Programmazione Java

Davide Di Ruscio
Dipartimento di Informatica
Università degli Studi dell'Aquila

diruscio@di.univaq.it

Sommario

- Introduzione
- Risorse
- Driver
- Contesto Applicativo
- SQL
- Classi e interfacce
- Passi
  - Registrare un driver
  - Stabilire una connessione al DB
  - Creare uno statement
  - Eseguire l'SQL
  - Processare il risultato
  - Eliminare gli oggetti JDBC

Java Database Connectivity (JDBC)

- E’ costituita da
  - Un insieme di interfacce che fanno parte della piattaforma Java e costituiscono le API per il programmatore
  - Gestore di driver che permette a driver di terze parti di connettersi ad un DB specifico
- Un ***driver*** JDBC permette di
  - Connettersi ad un DB
  - Inviare un *comando SQL*
  - Processare il risultato

Risorse

»Libri

- Titolo: **Java 2 Volume II – settima edizione**
  Autori: Cay S. Horstmann, Gary Cornell
  Casa Editrice: Prentice Hall
  ISBN: 88-7192-237-9
- Titolo: **JDBC API Tutorial and Reference – Terza edizione**
  Autori: Maydene Fisher, Jon Ellis, Jonathan Bruce
  Casa Editrice: Addison-Wesley Professional
  ISBN: 0321173848

»Java Tutorial – JDBC Trial

- http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html

»**JDBC API Documentation**

- http://java.sun.com/j2se/1.5.0/docs/guide/jdbc/index.html
- http://java.sun.com/j2se/1.5.0/docs/api/java/sql/package-summary.html

Driver (1)

1. Tipo 1 (Ponte JDBC-ODBC)
   - Traduce JDBC in ODBC
   - Viene utilizzato un driver ODBC
   - JDK contiene un ponte JDBC-ODBC
   - E' necessario configurare ODBC
   - Generalmente utilizzato in ambito di test
2. Tipo 2 (Driver con API parzialmente native e Java)
   - Scritto parzialmente in java e parzialmente in codice nativo
   - Chiamate JDBC vengono convertite in chiamate alle API dei client Oracle, IBM DB2
   - E' necessario installare le librerie native

Driver (2)

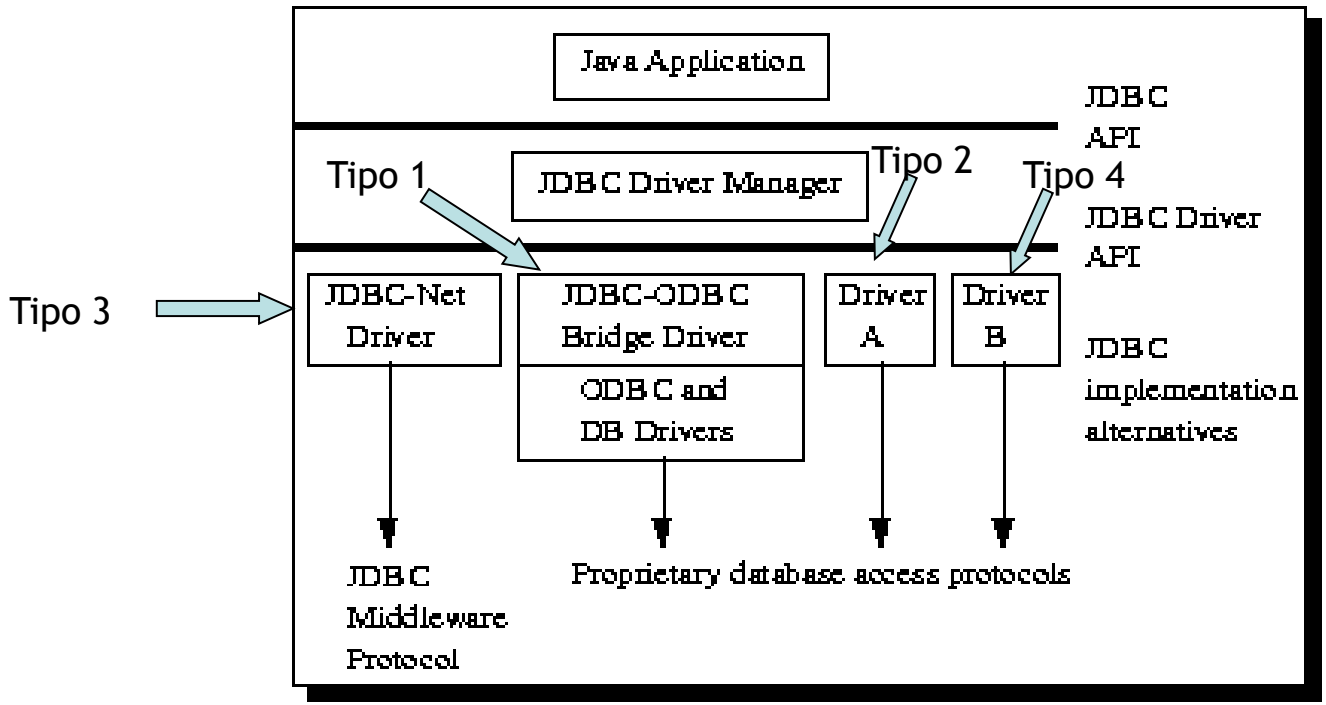
1. Tipo 3
   - Chiamate JDBC vengono traslate in un protocollo DBMS-independent
   - Vengono poi traslate da un server nel relativo protocollo del DBMS
2. Tipo 4
   - Chiamate JDBC vengono convertite direttamente nelle chiamate al protocollo del DBMS

Nota

E' presente una lista di driver disponibili sul mercato

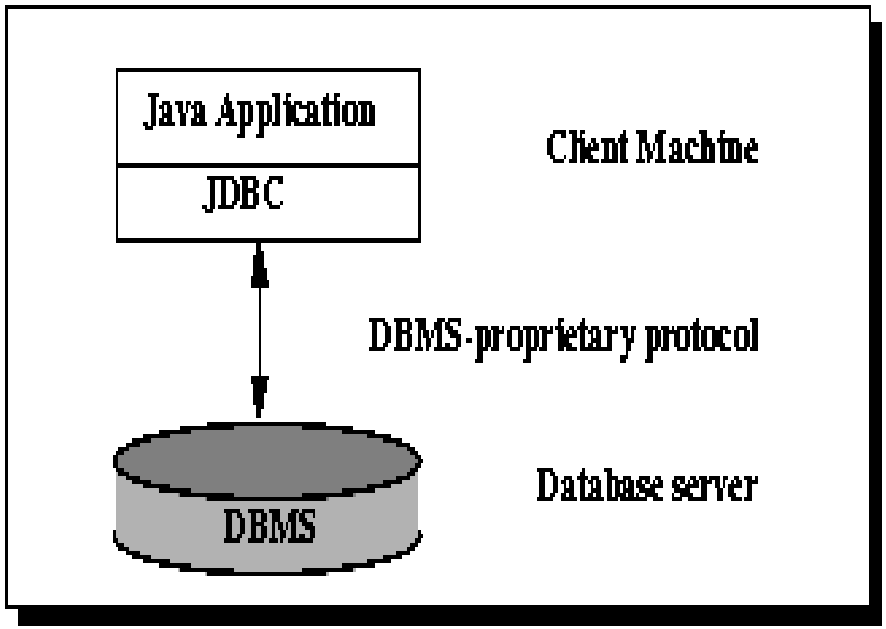
- http://developers.sun.com/product/jdbc/drivers

Driver (3)

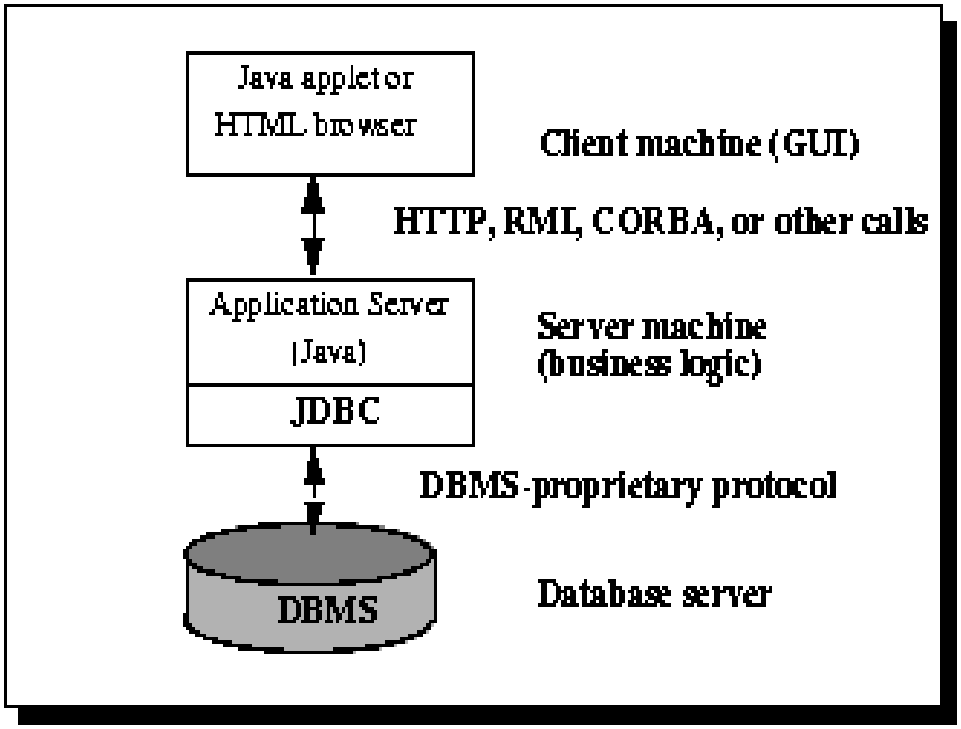

Java Application
JDBC API
Tipo 1
JDBC Driver Manager
Tipo 2
Tipo 4
JDBC Driver API
Tipo 3
JDBC-Net Driver
JDBC-ODBC Bridge Driver
Driver A
Driver B
JDBC implementation alternatives
ODBC and DB Drivers
JDBC Middleware Protocol
Proprietary database access protocols

Contesto Applicativo

Applicazioni Two-tier



Applicazioni Three-tier



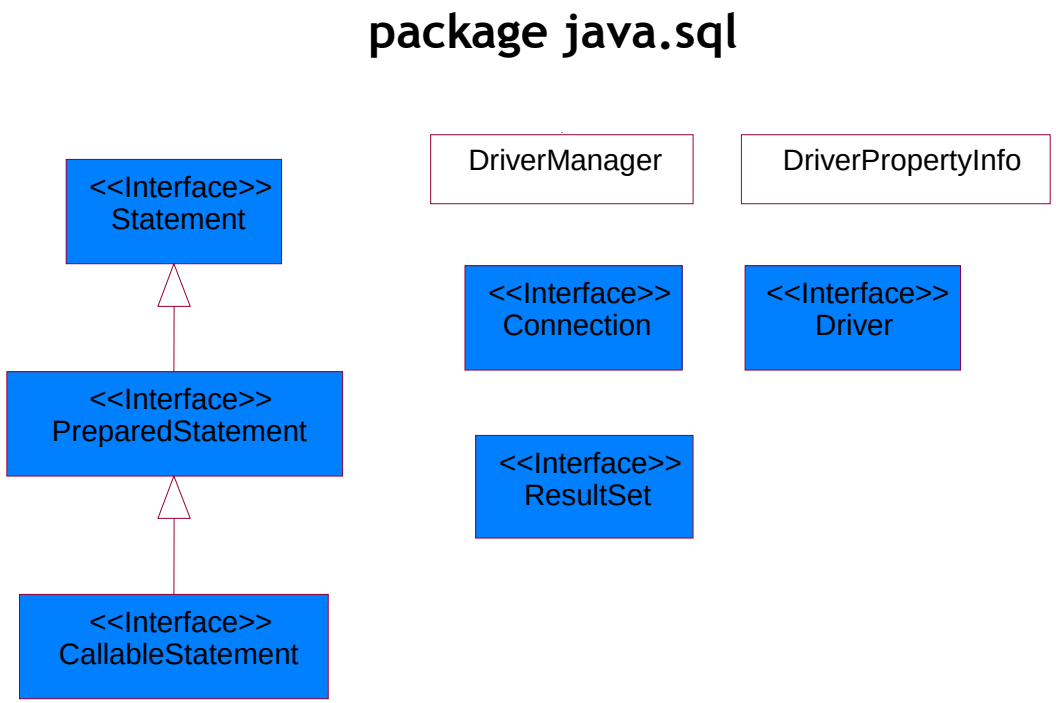
SQL (1)

- SQL è il linguaggio standard per accedere ai DB relazionali
- SQL non è standardizzato
  - Per i tipi
    - JDBC lo risolve introducendo un insieme generico di tipi
  - Per i diversi *statement*
    - JDBC permette l'invio di una qualsiasi istruzione SQL (i driver devono essere almeno conformi all'ANSI SQL-92 Entry Level)

SQL (2)

`INTEGER O INT`	Di solito intero a 32 bit
`SMALLINT`	Di solito intero a 16 bit
`NUMERIC(M,N), DECIMAL(M,N) DEC(M, N)`	Numero decimale a lunghezza fissa con m cifre totali e n cifre dopo il punto decimale
`FLOAT(N)`	Numero a virgola mobile con precisione di n cifre binarie
`REAL`	Di solito virgola mobile a 32bit
`DOUBLE`	Di solito virgola mobile a 64bit
`CHARACTER(N) CHAR(N)`	Stringa di lunghezza fissa n
`VARCHAR(N)`	Stringa di lunghezza variabile max n
`BOOLEAN`	Booleano
`DATE`	Data del calendario dipendente dall'implementazione
`TIME`	Ora del giorno dipendente dall'implementazione
`TIMESTAMP`	Data e ora del giorno dipendente dall'implementazione
`BLOB`	Oggetto binario large
`CLOB`	Oggetto carattere large

Classi e interfacce (1)



Classi e interfacce (2)

» DriverManager facility

- `DriverManager`: permette una connessione con un driver
- `Driver`: fornisce le API per registrare e connettere i driver; utilizzata dalla classe `DriverManager`
- `DriverPropertyInfo`: fornisce le proprietà di un driver JDBC (generalmente non utilizzata)

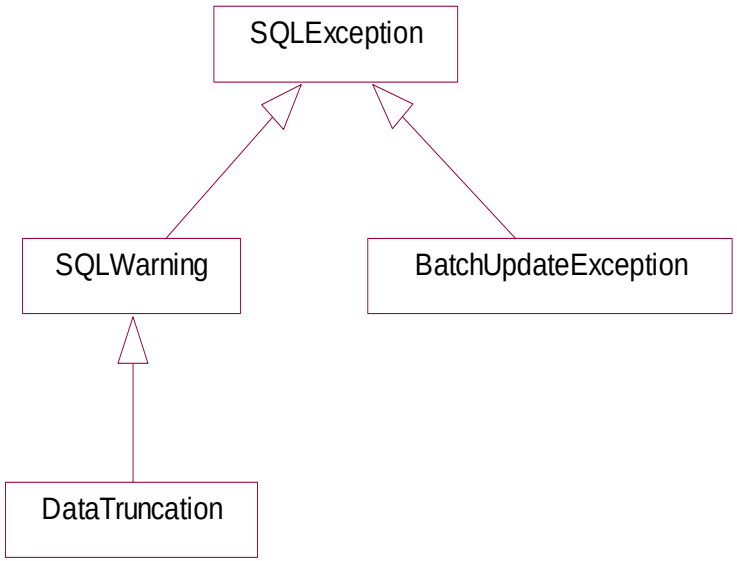
…

Classi e interfacce (3)

- **SQL statements**
  - `Statement`: utilizzata per inviare statements base SQL
  - `PreparedStatement`: utilizzata per inviare prepared statements
  - `CallableStatement`: utlilizzata per invocare le stored procedure
  - `Connection`: fornisce metodi per creare statements e gestire le connessioni
  - …
- `ResultSet`: rappresenta il risultato di una query

Classi e interfacce (4)

package java.sql



- `SQLException`
  - Lanciata dai metodi quando vi è un problema nell'accesso ai dati o per altri ragioni
- `SQLWarning`
  - Indica un warning
- `DataTruncation`
  - Indica dati che potrebbero essere stati troncati
- `BatchUpdateException`
  - Indica che non tutti i comandi in un update batch non sono stati eseguiti con successo

» Getting start

- Installa Java e JDBC (☺)
- Installa il DBMS
  - MySql 4.0.x
- Download and Installa un driver
  - mysql-connector-java-3.1.14-bin.jar (http://dev.mysql.com/downloads/connector/j/3.1.html)
- Creazione del Database
  - javalibrary

Database


```
drop database javalibrary;
create database javalibrary;
use javalibrary;

CREATE TABLE title_kind(
title_kind_id mediumint(10) NOT NULL AUTO_INCREMENT PRIMARY KEY,
name varchar(100) not null);

CREATE TABLE title(
title_id mediumint(10) NOT NULL AUTO_INCREMENT PRIMARY KEY,
name varchar(255) not null,
title_kind_id mediumint(10) not null,
description varchar(255),
author varchar(100),
isbn varchar(50),
constraint ttl_tik_fk foreign key(title_kind_id) references
  title_kind(title_kind_id));
```

Passi

- Registrare un driver
- Stabilire una connessione al DB
- Creare uno statement
- Eseguire l'SQL
- Processare il risultato
- *Eliminare* gli oggetti JDBC

Registrare un driver (1)

» Interfaccia `Driver` fornisce un'astrazione verso il DB

» Viene utilizzato dal `DriverManager` per connettersi al DB

- E' lo strato di gestione all'interno di JBDC che si interpone tra l'utente e i drivers
- Mantiene traccia dei driver che sono disponibili e gestisce le connessioni tra un DB e il relativo driver
- JDBC 2.0 ha introdotto un'interfaccia `DataSource` (`javax.sql`) che è un'alternativa alla gestione delle sorgenti di dati

» JDBC utilizza il primo driver che si connette con successo ad una data URL

» Si possono registrare diversi driver

Registrare un driver (2)

» Un driver viene registrato

- Utilizzando il Class Loader
  - `Class.forName("acme.db.Driver");`
  - Permette di caricare esplicitamente il driver
  - Il driver una volta caricato si registra presso il `DriverManager` mediante il metodo `registerDriver`
- Utilizzando la proprietà di sistema `jdbc.drivers`
  - Sintassi: `java -Djdbc.drivers=`*`driverName`*`[:`*`driverName`*`]`
- Istanziando esplicitamente una classe che implementa l'interfaccia `Driver`

Registrare un driver (3)

» Esempio 1

```
public class TestLoading1 {

    public static void main( String[] args ) {
        try {
            Class.forName("com.mysql.jdbc.Driver" );
        }
        catch ( ClassNotFoundException e ) {
            e.printStackTrace();
        }
    }
}
```

Registrare un driver (4)

» Esempio 2

```
public class TestLoading2 {

    public static void main( String[] args ) {

        System.out.println( "welcome!" );
    }

}
```

```
C:> java -Djdbc.drivers=com.mysql.jdbc.Driver
```

Registrare un driver (5)

» Esempio 3

```
public class TestLoading3 {

    public static void main( String[] args ) {
        try {
            Driver driver =
                    new com.mysql.jdbc.NonRegisteringDriver();
            DriverManager.registerDriver( driver );
        }
        catch ( SQLException e ) {
            e.printStackTrace();
        }
    }

}
```

Stabilire una connessione al DB (1)

- Esistono tre metodi nella classe `DriverManager` per effettuare la connessione
  - `getConnection(String url)`
  - `getConnection(String url, java.util.Properties info)`
    - Le proprietà sono dipendenti dal DBMS
    - Contengono almeno le proprietà user e password
  - `getConnection(String url, String user, String password)`

Stabilire una connessione al DB (2)

- Formato URL
  - `jdbc:protocollo_secondario:altro`
  - `protocollo_secondario` seleziona il driver specifico di connessione al db
  - `altro` dipende dal valore di `protocollo_secondario`
  - Esempi

```
jdbc:mysql://localhost/javalibrary
jdbc:oracle:thin:@127.0.0.1:1521:OracleDB
```

Stabilire una connessione al DB (3)

» Esempio

```
public class TestConnection {
    private static final String DRIVER_NAME = "com.mysql.jdbc.Driver";

    public static void main( String[] args ) {
        try {
            Class.forName( DRIVER_NAME );
            Connection connection = DriverManager.getConnection( args[ 0 ],
                                                    args[ 1 ], args[ 2 ] );
        }
        catch ( ClassNotFoundException e ) {
            e.printStackTrace();
        }
        catch ( SQLException e ) {
            e.printStackTrace();
       }
    }
}

C:> java TestConnection jdbc:mysql://localhost/javalibrary root “”
```

Creare uno statement (1)

- Un oggetto `Statement` è utilizzato per inviare istruzioni SQL al DB
- Tre tipi di oggetti statement
  - `Statement`
    - Permette di eseguire semplici istruzioni SQL senza parametri
  - `PreparedStatement`
    - Viene preparata una query che poi sarà utilizzata diverse volte
    - Aumento prestazioni
  - `CallableStatement`
    - Permette di eseguire chiamate a stored procedure

Creare uno statement (2)

- Per creare gli statement si utilizzano i rispettivi metodi presenti all'interno di `Connection`
  - `Statement createStatement()`
  - `PreparedStatement prepareStatement(String sql)`
  - `CallableStatement prepareCall(String sql)`
  - Esistono altri metodi per creare gli oggetti

Eseguire l'SQL

- Interfaccia `Statement`
  - `ResultSet executeQuery(String sql)`
    - Esegue un'istruzione SQL (SELECT) e ritorna un ResultSet che identifica il risultato
  - `int executeUpdate(String sql)`
    - Esegue un'istruzione SQL (INSERT, UPDATE, DELETE) e ritorna il numero di righe di cui è stato effettuato l'update
  - …

Processare il risultato

- Interfaccia `ResultSet`
  - Mantiene un cursore che punta alla riga corrente dei dati
  - Inizialmente è posizionato prima della prima riga
  - Metodo `boolean next()` muove il cursore in avanti e ritorna `false` se non ci sono più righe
  - Default con il `ResultSet` non si possono modificare le righe e tornare indietro
  - Esempio

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
```

Processare il risultato

» Per recuperare le colonne dal `ResultSet` vi sono dei metodi `getXXX (getBoolean, getLong, getString, …)`

» Valori possono essere recuperati

- Utilizzando l'indice (più efficiente e numerato da 1) oppure
- Nome della colonna (case insensitive)

» Driver JDBC tenta di convertire dati sottostanti con tipi di dati Java

» Problema con tipi del DB rispetto a tipi Java a causa della mancanza dello standard SQL

» JDBC risolve il problema definendo dei *propri tipi* mediante la classe `java.sql.Types`

Processare il risultato

String	CHAR, VARCHAR, or LONGVARCHAR
java.math.BigDecimal	NUMERIC
Boolean	BIT
Integer	INTEGER
Long	BIGINT
Float	REAL
Double	DOUBLE
byte[]	BINARY, VARBINARY, or LONGVARBINARY
java.sql.Date	DATE
java.sql.time	TIME
java.sql.Timestamp	TIMESTAMP
Clob	CLOB
Blob	BLOB
Array	ARRAY
Struct	STRUCT
Ref	REF
Java class	JAVA_OBJECT

Processare il risultato

» Esempio 1

```
public class TestStatement {
    private static final String DRIVER_NAME="com.mysql.jdbc.Driver";
    private static final String SQL = "SELECT * FROM title";

    public static void main( String[] args ) {
        try {
            Class.forName( DRIVER_NAME );

            Connection connection = DriverManager.getConnection(
                                args[ 0 ], args[ 1 ], args[ 2 ] );
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery( SQL );
```

……

Processare il risultato


```
      while ( resultSet.next() ) {
        System.out.println( "ID: " + resultSet.getInt( "title_id" ) );
        System.out.println( "Name: " + resultSet.getString( "name" ) );
        System.out.println( "Title Kind ID: " +
                                    resultSet.getInt( "title_kind_ID" ) );
        System.out.println( "Description: " +
                                    resultSet.getString( "description" ) );
        System.out.println( "Author: " + resultSet.getString( "author" ) );
        System.out.println( "Isbn: " + resultSet.getString( "isbn" ) );
        System.out.println( "-----------------------------------" );
      }
    }
    catch ( ClassNotFoundException e ) {
         e.printStackTrace();
    }
    catch ( SQLException e ) {
         e.printStackTrace();
    }
  }
}
```

Processare il risultato

» Esempio 2

```
public class TestPreparedStatement {
    private static final String DRIVER_NAME="com.mysql.jdbc.Driver";
    private static final String SQL =
                                "SELECT * FROM title WHERE name=?";

    public static void main( String[] args ) {
        try {
            Class.forName( DRIVER_NAME );

            Connection connection = DriverManager.getConnection(
                                args[ 0 ], args[ 1 ], args[ 2 ] );
            PreparedStatement preparedStatement =
                                connection.prepareStatement( SQL );
            preparedStatement.setString( 1, "name");
            ResultSet resultSet = preparedStatement.executeQuery();

……
```

Processare il risultato


```
        while ( resultSet.next() ) {
            System.out.println( "ID: " + resultSet.getInt( "title_id" ) );
            System.out.println( "Name: " + resultSet.getString( "name" ) );
            System.out.println( "Title Kind ID: " +
                                resultSet.getInt( "title_kind_ID" ) );
            System.out.println( "Description: " +
                                resultSet.getString( "description" ) );
            System.out.println( "Author: " +
                                resultSet.getString( "author" ) );
            System.out.println( "Isbn: " + resultSet.getString( "isbn" ) );
            System.out.println( "----------------------------------" );
        }

    }
    catch ( ClassNotFoundException e ) {
        e.printStackTrace();
    }
    catch ( SQLException e ) {
        e.printStackTrace();
    }
  }

}
```

Eliminare gli oggetti JDBC (1)

» Al termine delle operazioni è necessario *chiudere* tutti gli oggetti coinvolti

```
try {
   ……….
  resultset.close();
  statement.close();
  connection.close();
}
catch (SQLException ex) {
  ex.printStackTrace();
}
```

Eliminare gli oggetti JDBC (2)

» Esempio

```
public class TestPreparedStatementClose {
    private static final String DRIVER_NAME="com.mysql.jdbc.Driver";
    private static final String SQL =
                                "SELECT * FROM title WHERE name=?";

    public static void main( String[] args ) {
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            Class.forName( DRIVER_NAME );

            conn = DriverManager.getConnection(
                                args[ 0 ], args[ 1 ], args[ 2 ] );
            ps = con.prepareStatement( SQL );
            ps.setString( 1, "name");
            rs = ps.executeQuery();
.....
```

Eliminare gli oggetti JDBC (3)


```
        while ( rs.next() ) {
            System.out.println( "ID: " + rs.getInt( "title_id" ) );
            System.out.println( "Name: " + rs.getString( "name" ) );
            System.out.println( "Title Kind ID: " +
                                rs.getInt( "title_kind_ID" ) );
            System.out.println( "Description: " +
                                rs.getString( "description" ) );
            System.out.println( "Author: " +
                                rs.getString( "author" ) );
            System.out.println( "Isbn: " + rs.getString( "isbn" ) );
            System.out.println( "----------------------------------" );
        }

    }
    catch ( ClassNotFoundException e ) {
        e.printStackTrace();
    }
    catch ( SQLException e ) {
        e.printStackTrace();
    }
```

.....

Eliminare gli oggetti JDBC (4)


```
    finally {
       if (rs!=null) {
           try {
              rs.close();
           } catch (SQLException e) {/*Do Nothing*/}
       }
       if (ps!=null) {
           try {
              ps.close();
           } catch (SQLException e) {/*Do Nothing*/}
       }
       if (conn!=null) {
           try {
              conn.close();
           } catch (SQLException e) {/*Do Nothing*/}
       }
    }

  }
}
```