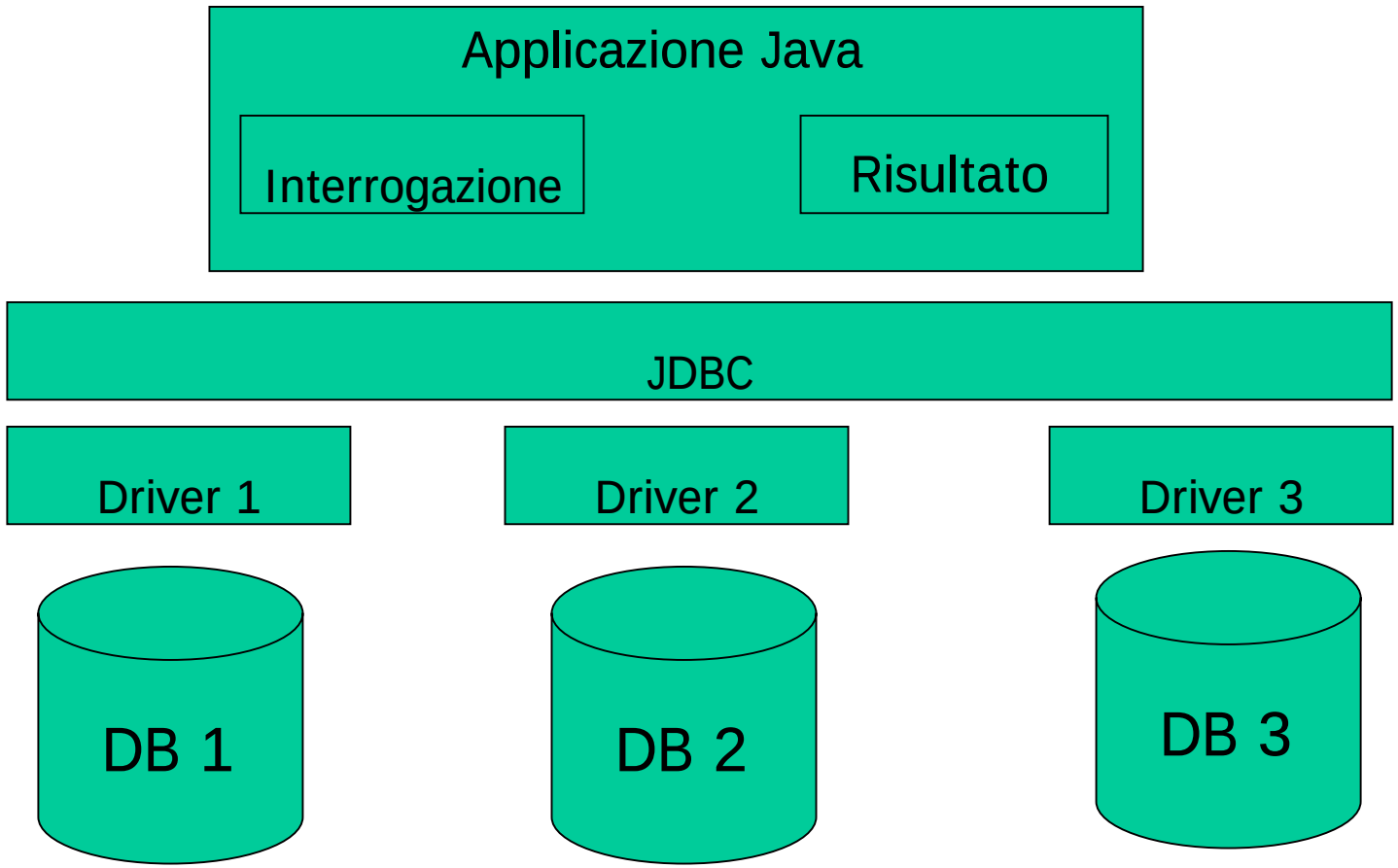
Introduzione a JDBC

- JDBC (Java Database Connectivity) è la parte delle API di J2SE che fornisce le primitive per la connessione a basi di dati relazionali:
  - si inviano comandi SQL;
  - si recuperano i risultati dei comandi.
- JDBC 3.0 è parte integrante di J2SE 1.4 (non J2EE!).
- Il package a cui faremo riferimento è **`java.sql`**.

Database e driver

- Ogni DB ha una sua API e un suo protocollo (implementato dal driver) particolari.
- JDBC astrae dalle particolarità a basso livello delle API e dei protocolli, fornendo un'interfaccia comune.
- I driver sottostanti si preoccupano poi di effettuare la traduzione dei comandi SQL nelle interfacce native dei database supportati.
- L'accesso al database può essere locale o remoto.

La funzione di JDBC



MySQL

- Negli esempi utilizzeremo il database MySQL (**`www.mysql.com`**) disponibile sia per Windows che per Linux e installato su **`latoserver.dimi.uniud.it`**.
- Driver usato:
  - MySQL Connector/J 3.0.9-stable (MM.MySQL)
  - È un file jar: bisogna ricordarsi di metterlo nel classpath
  - Per farlo funzionare con Tomcat deve essere presente in:
    - **`$TOMCAT_HOME/common/lib/`**
    - **`latoserver.dimi.uniud.it`**:
      - **`/home/tomcat/jakarta-tomcat-4.1.27/common/lib/`**

Interazione con MySQL da linea di comando

- Su **latoserver.dimi.uniud.it**:

```
$ mysql
…
mysql> use test_inventory
…
mysql> show tables;
+--------------------------+
| Tables_in_test_inventory |
+--------------------------+
| computers                |
| employees                |
| inventory                |
+--------------------------+
3 rows in set (0.00 sec)
```

Interazione con MySQL da linea di comando

```
mysql> select * from computers;
+------------+--------------------------+
| computerID | computerDescription      |
+------------+--------------------------+
|          1 | Dell Optiplex            |
|          2 | Dell Inspiron            |
|          3 | Dell Dimension           |
|          4 | iMac                     |
|          5 | Sun Ultra 1              |
|          6 | Gateway laptop           |
...
mysql > quit;
$ _
```

Procedura di connessione ed utilizzo di un database

- Caricare il gestore di driver
- Aprire la connessione
- Creare un oggetto istruzione
- Eseguire l’interrogazione (query)
- Elaborare i risultati
- Chiudere la connessione

MySQL e Java su `latoserver.dimi.uniud.it` (I)

- Per compilare ed eseguire correttamente programmi stand-alone che accedano a MySQL, procedere come segue:
  - compilare il programma con

```
javac –classpath
/home/tomcat/jakarta-tomcat-
4.1.27/common/lib/mysql-connector-
java-3.0.9-stable-bin.jar
<nomefile.java>
```

MySQL e Java su `latoserver.dimi.uniud.it` (II)

- Eseguire il programma con:

```
java –classpath /home/tomcat/jakarta-tomcat-4.1.27/common/lib/mysql-connector-java-3.0.9-stable-bin.jar:. <nomefile>
```

- Bisogna usare il classpath anche con l'interprete.
- Importante: il nome utente e la password devono essere vuoti (secondo e terzo parametro di **`getConnection()`**).

Importante: includere anche la directory corrente nel classpath!

Esempio di un programma Java standalone

```
import java.sql.*; // importiamo il package java.sql
public class provaJDBC {
  public static void main (String[] args) {
    try {
      Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
      System.err.println("Driver non trovato");
    }
    try {
      Connection c = DriverManager.getConnection(
        "jdbc:mysql://localhost/test_inventory",
        "","");
      Statement s = c.createStatement();
      ResultSet r = s.executeQuery(
                              "SELECT * FROM computers");
      while(r.next()){
        for (int i = 1; i <= 2; i++)
          System.out.print(r.getString(i) + " ");
        System.out.println();
      }
      r.close(); s.close(); c.close();
    }  catch (SQLException e) { System.err.println(e); }
  }
}
```

Utente e password vuoti per accedere al db di test

Compilazione ed esecuzione

- Compilazione:

```
javac –classpath /home/tomcat/jakarta-tomcat-4.1.27/common/lib/mysql-connector-java-3.0.9-stable-bin.jar provaJDBC.java
```

- Esecuzione:

```
java –classpath /home/tomcat/jakarta-tomcat-4.1.27/common/lib/mysql-connector-java-3.0.9-stable-bin.jar:. provaJDBC
```

Compilazione ed esecuzione impostando **$CLASSPATH**

- Impostazione della variabile d'ambiente:

  ```
  export CLASSPATH=/home/tomcat/jakarta-tomcat-4.1.27/common/lib/mysql-connector-java-3.0.9-stable-bin.jar:.
  ```

- Compilazione:

  ```
  javac provaJDBC.java
  ```

- Esecuzione:

  ```
  java provaJDBC
  ```

Output del programma

```
[scagnetto@latoserver temp]$ java -classpath /home/tomcat/jakarta-
   tomcat-4.1.27/common/lib/mysql-connector-java-3.0.9-stable-
   bin.jar:. provaJDBC
1 Dell Optiplex
2 Dell Inspiron
3 Dell Dimension
4 iMac
5 Sun Ultra 1
6 Gateway laptop
...
40 Commodore C64
41 Commodore Amiga
42 Sinclair Spectrum
43 Apple iBook
44 Atari ST
45 Silicon Graphics Octane
```

Analisi del programma

- Caricamento del gestore di driver:
```
Class.forName("com.mysql.jdbc.Driver");
```
- Apertura della connessione:
```
Connection c = DriverManager.getConnection(
"jdbc:mysql://localhost/test_inventory","","");
```
- Creazione di un'istruzione (statement):
```
Statement s = c.createStatement();
```
- Esecuzione della query:
```
ResultSet r = s.executeQuery("SELECT * FROM computers");
```
- Elaborazione dei risultati:
```
while(r.next()) {...};
```
- Chiusura della connessione e delle risorse aperte:
```
r.close(); s.close(); c.close();
```

Classi fondamentali di JDBC

- **`DriverManager`**
  - Gestore del driver
- **`Connection`**
  - Connessione al DB
- **`Statement`**
  - Istruzione SQL
- **`ResultSet`**
  - Risultato dell'interrogazione

DriverManager

- Implementa un gestore di driver
- Il metodo (statico) per ottenere una connessione al DB è:
  `static Connection getConnection(String,String,String)`
  che prende 3 argomenti:
  - URL, username, password
- Sintassi dell'URL:
  - **`jdbc:mysql://<host>[:<port>]/<dbname>`**
- Siccome le connessioni occupano risorse, in un ambiente multiutente e multitasking è opportuno adottare la seguente politica:
  - aprire una connessione solo quando necessario,
  - assicurarsi di chiuderla,
  - non aprire/chiudere connessioni inutilmente.

Connection

- Rappresenta una connessione al DB.
- Prima di fare qualunque cosa con un db, devo stabilire una connessione. Ad esempio, per creare un comando, devo disporre di una connessione (**`c`**):
  - **`Statement s = c.createStatement();`**
- Per chiudere la connessione, si utilizza il metodo
  - **`close()`**

Statement

- Rappresenta un'istruzione/comando (*statement*).
- Ha i metodi sia per eseguire un'interrogazione (*query*) SQL che restituisca un insieme di dati che per eseguire una query di aggiornamento/modifica del DB:
  - **`ResultSet executeQuery(String)`**
  - **`int executeUpdate(String)`**
- Quando non serve più va chiuso con il metodo:
  - **`close`**

ResultSet

- Rappresenta il risultato di un'interrogazione. Tramite il metodo **`next()`** possiamo spostarci da un record al successivo:
  - **`boolean next()`**
- Abbiamo a disposizione una serie di metodi per recuperare i dati memorizzati nei vari campi di ogni record, in base al tipo di dato originario del DB:
  - **`String getString(int)`** [gli indici partono da 1!]
  - **`getXxx()`** [**`Byte`**, **`Boolean`**, **`Blob`**, …]
- E' buona norma chiudere il ResultSet quando non serve più con il metodo:
  - **`close`**

JDBC nelle applicazioni Web

- Praticamente tutte le applicazioni Web non banali sono “alimentate” da una base di dati.
- Le pagine sono generate dinamicamente a partire da contenuti presenti in un database.
- Vantaggi di un DB rispetto a un repositorio di file:
  - velocità delle operazioni di accesso;
  - consistenza dei dati;
  - sicurezza,
  - …

Utilizzo di JDBC da JSP (I)

```
<html>
  <head>
      <title>JDBC</title>
  </head>
  <body>
    <h1>JDBC</h1>
    <%@ page import="java.sql.*" %>
    <table border="1">
    <% try {
         Class.forName("com.mysql.jdbc.Driver");
       } catch (ClassNotFoundException e) {
         out.println("Driver non trovato" + e);
       }
```

Utilizzo di JDBC da JSP (II)

```
    try {
      Connection c =
        DriverManager.getConnection(
        "jdbc:mysql://localhost/test_inventory","","");
      Statement s = c.createStatement();
      ResultSet r = s.executeQuery("SELECT * FROM computers");
      ResultSetMetaData md = r.getMetaData();
      while(r.next()){
        out.print("<tr>");
        for (int i = 1; i <= md.getColumnCount(); i++)
          out.print("<td>" + r.getString(i) + "</td> ");
        out.println("</tr>");
      }
      r.close(); s.close(); c.close();
    }  catch (SQLException e) {
      out.println(e);
    }
    %>
    </table>
  </body>
</html>
```

Esercizi

- Provare a scrivere una servlet equivalente alla pagina JSP che stampa la tabella degli impiegati.
- Provare ad eseguire query diverse (su differenti tabelle o selezionando un insieme differente di campi) per testare le classi di **`java.sql`**.