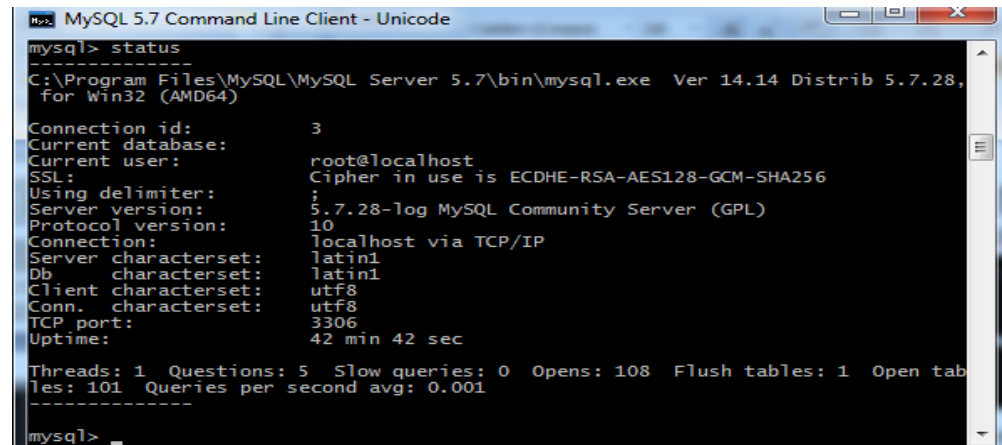
'> si tratta di un comando multiple-line, quindi è in attesa della successiva linea, apice di apertura nella riga precedente (');

"> si tratta di un comando multiple-line, quindi è attesa della successiva linea, doppio apice di apertura nella riga precedente (");

`> **si tratta di un comando** multiple-line e fa riferimento ad un identificatore che inizia con backtick (`).

Il comando **status** visualizza la cartella dove è installato MySQL, lo user corrente ed altre informazioni.

Il comando **quit** esce dal client MySQL



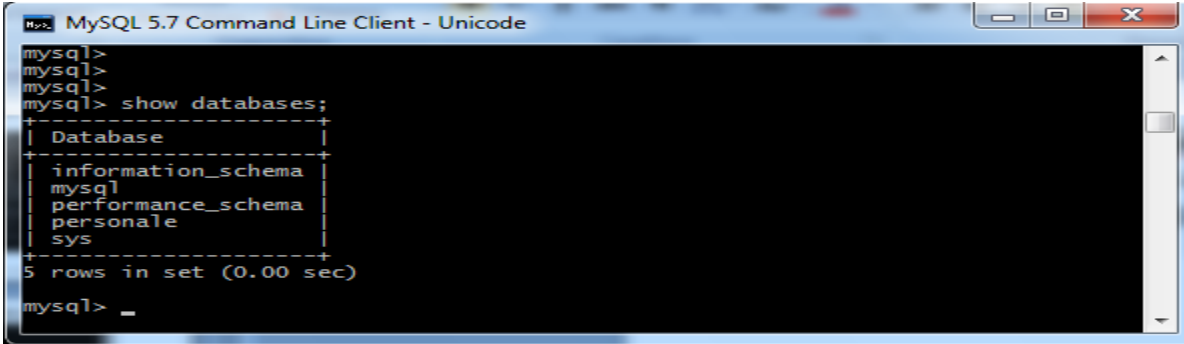
```
mysql> status  
-----  
C:\Program Files\MySQL\MySQL Server 5.7\bin\mysql.exe Ver 14.14 Distrib 5.7.28,  
for Win32 (AMD64)  
  
Connection id:          3  
Current database:        
Current user:          root@localhost  
SSL:                   Cipher in use is ECDHE-RSA-AES128-GCM-SHA256  
;  
Using delimiter:      ;  
Server version:        5.7.28-log MySQL Community Server (GPL)  
Protocol version:      10  
Connection:           localhost via TCP/IP  
Server characterset:   latin1  
Db characterset:       latin1  
Client characterset:   utf8  
Conn. characterset:    utf8  
TCP port:              3306  
Uptime:                42 min 42 sec  
  
Threads: 1 Questions: 5 Slow queries: 0 Opens: 108 Flush tables: 1 Open tab  
les: 101 Queries per second avg: 0.001  
-----  
mysql>
```

MySQL – Principali comandi

Verifica dei database collegati all'utente:

Per verificare i database collegati all'utente root, digitiamo il comando:

show databases;



```
mysql>
mysql>
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| personale |
| sys |
+-----+
5 rows in set (0.00 sec)

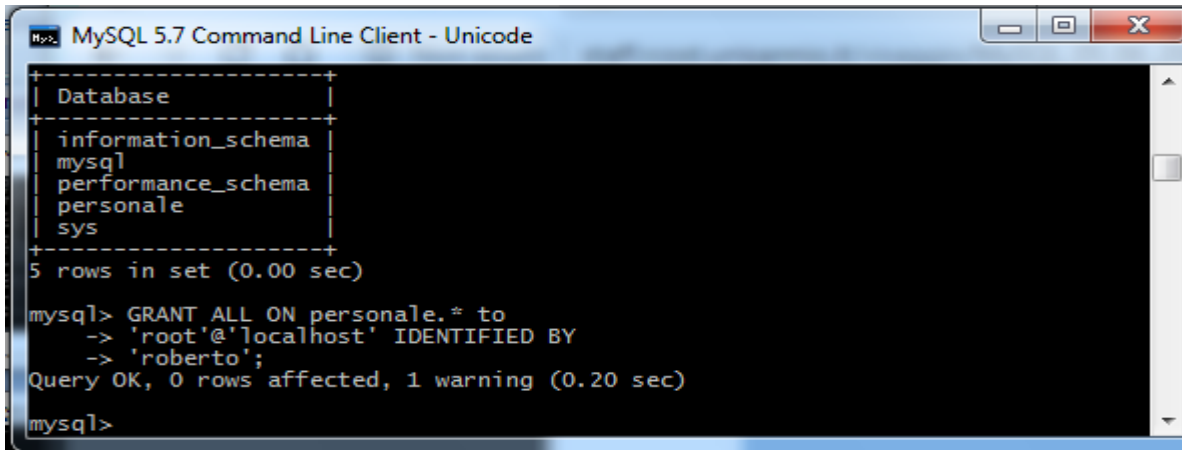
mysql> _
```

Assegnazione di tutti i permessi dell'utente root al database personale.

Il comando è GRANT ALL ON personale.* to

'root'@'localhost' IDENTIFIED BY

'roberto';



```
mysql> GRANT ALL ON personale.* to
-> 'root'@'localhost' IDENTIFIED BY
-> 'roberto';
Query OK, 0 rows affected, 1 warning (0.20 sec)

mysql>
```

MySQL – Principali comandi

Si ricorda che i permessi vengono dati con il comando **GRANT** e posso essere tolti con il comando **REVOKE**.

Con il comando **GRANT ALL ON personale.* to 'root'@'localhost' IDENTIFIED BY 'roberto';**

Diamo tutti i permessi all'utente root sul database personale.

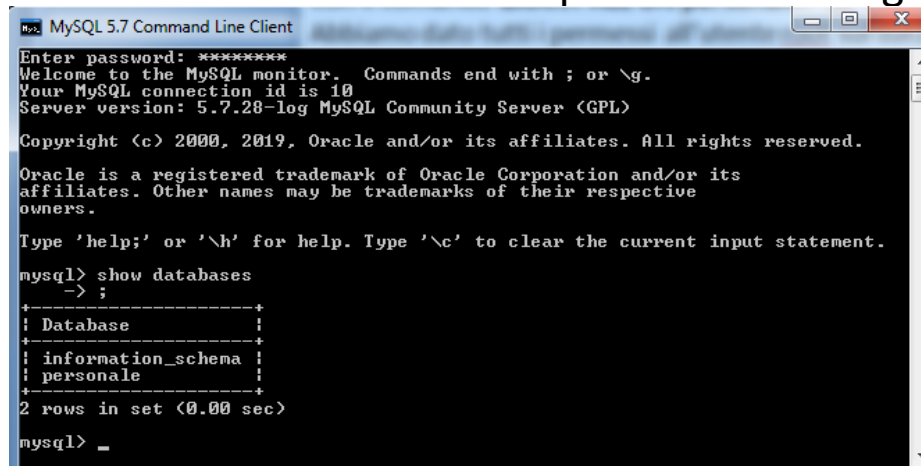
Vediamo alcuni esempi di comandi GRANT e REVOKE:

GRANT SELECT ON personale.* TO marco@localhost IDENTIFIED BY 'password' WITH GRANT OPTION;

Questa istruzione assegna il privilegio SELECT all'utente marco@localhost sul database personale. Se l'utente non esisteva in precedenza, la riga relativa viene aggiunta alla tabella user e 'password' sarà la sua password. Se l'utente esisteva già, la password viene sostituita. Nel nostro caso l'utente marco è stato creato precedentemente con la password prova, quindi ad esso è stato associato il permesso per gestire il database personale e sarà assegnata la nuova password. L'opzione **WITH GRANT OPTION** serve per indicare che l'utente **marco** potrà assegnare ad altri utenti **tutti** i propri permessi: quelli che ha ricevuto con questa istruzione, quelli che già aveva e quelli che riceverà in futuro. Se ora proviamo ad eseguire il comando

show databases;

il risultato sarà:



```
MySQL 5.7 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 5.7.28-log MySQL Community Server (GPL)

Copyright (c) 2000, 2019, 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.

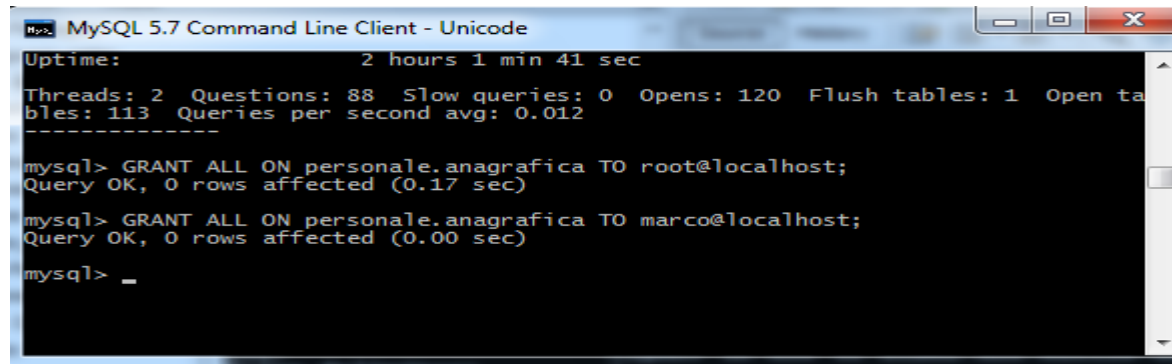
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases
-> ;
+-----+
| Database |
+-----+
| information_schema |
| personale |
+-----+
2 rows in set (0.00 sec)

mysql> _
```

MySQL– Principali comandi

L'istruzione **GRANT ALL ON personale.anagrafica TO marco@localhost;** assegna tutti i permessi sulla tabella **anagrafica**, del database **personale**, all'utente **marco@localhost**. Attenzione questa operazione va fatto sotto l'account root in quanto deve essere l'amministratore a dare le autorizzazioni all'account marco sulla tabella anagrafica. Poiché non è stata specificata nessuna password, allora se l'utente esiste già non verrà cambiata, altrimenti se l'utente non esiste sarà creato senza password.



```
MySQL 5.7 Command Line Client - Unicode
Uptime: 2 hours 1 min 41 sec
Threads: 2 Questions: 88 Slow queries: 0 Opens: 120 Flush tables: 1 Open tables: 113
Queries per second avg: 0.012
-----
mysql> GRANT ALL ON personale.anagrafica TO root@localhost;
Query OK, 0 rows affected (0.17 sec)

mysql> GRANT ALL ON personale.anagrafica TO marco@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> _
```

REVOKE SELECT on personale.* FROM marco@localhost;

Con questa istruzione togliamo il privilegio SELECT sul db personale all'utente marco@localhost.

REVOKE ALL PRIVILEGES, GRANT OPTION FROM marco@localhost;

Con questa istruzione togliamo tutti i privilegi sulle tabelle più quello di GRANT a marco@localhost. Resta inteso che l'utente, anche se privo di privilegi, avrà sempre l'utenza attiva: non viene eliminata dalla tabella user.

Si ricorda che quando si elimina un database o una tabella, tutti i permessi esistenti rimangono attivi. Quindi se si creano nuovi database o tabelle con lo stesso nome, in tal caso, tutti i vecchi permessi rimangono.

MySQL – Principali comandi

Riportiamo nella tabella successiva i principali permessi che si possono assegnare ad un utente relativamente ai database e alle tabelle:

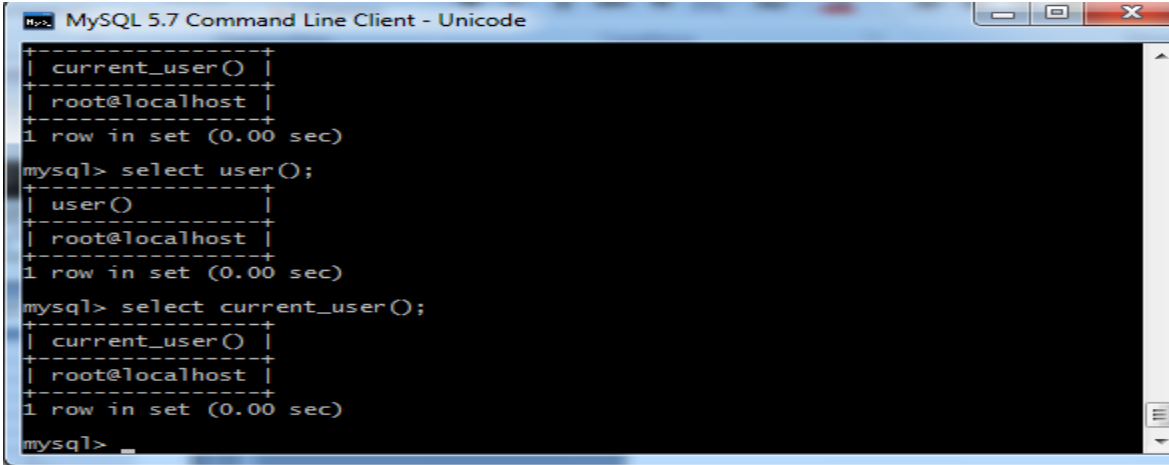
| Sintassi dell'istruzione | Tipo permesso |
|--------------------------|-------------------------|
| tutte escluse GRANT | ALL |
| ALTER TABLE | ALTER |
| CREATE TABLE | CREATE |
| CREATE TEMPORARY TABLE | CREATE TEMPORARY TABLES |
| CREATE VIEW | CREATE VIEW |
| DELETE | DELETE |
| DROP TABLE | DROP |
| CREATE INDEX, DROP INDEX | INDEX |
| INSERT | INSERT |
| LOCK TABLES | LOCK TABLES |
| SELECT | SELECT |
| SHOW CREATE VIEW | SHOW VIEW |
| UPDATE | UPDATE |
| nessuna | USAGE |
| GRANT, REVOKE | GRANT OPTION |

MySQL – Principali comandi

Riportiamo ora due funzioni di MySQL :

SELECT USER(); e la funzione **SELECT CURRENT_USER();**

Con **SELECT USER();** otteniamo il nome dell'utente e dell'host con i quali abbiamo richiesto l'accesso, mentre **CURRENT_USER();** indica quale utente si è autenticato.



```
MySQL 5.7 Command Line Client - Unicode
+-----+
| current_user() |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)

mysql> select user();
+-----+
| user() |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)

mysql> select current_user();
+-----+
| current_user() |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)

mysql>
```

Istruzione per creare un database:

CREATE DATABASE IF NOT EXISTS database_prova;

L'opzione IF NOT EXISTS consente di verificare se il database esiste già.

Istruzione per cancellare un database con tutte le tabelle:

DROP DATABASE IF EXISTS database_prova;

MySQL– Principali comandi

Di seguito vengono descritte le istruzioni per creare due tabelle, corsi e studente, appartenenti al database personale.

Prima di tutto bisogna attivare il database personale, il comando è il seguente:

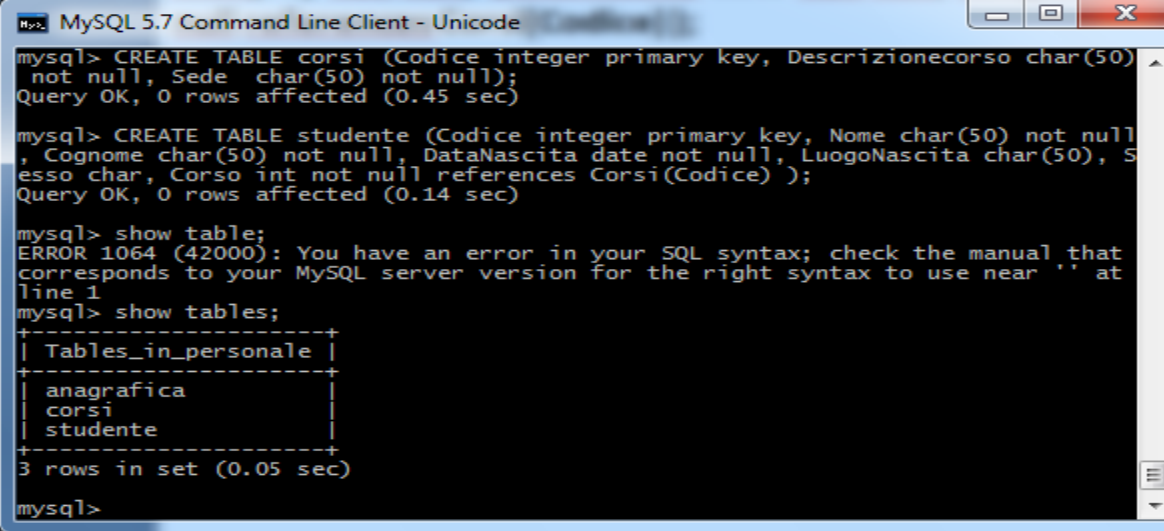
USE personale;

Successivamente andiamo a creare le tabelle;

CREATE TABLE corsi (Codice integer primary key, Descrizionecorso char(50) not null, Sede char(50) not null);

CREATE TABLE studente (Codice integer primary key, Nome char(50) not null, Cognome char(50) not null, DataNascita date not null, LuogoNascita char(50), Sesso char, Corso int not null references Corsi(Codice));

Eseguiamo il comando **show tables;** ed otteniamo il seguente listato:



```
mysql> CREATE TABLE corsi (Codice integer primary key, Descrizionecorso char(50) not null, Sede char(50) not null);
Query OK, 0 rows affected (0.45 sec)

mysql> CREATE TABLE studente (Codice integer primary key, Nome char(50) not null, Cognome char(50) not null, DataNascita date not null, LuogoNascita char(50), Sesso char, Corso int not null references Corsi(Codice) );
Query OK, 0 rows affected (0.14 sec)

mysql> show table;
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 '' at line 1
mysql> show tables;
+-----+
| Tables_in_personale |
+-----+
| anagrafica          |
| corsi               |
| studente            |
+-----+
3 rows in set (0.05 sec)

mysql>
```

MySQL – Principali comandi

Vediamo ora come è possibile inserire all'interno della tabella un nuovo corso.

INSERT INTO corsi (Codice, Descrizionecorso, Sede) VALUES (102, 'Fondamenti di informatica', 'Frosinone');

Per inserire più righe nella tabella, basta aggiungere altre parentesi dopo la clausola values

INSERT INTO corsi (Codice, Descrizionecorso, Sede) VALUES (103, 'Fondamenti di informatica', 'Frosinone'), (104, 'Fisica1', 'Frosinone');

```
Fisical', 'Frosinone')' at line 1
mysql> INSERT INTO corsi (Codice, Descrizionecorso, Sede) VALUES (106, 'Fondame
ti di informatica', 'Frosinone'), (105, 'Fisical', 'Frosinone');
Query OK, 2 rows affected (0.02 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql>
```

Con il comando **SELECT * FROM corsi;** visualizzo il contenuto della tabella

```
mysql> SELECT * FROM corsi;
+----+-----+-----+
| Codice | Descrizionecorso | Sede |
+----+-----+-----+
| 102 | Fondamenti di informatica | Frosinone |
| 103 | Fisica | Frosinone |
| 105 | Fisical | Frosinone |
| 106 | Fondamenti di informatica | Frosinone |
+----+-----+-----+
4 rows in set (0.00 sec)
```

Per cancellare l'ultima riga duplicata, il comando è:

DELETE FROM corsi WHERE codice= 106 LIMIT 1;

Si usa LIMIT 1 per limitare l'eliminazione ad un solo elemento. Si noti che abbiamo ripetuto la **select** per verificare se effettivamente è stato tolta la riga con codice 106

```
mysql> DELETE FROM corsi WHERE codice= 106 LIMIT 1;
Query OK, 1 row affected (0.08 sec)

mysql> DELETE FROM corsi WHERE codice= 106 LIMIT 1;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from corsi;
+----+-----+-----+
| Codice | Descrizionecorso | Sede |
+----+-----+-----+
| 102 | Fondamenti di informatica | Frosinone |
| 103 | Fisica | Frosinone |
| 105 | Fisical | Frosinone |
+----+-----+-----+
3 rows in set (0.00 sec)
```

MySQL – Principali comandi

Eliminare più righe

Per eliminare più righe da una tabella, il comando MySQL è.

```
DELETE FROM corsi WHERE codice > 100 and codice<103 LIMIT 5;
```

Eliminare tutti i records di una tabella

Si possono cancellare tutti i records da una singola tabella con la seguente istruzione:

```
DELETE * FROM corsi;
```

Anche se vengono rimossi tutti i campi, resta comunque la struttura della tabella.

Pertanto, se inseriamo nuovi campi dopo aver cancellato tutte le righe, notiamo che tutti i campi autoincrementali partono dalla vecchia numerazione.

Ad esempio, se avessimo avuto un campo "codice" auto incrementale arrivato fino a 10, con l'istruzione `DELETE * FROM nomeTabella`, i nuovi campi partiranno da 11.

Per evitare ciò, si utilizza il comando `TRUNCATE`, ad esempio:

```
TRUNCATE TABLE nomeTabella;
```

Con l'istruzione `TRUNCATE TABLE` il contenuto della tabella sarà completamente svuotato e i campi auto incrementali ripartiranno da 1: si chiede pertanto di fare molta attenzione quando si usa questo comando.

MySQL – Principali comandi

Comando ALTER TABLE nome_tabella AUTO_INCREMENT = numero;

Il comando **ALTER TABLE nome_tabella AUTO_INCREMENT = numero**, serve proprio per far ripartire da un valore desiderato l'auto incremento della chiave primaria; tale istruzione è utile se si vuole ripartire da una numerazione corretta dopo la cancellazione di record.

Vediamo un esempio sulla tabella anagrafica del database personale.

Eseguiamo il comando **SELECT * FROM anagrafica;**

```
mysql> SELECT * FROM anagrafica;
+----+-----+-----+-----+-----+-----+-----+
| codice | nome   | cognome | partitaiva | codicefiscale | provincia | città |
+----+-----+-----+-----+-----+-----+-----+
| 1 | Roberto | Bianchi | 01212342343 | BNCRBT67A26C034I | RM | Roma |
| 2 | Mario   | Verdi   | 01312342323 | VRDMRO66A25C034I | MI | MILA |
| 3 | Mario   | Verdi   | 01312342323 | VRDMRO66A25C034I | MI | MILA |
| 4 | Mario   | Verdi   | 01312342323 | VRDMRO66A25C034I | MI | MILA |
+----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Abbiamo quattro righe in tutto, ora proviamo a cancellare la riga 4 che ha il codice=4 (auto incrementale), l'istruzione è **DELETE FROM anagrafica WHERE codice=4 LIMIT 1;**

Successivamente aggiungiamo un nuovo campo con l'istruzione:

INSERT INTO anagrafica (nome,cognome,partitaiva,codicefiscale,provincia,città,livello) VALUES ('Mario', 'Verdi', '01312342323','VRDMRO66A25C034I','MI', 'MILANO', 3);

```
+----+-----+-----+-----+-----+-----+-----+
| codice | nome   | cognome | partitaiva | codicefiscale | provincia | città |
+----+-----+-----+-----+-----+-----+-----+
| 1 | Roberto | Bianchi | 01212342343 | BNCRBT67A26C034I | RM | Roma |
| 2 | Mario   | Verdi   | 01312342323 | VRDMRO66A25C034I | MI | MILA |
| 3 | Mario   | Verdi   | 01312342323 | VRDMRO66A25C034I | MI | MILA |
| 5 | Mario   | Verdi   | 01312342323 | VRDMRO66A25C034I | MI | MILA |
+----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Notiamo che il nuovo record aggiunto ha il codice (tipo auto incrementale) uguale a 5, in quanto il record cancellato aveva codice uguale a 4. Vi è pertanto un salto dal codice 3 al codice 5, analizziamo il modo per risolvere tale problema. Basta cancellare il nuovo record con codice=5 tramite il comando:

DELETE FROM anagrafica WHERE codice=5; e successivamente digitare **ALTER TABLE anagrafica AUTO_INCREMENT = 4;** quindi riprovare ad inserire la **INSERT INTO** sopra e verificare il tutto.

MySQL – Principali comandi

Cambiare database

Con il comando **show databases;** vengono elencati tutti i database presenti. Per cambiare database si utilizza il comando **use nomedatabase;** in alternativa **\u nomedatabase;**

Per visualizzare le tabelle del database in uso, si utilizza, come già visto in precedenza, l'istruzione **show tables;**

Vediamo un esempio:

use personale;

show tables;

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| personale |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql> use personale;
Database changed
mysql> show tables;
+-----+
| Tables_in_personale |
+-----+
| anagrafica |
| corsi |
| studente |
+-----+
3 rows in set (0.00 sec)

mysql>
```

Visualizza la struttura di una tabella.

Per verificare la struttura di una tabella di un database, l'istruzione è

describe anagrafica;

```
mysql> describe anagrafica;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codice | int(11) | NO | PRI | NULL | auto_increment |
| nome | varchar(45) | YES | | NULL | |
| cognome | varchar(45) | YES | | NULL | |
| partitativa | varchar(11) | YES | | NULL | |
| codicefiscale | varchar(16) | YES | | NULL | |
| provincia | varchar(2) | YES | | NULL | |
| città | varchar(45) | YES | | NULL | |
| livello | int(4) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
8 rows in set (0.08 sec)

mysql>
```

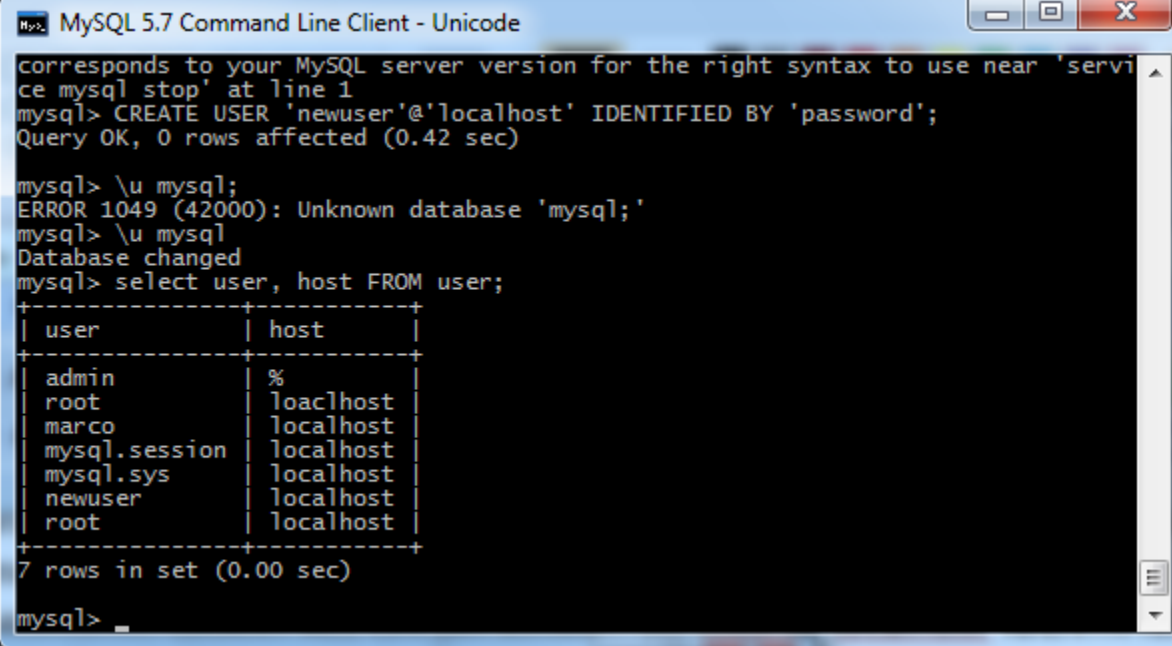
MySQL– Principali comandi

Mostra gli user di MySQL

Il comando per mostrare gli user di MySQL è:

`\u mysql`

`select user, host FROM user;`



```
MySQL 5.7 Command Line Client - Unicode
corresponds to your MySQL server version for the right syntax to use near 'servi
ce mysql stop' at line 1
mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.42 sec)

mysql> \u mysql;
ERROR 1049 (42000): Unknown database 'mysql;'
mysql> \u mysql
Database changed
mysql> select user, host FROM user;
+-----+-----+
| user          | host          |
+-----+-----+
| admin         | %             |
| root          | localhost    |
| marco         | localhost    |
| mysql.session | localhost    |
| mysql.sys     | localhost    |
| newuser       | localhost    |
| root          | localhost    |
+-----+-----+
7 rows in set (0.00 sec)

mysql> _
```


MySQL– administrator

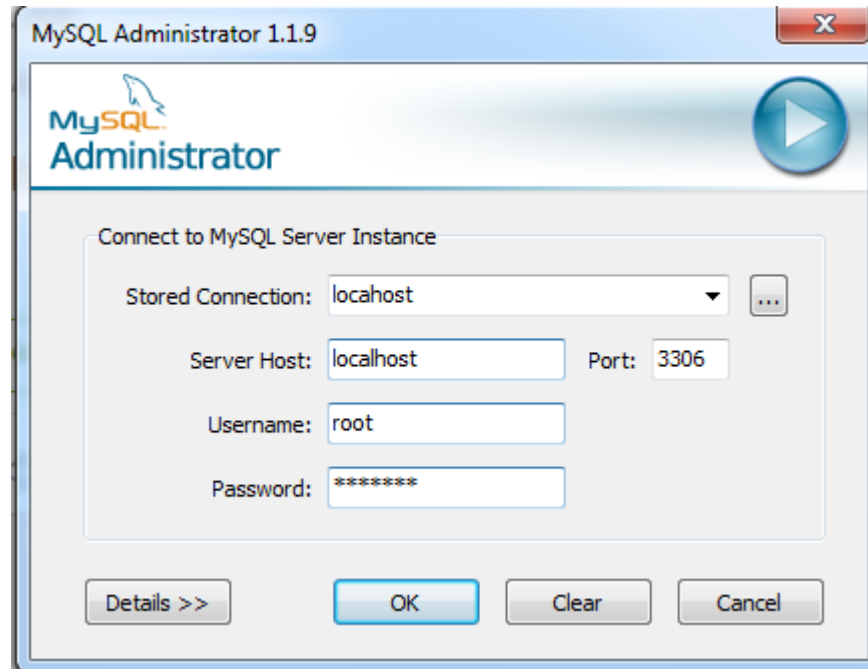
Il programma MySQL Administrator.

MySQL Administrator è un software per effettuare operazioni di amministrazione, quali configurare il server MySQL, avviarlo, fermarlo, effettuare backup ed una serie di altri compiti.

E' possibile compiere la maggior parte di queste operazioni anche usando un'interfaccia a riga di comando, come quella fornita da mysqladmin o quella di mysql, ma MySQL Administrator offre il vantaggio di avere un'interfaccia grafica più intuitiva (**friendly**).

E' possibile scaricare il software MySQL Administrator dal seguente link:

<https://mysql-administrator.it.uptodown.com/windows/download>



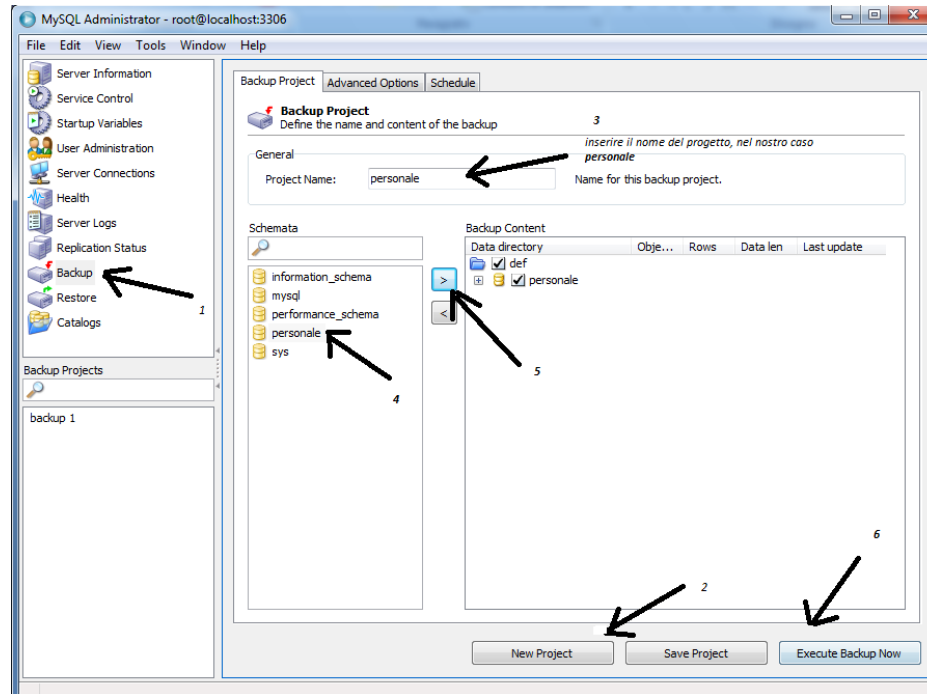
Una volta scaricato ed eseguito, andiamo nella voce Stored Connection inseriamo localhost, quindi Username root e password roberto.

MySQL– Administrator

Il programma MYSQL Administrator.

Tra le tante opzioni del MySQL Administrator, rivolgiamo la nostra attenzione alla possibilità di eseguire un backup dei database. Vediamo di seguito le operazioni da seguire.

Seguiamo l'ordine delle frecce e salviamo il backup sulla cartella desktop.



Si sottolinea l'importanza di eseguire periodicamente il backup dei database. L'esecuzione di procedure di backup e restore (ripristino) degli archivi in una posizione esterna sicura costituiscono modi validi per evitare una perdita di dati potenzialmente irreversibile.

Strategie di backup e ripristino

Il backup e il ripristino dei dati devono essere sempre personalizzati per uno specifico ambiente. Un utilizzo affidabile di backup dei dati ed un corretto ripristino delle informazioni richiede una strategia che consideri sempre il giusto bilanciamento tra costi e benefici.

Esercizi da implementare per connessione database

1) Sulla base degli esercizi analizzati nelle pagine precedenti, implementare in java le operazioni sotto descritte (usare il database personale e la tabella anagrafica).

- Inserire un nuovo record da tastiera (costruire il file `Inserimento_rete.java`)
- cancellazione record in base al codice (costruire il file `Cancellazione_rete.java`)
- Aggiornare un record scelto dall'utente (costruire il file `Aggiornamento_rete.java`)
- Eseguire a piacere una selezione (costruire il file `Selezione_rete.java`)
- Eseguire a piacere una proiezione (costruire il file `Proiezione_rete.java`)
- Eseguire a piacere una selezione con parametri (file `Selezione_param_rete.java`)
- Visualizzare la struttura dati – Metadati (costruire il file `Struttura_rete.java`)

2) Tramite MySQL Workbench, creare nel database personale una nuova tabella di nome **livello** contenente i seguenti campi:

Codicelivello int primary key (autoincrementato)

Descrizionelivello char(30)

StipendiBase intero(10)

Titolodistudio char(30)

Quindi, progettare un programma completo che permetta di eseguire le principali operazioni (inserimento nuovo record, cancellazione record, Aggiornamento record, query di selezione, query di proiezione, visualizza struttura dati ed altre operazioni scelte a piacere).

Esercizi da implementare per connessione database

3) Contribuenti

Creare database con una tabella di nome **Contribuenti**, contenente i seguenti attributi: CodiceFiscale, Cognome, Nome, Data di nascita, indirizzo di residenza, città di residenza, cap, provincia di residenza e Reddito.

Implementare in Java le seguenti operazioni:

- Inserire nuovo contribuente.
- Stampare il codice fiscale, cognome, nome e reddito del contribuenti secondo la provincia di residenza data in input.
- Stampa solo i contribuenti che hanno un reddito superiore a un valore fornito in input.

4) Motivi musicali

Creare un database con due tabelle chiamate **Artisti** e **Brani**. La tabella Artisti deve contenere i seguenti campi (Codice Artista, Nome Artista, Codice Brano). La tabella Brani deve contenere i seguenti campi (Codice Brano, Titolo, Genere, Anno di produzione).

Costruire delle classi in java che consentano di gestire le varie operazioni di inserimento, cancellazione e modifica dei record. Infine progettare le seguenti operazioni:

- Inserire il nome dell'artista e stampare tutti i brani associati.
- Scegliere un genere e stampare tutti i brani appartenente a quel genere.
- Stampare l'elenco dei brani riferiti ad un determinato anno di produzione.
- Stampare l'elenco completo degli artisti e dei brani.

Esercizi da implementare per connessione database

5) Voli aerei

Creare un database con una tabella denominata **voli**.

La tabella voli deve contenere i seguenti campi (Numero del volo, Descrizione compagnia, Data partenza, Ora partenza, Numero passeggeri, Prezzo volo).

Implementare in Java tutte le procedure necessarie per gestire la tabella (inserimento, cancellazione, modifica, ecc.).

Fornire anche il codice sorgente per gestire le attività di seguito richieste:

-Dopo avere fornito da tastiera il nome della compagnia, calcolare quanti sono i voli appartenenti ad essa.

-Inserire una data di inizio ed una di fine, stampare tutti i voli effettuati nelle due date.

-Calcolare la fascia oraria dove si hanno più passeggeri.

6) Concorso pubblico

Implementare in Java un progetto che consenta di gestire un concorso pubblico. Le prove del concorso sono tre: scritto, colloquio e prova pratica. Predisporre un database che contenga una tabella denominata **Anagrafica** con i seguenti campi (Codice, Cognome, Nome, Sesso, Luogo nascita, Data nascita, Codice fiscale, risultato scritto, risultato colloquio, risultato pratico). Scrivere le applicazioni in Java che consentono di:

Inserire i dati di un candidato; aggiornare i risultati delle prove; rimuovere un candidato, elencare i candidati che non hanno superato la prima prova (punteggio inferiore a 18); elencare i candidati con il punteggio totale, dopo aver selezionato i candidati che hanno ottenuto in tutte le prove un punteggio superiore al 18.

Esercizi da implementare per connessione database

7) Centraline per rilevazione dell'inquinamento

Si implementi un progetto in Java per realizzare le rilevazioni dell'inquinamento in una determinata regione. Si precisa che le centraline sono dislocate in punti diversi. I dati sono raccolti in un database di nome **inquinamento**.

Le tabelle del database sono:

- Centralina(Codice, modello, indirizzo, provincia, regione);
- Rilevazioni(ID, Data, valore, **CodiceCentralina**).

Le chiavi primarie sono sottolineate, mentre la chiave esterna, che collega la tabella Rilevazioni alla tabella Centraline, è in grassetto.

Svolgere le seguenti operazioni:

- Creare il database con le tabelle e assegnare le chiavi primarie alla tabella;
- Creare le relazioni tra le tabelle facendo corrispondere chiave primaria e chiave esterna;
- Inserire almeno 10 righe di dati per ciascuna tabella;
- Visualizzare con una query l'elenco delle rilevazioni di una centralina, fornita come parametro;
- elencare tutte le centraline di una determinata provincia;
- elencare tutte le rilevazioni di una data odierna;
- elencare tutte le rilevazioni che superano un determinato valore minimo;