

Accedere a un database con Java

Premessa

Per svolgere questo esercizio guidato occorre disporre del seguente software già installato e configurato:

- **MySQL** (<http://dev.mysql.com/downloads/>)
- facoltativamente un client a interfaccia grafica per la gestione del database, come **EMS** (<http://www.sqlmanager.net/products/mysql/manager>) o analoghi (es. **PhpMyAdmin** http://www.phpmyadmin.net/home_page/downloads.php)
- il **Java Development Kit** (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>)
- l'ide **Eclipse** (<http://www.eclipse.org/downloads/>)

Creare il database

Utilizzando il client a interfaccia grafica disponibile, creiamo:

- il database **libri**
- la tabella **autori** seguente:

autori

autID	nome	cogn
1	Mario	Rossi
2	Anna	Verdi
3	Laura	Bianchi
4	Ettore	Neri

L'attributo **autID** è un intero che si incrementa automaticamente ad ogni nuovo inserimento ed è la *chiave primaria* della tabella autori.

L'attributo **nome** è una stringa di lunghezza 30 che contiene il nome dell'autore e non può essere NULL.

L'attributo **cogn** è una stringa di lunghezza 30 che contiene il cognome dell'autore e non può essere NULL.

La tabella è definita dal seguente codice SQL

```
CREATE TABLE autori (  
    autID INT(11) AUTO_INCREMENT PRIMARY KEY,  
    nome VARCHAR(30) NOT NULL,  
    cogn VARCHAR(30) NOT NULL  
);
```

Vedere il tutorial "Creare una tabella con PhpMyAdmin"

<http://www.scribd.com/doc/100590392/Creare-una-tabella-con-PhpMyAdmin>

Ottenere il JDBC driver per MySQL

L'accesso a un database da Java avviene per mezzo di JDBC, che è un'estensione delle Java API per la preparazione e l'esecuzione di statement SQL. JDBC consiste di una serie di classi e interfacce Java, grazie alle quali è possibile sviluppare applicazioni che accedono a database in maniera standardizzata. Quando si scrive un'applicazione con Java-JDBC questa applicazione può essere usata per accedere a qualsiasi database relazionale. In altre parole non ci si deve preoccupare dell'ambiente. Il collegamento reale tra JDBC e il database è realizzato tramite uno strato software noto come **JDBC driver**, di cui ne esiste una versione per ogni database accessibile da Java. In definitiva, un'applicazione deve scegliere il JDBC driver giusto e usare la stessa sintassi per connettersi ad un database qualsiasi.

Il driver per accedere a un database MySQL è disponibile per il download all'url <http://dev.mysql.com/downloads/connector/j>

Una volta effettuato il download, estraete i file dalla cartella compressa e cercate il file **mysql-connector-java-x.x.xx.jar**.

Per rendere "visibile" il driver ai programmi Java che lo utilizzeranno si deve inserire il percorso che porta a questo file nella variabile d'ambiente CLASSPATH.

In ambiente Eclipse è sufficiente importare il file jar nel progetto agendo sulle proprietà dello stesso.

Vedere il tutorial "Importazione di librerie di terze parti in Eclipse"

<http://www.scribd.com/doc/100590732/Importazione-Libreria-Terze-Parti>

Passi fondamentali per accedere al database

Il codice Java che consente di accedere al database e alle tabelle desiderate, per compiere le operazioni di interesse, comprende alcune istruzioni che analizzeremo nel seguito.

1) Caricamento del driver

Class.forName(driver);

Il parametro driver è una stringa che indica il driver per MySQL e ha il seguente formato

String driver = "com.mysql.jdbc.Driver";

Il metodo **forName** è un metodo statico della classe **Class** (package java.lang) che ha come effetto quello di caricare la classe indicata nella stringa passata come parametro. La chiamata a **forName** carica, quindi, il driver oppure lancia un'eccezione **ClassNotFoundException** se la classe non viene trovata.

2) Connessione a MySQL

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

Il parametro **url** è una stringa che specifica l'host a cui connettersi (il server su cui "gira" MySQL). La stringa ha il seguente formato:

jdbc:<subprotocol>:<subname>

dove subname è l'indirizzo dell'host, mentre subprotocol è il nome del driver o del meccanismo di connessione al database. Se l'host coincide con localhost si avrà:

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

I parametri **username** e **password** sono altre due stringhe che specificano lo username e la password per accedere a MySQL. Se non occorre specificare username e password, le due stringhe coincideranno con stringa vuota ("").

Il metodo **getConnection** è un metodo statico della classe **DriverManager** (package java.sql), classe che fornisce funzionalità per la gestione dei driver JDBC. Il metodo getConnection restituisce un oggetto **Connection** (*interfaccia* del package java.sql) oppure lancia un'eccezione **SQLException** se non è possibile stabilire la connessione. Dopo l'esecuzione dello statement, l'oggetto **conn** rappresenta il collegamento fisico al database server.

3) Selezione del database

conn .setCatalog(database);

Il parametro **database** è una stringa contenente il nome del database su cui si intende lavorare.

Il metodo **setCatalog** di Connection lancia un'eccezione **SQLException** se si verifica un errore di accesso al database o se la connessione conn è chiusa.

4) Preparazione ed esecuzione di query

Statement st = conn.createStatement();

Il metodo **createStatement** di Connection restituisce un oggetto Statement (*interfaccia* del package java.sql) per inviare comandi al database oppure lancia un'eccezione **SQLException** se si verifica un errore di accesso al database o se la connessione conn è chiusa.

Successivamente, prepariamo la stringa che contiene la query

String sqlQuery = ".....";

L'esecuzione avviene in modi diversi a seconda del tipo di query.

Nel caso di una query SELECT occorre scrivere

ResultSet rs = st.executeQuery(sqlQuery);

Il metodo **executeQuery** di Statement restituisce un singolo resultset ("tabella") oppure lancia un'eccezione **SQLException** se si verifica un errore di accesso al database o se la connessione st è chiusa o se la query restituisce qualcosa di diverso da un singolo resultset.

Nel caso di query di aggiornamento (INSERT, UPDATE, ...), che non restituiscono un resultset, occorre scrivere

```
int rows = st.executeUpdate(sqlQuery);
```

Il metodo **executeUpdate** di Statement restituisce il numero di righe che sono state modificate oppure lancia un'eccezione **SQLException** se si verifica un errore di accesso al database o se la connessione st è chiusa o se la query restituisce un resultset.

5) Preparazione ed esecuzione di query parametriche

Una query parametrica è una query che ha uno o più valori non specificati (perché non ancora noti all'atto della preparazione della query), rappresentati da **punti interrogativi**.

Per esempio, la seguente è una query non parametrica:

```
SELECT * FROM aziende WHERE idNazione = "Italia";
```

La stessa query con parametri:

```
SELECT * FROM aziende WHERE idNazione = ?;
```

Per preparare la query occorre scrivere

```
PreparedStatement ps= conn.prepareStatement(sqlQuery);
```

Il parametro **sqlQuery** è la stringa che contiene la query SQL.

Il metodo **prepareStatement** appartiene all'interfaccia Connection ed effettua una pre-compilazione della query SQL consentendo così di creare query parametriche. Il metodo restituisce un oggetto **PreparedStatement** (*interfaccia* del package java.sql).

Nel caso di utilizzo di parametri è necessario, subito dopo la precompilazione, specificare il loro valore. PreparedStatement fornisce una serie di metodi per impostare i parametri di una query, che dipendono dal tipo del parametro.

Nel precedente esempio, siccome il parametro è una stringa (Italia), si scriverà

```
ps.setString(1,"Italia")
```

Il primo argomento del metodo setString rappresenta sempre la posizione del parametro all'interno della query (a partire da 1); il secondo argomento è il suo valore.

Altri metodi set sono setInt, setDouble, setFloat, setBoolean, setDate, setNull, setTime; l'elenco completo è consultabile all'url:

<http://download.oracle.com/javase/6/docs/api/java/sql/PreparedStatement.html>

Tutti i metodi set per mappare un tipo Java in un tipo SQL lanciano un'eccezione **SQLException** se non vi è corrispondenza tra il tipo del parametro passato e il tipo del parametro nella query, se si verifica un errore di accesso al database o se la connessione ps è chiusa.

La tabella seguente mostra la corrispondenza tra i principali tipi Java e MySQL.

MySQL Type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte []
VARBINARY	byte []
LONGVARBINARY	byte []
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

<http://www.roseindia.net/jdbc/jdbc-mysql/mapping-mysql-data-types-in-java.shtml>

La modalità di preparazione ed esecuzione di statement SQL può essere utilizzata anche per query non parametriche, in alternativa a quanto illustrato nel punto 4.

L'esecuzione richiede statement diversi a seconda del tipo di query.

Nel caso di una query SELECT occorre scrivere

ResultSet rs = ps.executeQuery();

Il metodo **executeQuery** di PreparedStatement restituisce un singolo resultset ("tabella") oppure lancia un'eccezione **SQLException** se si verifica un errore di accesso al database o se la connessione st è chiusa o se la query restituisce qualcosa di diverso da un singolo resultset.

Nel caso di query di aggiornamento (INSERT, UPDATE, ...), che non restituiscono un resultset, occorre scrivere

```
int rows = ps.executeUpdate();
```

Il metodo **executeUpdate** di PreparedStatement restituisce il numero di righe che sono state modificate oppure lancia un'eccezione **SQLException** se si verifica un errore di accesso al database o se la connessione st è chiusa o se la query restituisce un resultset.

6) Analisi del resultset

Il resultset rs ottenuto attraverso una executeQuery si comporta come una tabella; ogni riga della tabella è composta di campi (le colonne della tabella), che sono di tipo diverso.

La tabella ha un cursore che inizialmente è posizionato **prima** della prima riga; il cursore può essere spostato sulla riga successiva con **rs.next()**.

Il metodo **next** restituisce true se la riga successiva è valida, false se non ci sono più righe e lancia un'eccezione **SQLException** se si verifica un errore di accesso al database o se la connessione rs è chiusa.

Quando il cursore è posizionato su una riga, possiamo estrarre i valori dei vari campi usando appositi metodi get dell'interfaccia ResultSet: getInt, getDouble, getBoolean,...; l'elenco completo è consultabile all'url:

<http://download.oracle.com/javase/6/docs/api/java/sql/ResultSet.html>

I metodi get citati in precedenza sono forniti in due differenti formati; consideriamo ad esempio il metodo getString, ma il discorso è generalizzabile a tutti gli altri metodi.

getString(String columnLabel), con un parametro stringa che indica l'etichetta della colonna della tabella, restituisce la stringa contenuta nella colonna specificata e nella riga corrente. L'etichetta della colonna è quella specificata con **AS**¹, se esiste, altrimenti coincide con il nome della colonna.

getString(int columnIndex), con un parametro intero che indica il numero della colonna della tabella, restituisce la stringa che si trova nella colonna specificata e nella riga corrente.

I metodi get lanciano un'eccezione **SQLException** se la colonna specificata è errata (etichetta o numero non esistente), se si verifica un errore di accesso al database o se la connessione rs è chiusa.

L'estrazione dei risultati può avvenire, per esempio, con un ciclo

```
while (rs.next()) {  
    //estrazione dei dati  
}
```

¹ AS è utilizzato da SQL per gli alias, cioè per ridenominare un identificatore (di campo, di tabella).

Esempio

Supponiamo di aver eseguito la query

```
SELECT * FROM autori;
```

L'esecuzione della query restituisce in rs tutta la tabella autori

autori

autID	nome	cogn
1	Mario	Rossi
2	Anna	Verdi
3	Laura	Bianchi
4	Ettore	Neri

Per estrarre i dati dalla tabella e stamparli, scriviamo

```
while (rs.next()) {  
  
    //estrazione dei dati  
    System.out.println(rs.getInt("autID"));  
    System.out.println(rs.getString("nome"));  
    System.out.println(rs.getString("cogn"));  
  
}
```

o, analogamente

```
while (rs.next()) {  
  
    //estrazione dei dati  
    System.out.println(rs.getInt(1));  
    System.out.println(rs.getString(2));  
    System.out.println(rs.getString(3));  
  
}
```

o una combinazione delle due.

7) Chiusura delle connessioni aperte

Al termine delle operazioni o appena possibile, bisogna chiudere le connessioni che sono state aperte, con il metodo **close**.

```
rs.close();  
st.close();  
ps.close();  
conn.close();
```

I metodi close lanciano un'eccezione **SQLException** se si verificano problemi di accesso al database (le prime tre) o se non è possibile accedere al server (l'ultima).

Il codice della classe DbConnect

Il codice seguente è stato organizzato e commentato per uso strettamente didattico. La classe DbConnect versione 2 offre le stesse funzionalità ma presenta il codice organizzato in un unico blocco try.

Prima di testare il codice abbiate cura di specificare username e password per accedere a MySQL al posto di mio_username e mia_password (evidenziate in giallo nel codice).

DbConnect.java versione 1

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.swing.JOptionPane;

public class DbConnect {
    public static void main(String[] args){
        /*-----VARIABILI LOCALI CONDIVISE DA PIU' BLOCCHI-----*/

        String sqlQuery = null;
        String cognome = null;
        //occorre importare java.sql.Connection
        Connection conn = null;
        //occorre importare java.sql.Statement
        Statement st = null;
        //occorre importare java.sql.PreparedStatement
        PreparedStatement ps = null;
        //occorre importare java.sql.ResultSet
        ResultSet rs = null;

        /*-----STEP 1 - CARICAMENTO DEL DRIVER-----*/

        try{
            //forName può lanciare una ClassNotFoundException
            String driver = "com.mysql.jdbc.Driver";
            Class.forName(driver);
        }
        catch (ClassNotFoundException e){
            System.out.println("Impossibile caricare il driver");
            e.printStackTrace();
        }

        /*-----STEP 2 - CONNESSIONE A MYSQL-----*/

        try{
            //occorre importare java.sql.DriverManager
            //getConnection può lanciare una SQLException
            String url = "jdbc:mysql://localhost";
            String username = "mio_username";
            String password = "mia_password";
            conn = DriverManager.getConnection(url,username,password);
```



```

    }
    //occorre importare java.sql.SQLException
    catch (SQLException e) {
        System.out.println("Impossibile stabilire una connessione con
il server");
        e.printStackTrace();
    }

    /*-----STEP 3 - SELEZIONE DEL DATABASE-----*/

    try{
        //getConnection può lanciare una SQLException
        String database = "libri";
        conn .setCatalog(database);
    }
    catch (SQLException e) {
        System.out.println("Impossibile accedere al database
selezionato");
        e.printStackTrace();
    }

    /*-----STEP 4 E 6 - PREPARAZIONE E ESECUZIONE DI QUERY SENZA
PARAMETRI E ANALISI DEL RESULTSET-----*/

    try{
        //i metodi di questo blocco possono lanciare una SQLException
        st = conn.createStatement();

        /*QUERY NON PARAMETRICA CHE RESTITUISCE LA TABELLA AUTORI*/
        sqlQuery = "SELECT * FROM autori;";
        rs = st.executeQuery(sqlQuery);
        //estrazione dei dati
        System.out.println();
        System.out.println("La query " + sqlQuery + " ha
restituito:");
        while (rs.next()) {
            System.out.println("autID = "+rs.getInt("autID"));
            System.out.println("nome = "+rs.getString("nome"));
            System.out.println("cognome = "+rs.getString("cogn"));
        }

        /*QUERY NON PARAMETRICA CHE AGGIUNGE UNA RIGA ALLA TABELLA
AUTORI*/
        sqlQuery = "INSERT INTO autori (nome, cogn) VALUES ('Paolo',
'Rossi');";
        int row = st.executeUpdate(sqlQuery);
        //Stampa di row per verificare che sia stata aggiunta una
riga (record) al database
        System.out.println();
        System.out.println("La query " + sqlQuery + " ha inserito " +
row + " record");
    }
    catch (SQLException e) {
        System.out.println("Errore nell'esecuzione dello statement
SQL");
        e.printStackTrace();
    }

    /*-----STEP 5 E 6 - PREPARAZIONE E ESECUZIONE DI QUERY PARAMETRICHE
E ANALISI DEL RESULTSET-----*/

    try{

```

```

        //i metodi di questo blocco possono lanciare una SQLException

        /*QUERY PARAMETRICA CHE RESTITUISCE I DATI DEGLI AUTORI IL
CUI COGNOME E' SPECIFICATO IN INPUT DALL'UTENTE*/
        sqlQuery = "SELECT * FROM autori WHERE cogn = ?;";
        ps= conn.prepareStatement(sqlQuery);
        cognome = JOptionPane.showInputDialog("Cognome dell'autore
che stai cercando:");
        ps.setString(1,cognome);
        rs = ps.executeQuery();
        //estrazione dei dati
        System.out.println();
        System.out.println("La query " + sqlQuery + " ha
restituito:");
        while (rs.next()) {
            System.out.println("autID = "+rs.getInt("autID"));
            System.out.println("nome = "+rs.getString("nome"));
            System.out.println("cognome = "+rs.getString("cogn"));
        }

        /*QUERY PARAMETRICA CHE AGGIUNGE UNA RIGA ALLA TABELLA
AUTORI*/
        sqlQuery = "INSERT INTO autori (nome, cogn) VALUES (?,?);";

        ps= conn.prepareStatement(sqlQuery);
        String nome = JOptionPane.showInputDialog("Nome dell'autore
da inserire:");
        ps.setString(1,nome);
        cognome = JOptionPane.showInputDialog("Cognome dell'autore da
inserire:");
        ps.setString(2,cognome);
        int row = ps.executeUpdate();
        //Stampa di row per verificare che sia stata aggiunta una
riga (record) al database
        System.out.println();
        System.out.println("La query " + sqlQuery + " ha inserito " +
row + " record");
    }
    catch (SQLException e) {
        System.out.println("Errore nell'esecuzione dello statement
SQL");
        e.printStackTrace();
    }

    /*-----STEP 7 - CHIUSURA DELLE CONNESSIONI-----*/

    try{
        rs.close();
        st.close();
        ps.close();
        conn.close();
    }
    //occorre importare java.sql.SQLException
    catch (SQLException e) {
        System.out.println("Impossibile accedere al database
selezionato o impossibile stabilire una connessione con il server");
        e.printStackTrace();
    }
}
}

```

DbConnect.java versione 2

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.swing.JOptionPane;

public class DbConnect {
    public static void main(String[] args) {

        String sqlQuery = null;
        String cognome = null;
        int row = 0;
        //occorre importare java.sql.Connection
        Connection conn = null;
        //occorre importare java.sql.Statement
        Statement st = null;
        //occorre importare java.sql.PreparedStatement
        PreparedStatement ps = null;
        //occorre importare java.sql.ResultSet
        ResultSet rs = null;
        try
        {
            /*-----STEP 1 - CARICAMENTO DEL DRIVER-----*/

            String driver = "com.mysql.jdbc.Driver";
            Class.forName(driver);

            /*-----STEP 2 - CONNESSIONE A MYSQL-----*/

            String url = "jdbc:mysql://localhost";
            String username = "mio_username";
            String password = "mia_password";
            conn = DriverManager.getConnection(url,username,password);

            /*-----STEP 3 - SELEZIONE DEL DATABASE-----*/

            String database = "libri";
            conn.setCatalog(database);

            /*-----STEP 4 E 6 - PREPARAZIONE E ESECUZIONE DI QUERY SENZA
PARAMETRI E ANALISI DEL RESULTSET-----*/

            st = conn.createStatement();

            /*QUERY NON PARAMETRICA CHE RESTITUISCE LA TABELLA AUTORI*/
            sqlQuery = "SELECT * FROM autori;";
            rs = st.executeQuery(sqlQuery);
            //estrazione dei dati
            System.out.println();
            System.out.println("La query " + sqlQuery + " ha
restituito:");
            while (rs.next()) {
                System.out.println("autID = "+rs.getInt("autID"));
                System.out.println("nome = "+rs.getString("nome"));
                System.out.println("cognome = "+rs.getString("cogn"));
            }
        }
    }
}
```

```

        /*QUERY NON PARAMETRICA CHE AGGIUNGE UNA RIGA ALLA TABELLA
AUTORI*/
        sqlQuery = "INSERT INTO autori (nome, cogn) VALUES ('Paolo',
'Rossi');";
        row = st.executeUpdate(sqlQuery);
        //Stampa di row per verificare che sia stata aggiunta una
riga (record) al database
        System.out.println();
        System.out.println("La query " + sqlQuery + " ha inserito " +
row + " record");

        /*-----STEP 5 E 6 - PREPARAZIONE E ESECUZIONE DI QUERY
PARAMETRICHE E ANALISI DEL RESULTSET-----*/

        /*QUERY PARAMETRICA CHE RESTITUISCE I DATI DEGLI AUTORI IL
CUI COGNOME E' SPECIFICATO IN INPUT DALL'UTENTE*/
        sqlQuery = "SELECT * FROM autori WHERE cogn = ?";
        ps= conn.prepareStatement(sqlQuery);
        cognome = JOptionPane.showInputDialog("Cognome dell'autore
che stai cercando:");
        ps.setString(1,cognome);
        rs = ps.executeQuery();
        //estrazione dei dati
        System.out.println();
        System.out.println("La query " + sqlQuery + " ha
restituito:");
        while (rs.next()) {
            System.out.println("autID = "+rs.getInt("autID"));
            System.out.println("nome = "+rs.getString("nome"));
            System.out.println("cognome = "+rs.getString("cogn"));
        }

        /*QUERY PARAMETRICA CHE AGGIUNGE UNA RIGA ALLA TABELLA
AUTORI*/
        sqlQuery = "INSERT INTO autori (nome, cogn) VALUES (?,?)";

        ps= conn.prepareStatement(sqlQuery);
        String nome = JOptionPane.showInputDialog("Nome dell'autore
da inserire:");
        ps.setString(1,nome);
        cognome = JOptionPane.showInputDialog("Cognome dell'autore da
inserire:");
        ps.setString(2,cognome);
        row = ps.executeUpdate();
        //Stampa di row per verificare che sia stata aggiunta una
riga (record) al database
        System.out.println();
        System.out.println("La query " + sqlQuery + " ha inserito " +
row + " record");

        /*-----STEP 7 - CHIUSURA DELLE CONNESSIONI-----*/

        rs.close();
        st.close();
        ps.close();
        conn.close();
    }

    catch (ClassNotFoundException e) {
        System.out.println("Impossibile caricare il driver");
        e.printStackTrace();
    }

```

```

    }
    //occorre importare java.sql.SQLException
    catch(SQLException e){
        System.out.println("Impossibile accedere al database
selezionato o " +
                        "impossibile stabilire una connessione con il
server o errore nell'esecuzione dello statement SQL");
        e.printStackTrace();
    }
}
}

```

L'output prodotto












La query `SELECT * FROM autori;` ha restituito:

```

autID = 1
nome = Mario
cognome = Rossi
autID = 2
nome = Anna
cognome = Verdi
autID = 3
nome = Laura
cognome = Bianchi
autID = 4
nome = Ettore
cognome = Neri

```

La query `INSERT INTO autori (nome, cogn) VALUES ('Paolo', 'Rossi');` ha inserito 1 record

			<u>autID</u>	<u>nome</u>	<u>cogn</u>
<input type="checkbox"/>			1	Mario	Rossi
<input type="checkbox"/>			2	Anna	Verdi
<input type="checkbox"/>			3	Laura	Bianchi
<input type="checkbox"/>			4	Ettore	Neri
<input type="checkbox"/>			5	Paolo	Rossi


La query `SELECT * FROM autori WHERE cogn = ?;` ha restituito:

```

autID = 1
nome = Mario
cognome = Rossi
autID = 6
nome = Paolo
cognome = Rossi

```

La query `INSERT INTO autori (nome, cogn) VALUES (?,?);` ha inserito 1 record

			<u>autID</u>	<u>nome</u>	<u>cogn</u>
<input type="checkbox"/>			1	Mario	Rossi
<input type="checkbox"/>			2	Anna	Verdi
<input type="checkbox"/>			3	Laura	Bianchi
<input type="checkbox"/>			4	Ettore	Neri
<input type="checkbox"/>			5	Paolo	Rossi
<input type="checkbox"/>			6	Luca	Gialli

Quest'opera è stata rilasciata con licenza Creative Commons Attribution-ShareAlike 3.0 Unported. Per leggere una copia della licenza visita il sito web <http://creativecommons.org/licenses/by-sa/3.0/> o spedisci una lettera a Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.