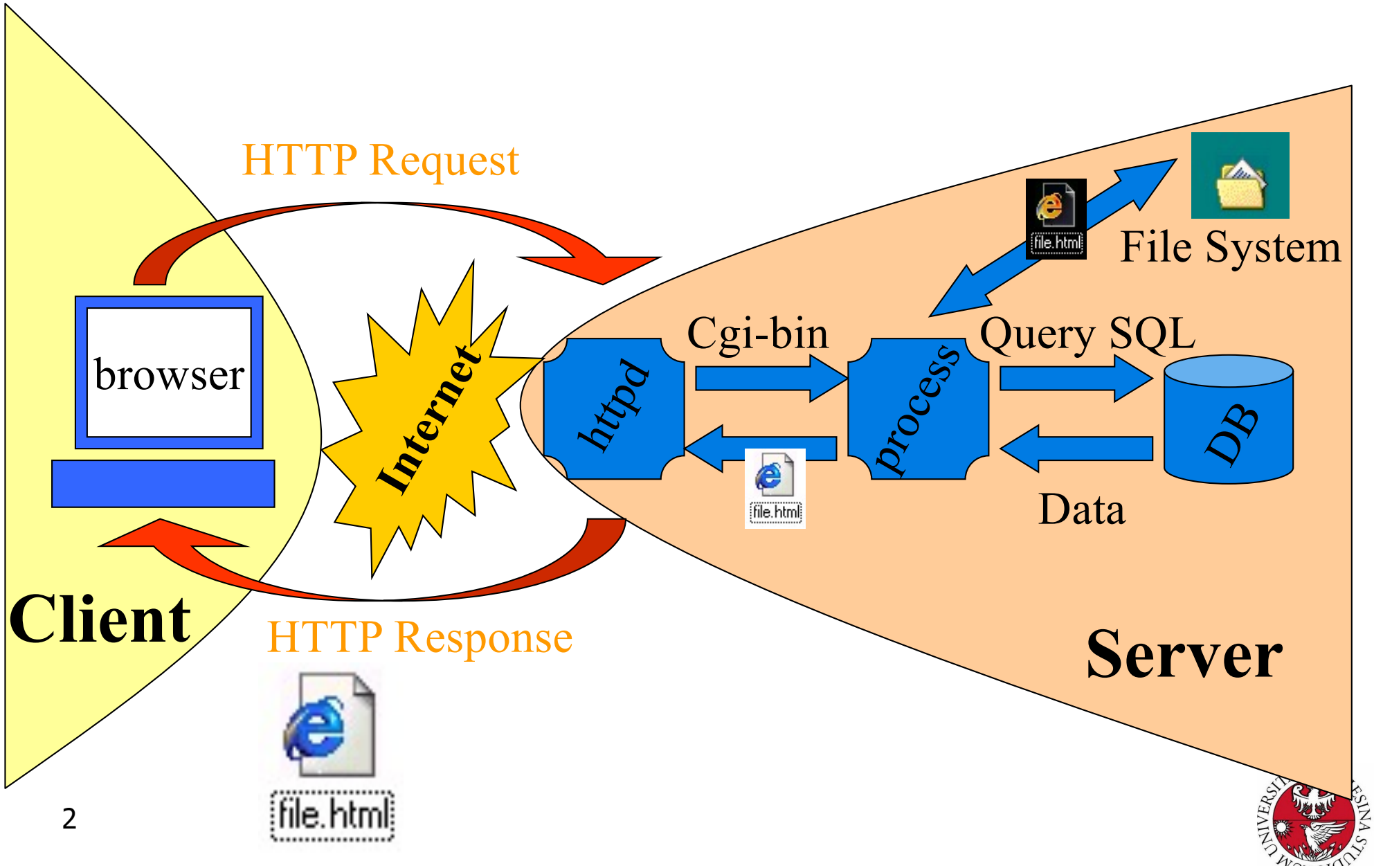
Access to DB

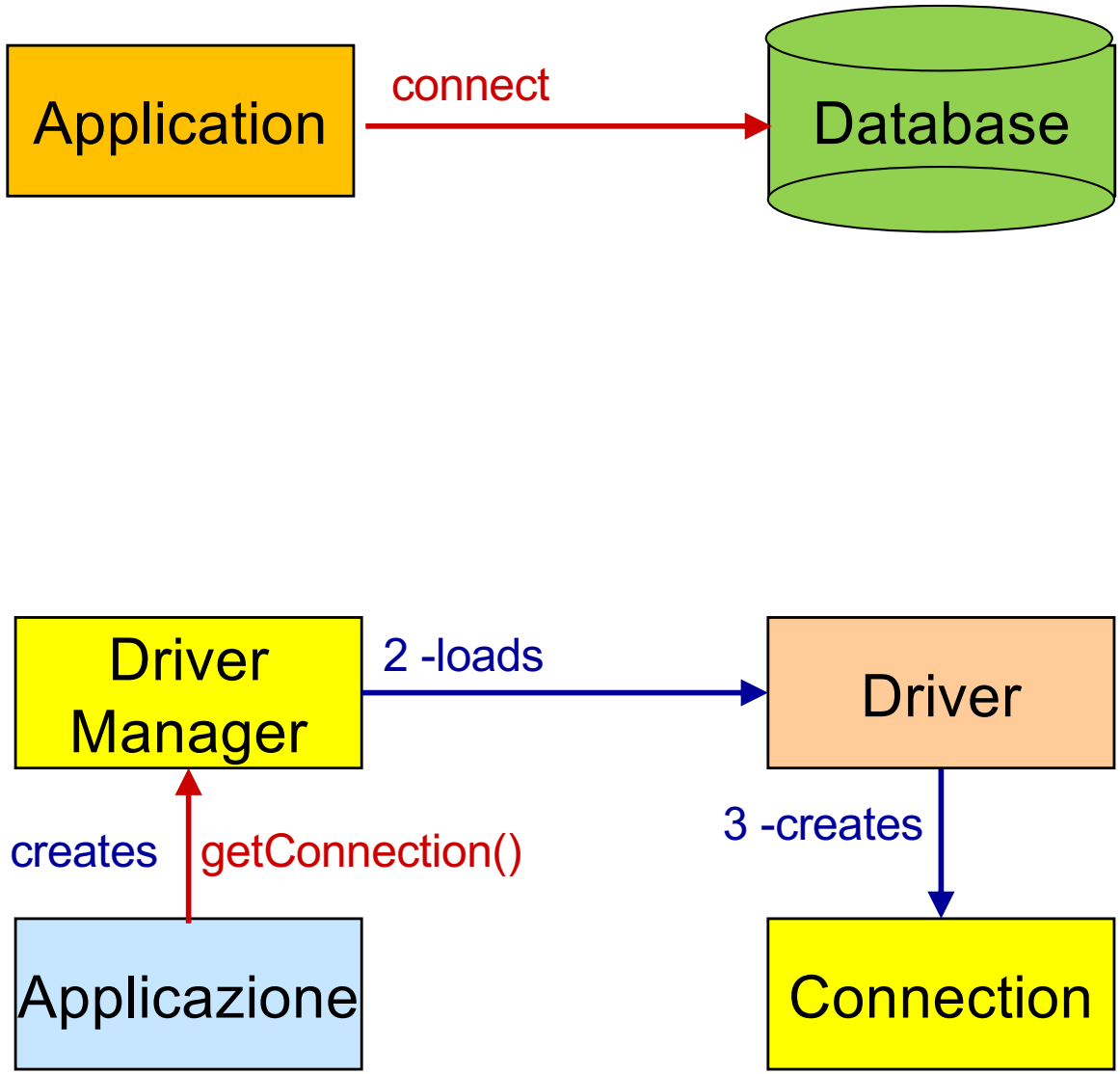
Using JDBC in servlets


HTTP Request
browser
Internet
Client
HTTP Response
file.html
Cgi-bin
Query SQL
File System
file.html
httpd
process
DB
file.html
Data
Server

Q

How do I access a database from Java?

The java.sql Object Model



Reminder: Class.forName

static Class **forName**(String className)

Returns the Class object associated with the class or interface with the given string name.

Typical use:

```
Object o=Class.forName("java.lang.String").newInstance();
```

is equivalent to:

```
Object o=new String();
```

JDBC – Steps – 1 – Get the driver

```
Class.forName("org.apache.derby.jdbc.ClientDriver");
```

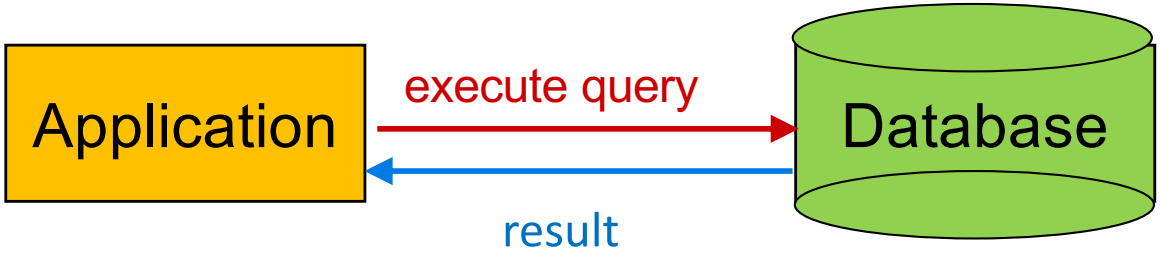
JDBC – Steps – 2 - LOAD THE DRIVER

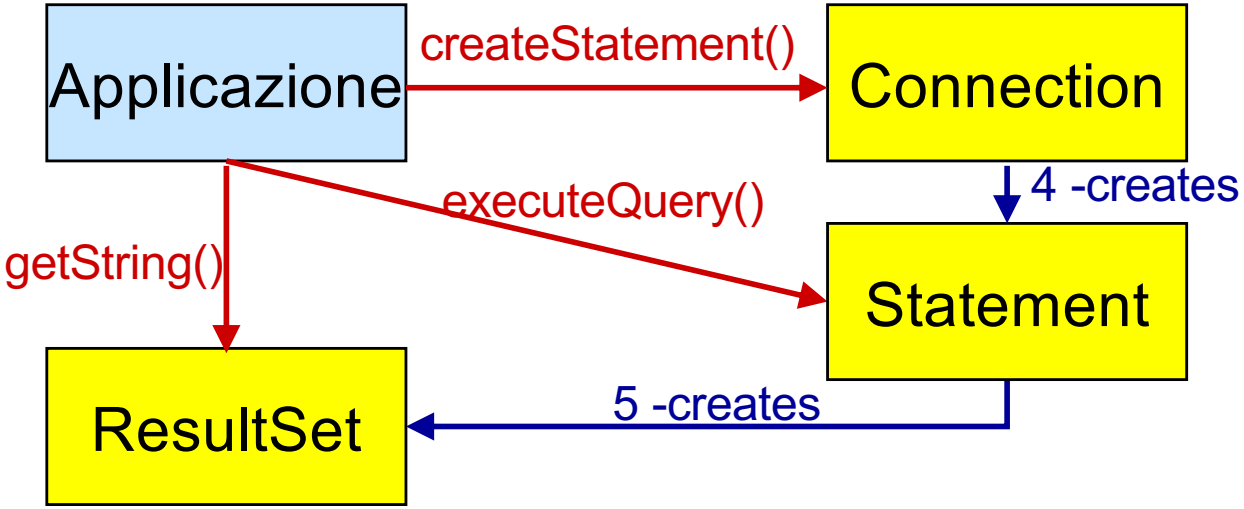
```
Connection con = DriverManager.getConnection(url, "myLogin",
"myPassword");
```

If you are using a JDBC driver developed by a third party, the documentation will tell you what subprotocol to use, that is, what to put after jdbc: in the JDBC URL. For example, if the driver developer has registered the name acme as the subprotocol, the first and second parts of the JDBC URL will be jdbc:acme: . The driver documentation will also give you guidelines for the rest of the JDBC URL. This last part of the JDBC URL supplies information for identifying the data source.

```
example: String dbURL = "jdbc:derby://localhost:1527/DemoDB";
```

The java.sql Object Model





JDBC – Steps – 3 CREATE STATEMENT

A Statement object is what sends your SQL statement to the DBMS.

For a SELECT statement, the method to use is executeQuery .
For statements that create or modify tables, the method to use is executeUpdate.

```
Statement stmt = con.createStatement();
stmt.executeUpdate("CREATE TABLE COFFEES " +
    "(COF_NAME VARCHAR(32), SUP_ID INTEGER, PRICE FLOAT, " +
    "SALES INTEGER, TOTAL INTEGER)");
```

Typically you would put the SQL statement in a String (called let's say createTableCoffees), and then use

```
stmt.executeUpdate(createTableCoffees);
```

JDBC – Steps – 4 RETRIEVING VALUES

JDBC returns results in a ResultSet object.

```
ResultSet rs = stmt.executeQuery( "SELECT COF_NAME, PRICE FROM COFFEES");
```

In order to access the names and prices, we will go to each row and retrieve the values according to their types. The method next moves what is called a cursor to the next row and makes that row (called the current row) the one upon which we can operate. Since the cursor is initially positioned just above the first row of a ResultSet object, the first call to the method next moves the cursor to the first row and makes it the current row. Successive invocations of the method next move the cursor down one row at a time from top to bottom. Note that with the JDBC 2.0 API, you can move the cursor backwards, to specific positions, and to positions relative to the current row in addition to moving the curs or forward.

```
String query = "SELECT COF_NAME, PRICE FROM COFFEES"; ResultSet rs = stmt.executeQuery(query);
while (rs.next()) {
        String s = rs.getString("COF_NAME");
        float n = rs.getFloat("PRICE");
        System.out.println(s + " " + n);
}
```

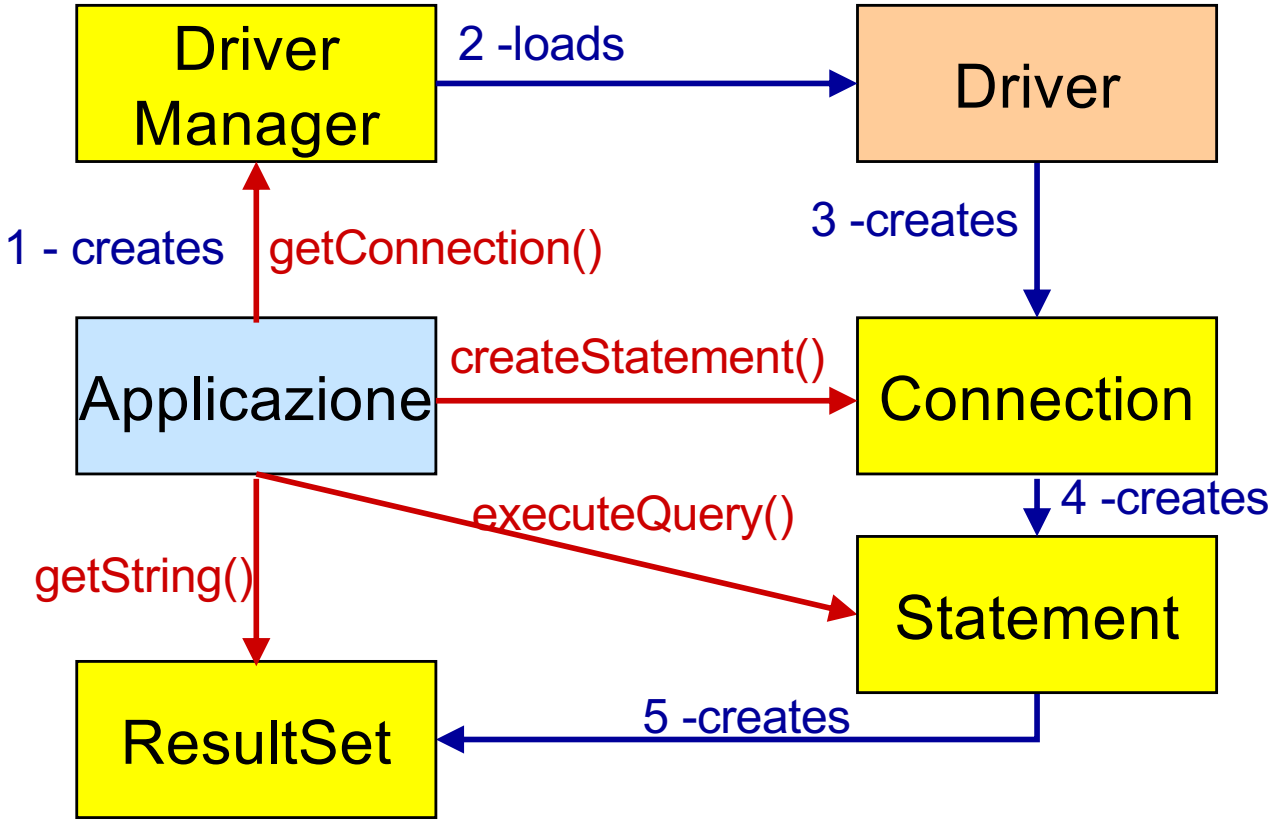
JDBC – Installation and usage (sum up)

1) Install a driver on your machine.
Your driver should include instructions for installing it. For JDBC drivers written for specific DBMSs, installation consists of just copying the driver onto your machine; there is no special configuration needed. .

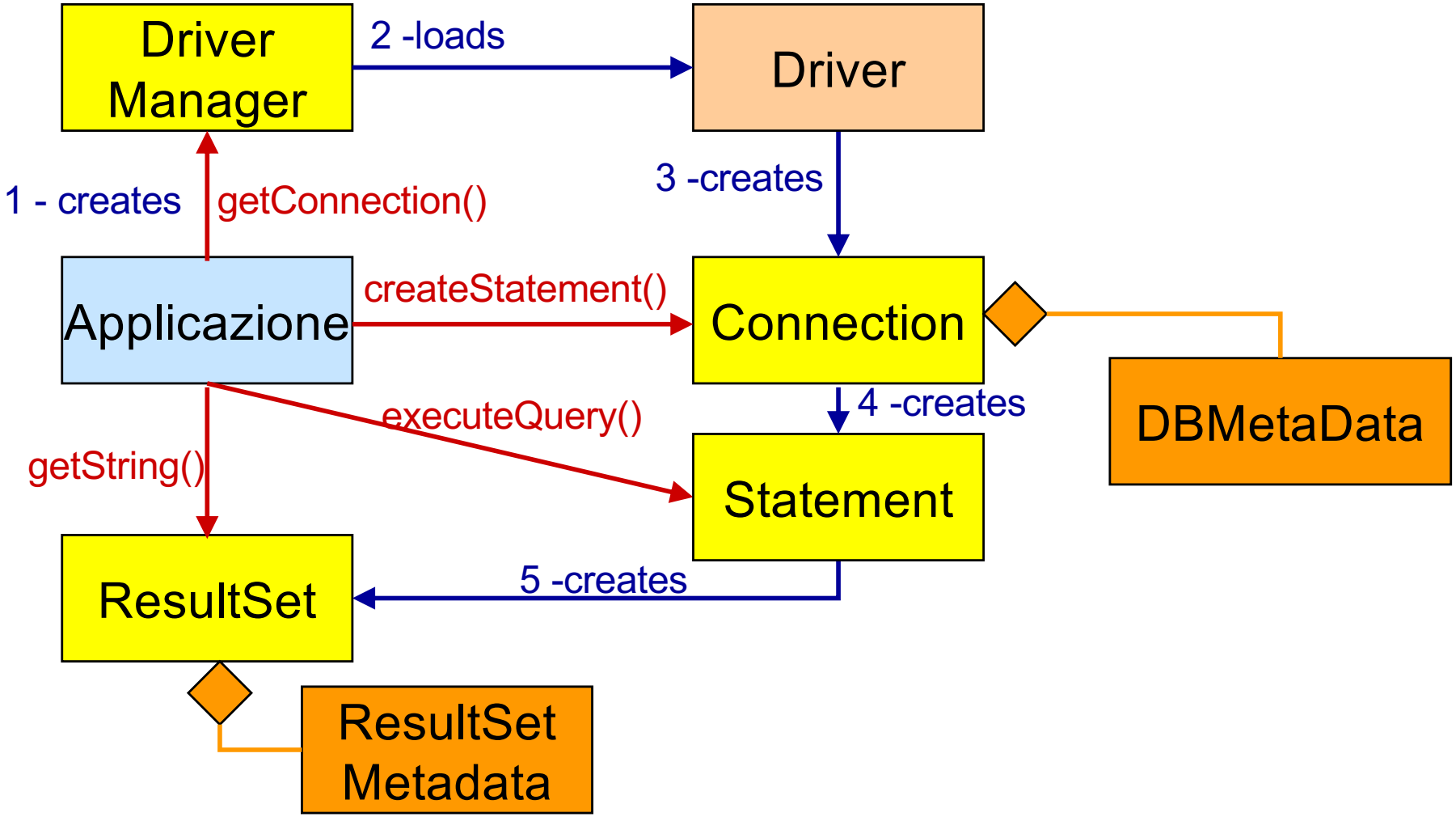
A) Load the driver.
B) Open a connection.
C) Create Statement.
D) Retrieve Values.

Always catch exceptions!
JDBC lets you see the warnings and exceptions generated by your DBMS and by the Java compiler. To see exceptions, you can have a catch block print them out. .

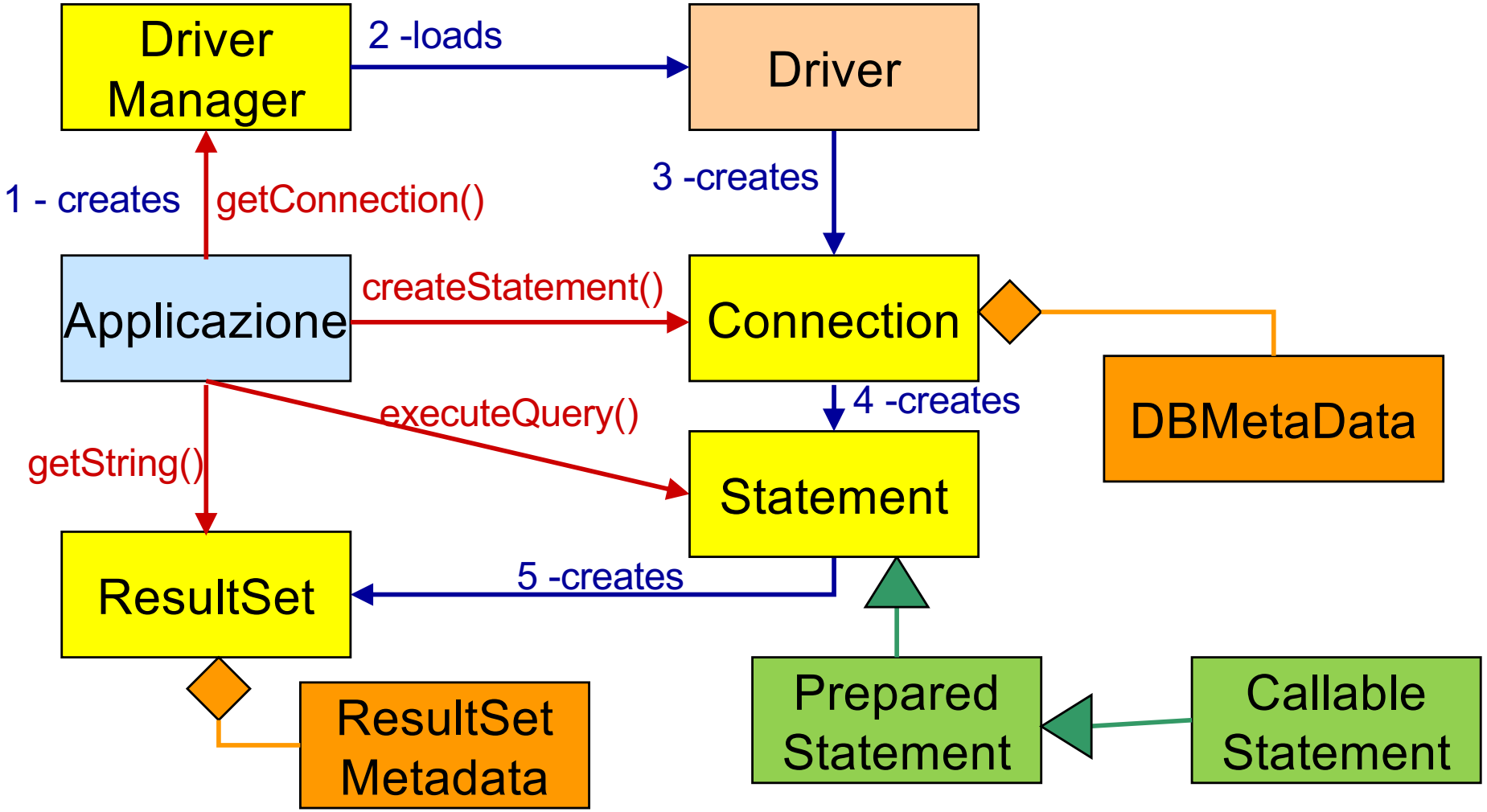
The java.sql Object Model



The java.sql Object Model: Metadata



The java.sql Object Model: predefined statement



JDBC – Prepared statements

If you want to execute a Statement object many times, it will normally reduce execution time to use a PreparedStatement object instead.

The main feature of a PreparedStatement object is that, unlike a Statement object, it is given an SQL statement when it is created.

The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled. As a result, the PreparedStatement object contains not just an SQL statement, but an SQL statement that has been precompiled.
This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement 's SQL statement without having to compile it first.

```
PreparedStatement updateSales = con.prepareStatement( "UPDATE COFFEES
SET SALES = ? WHERE COF_NAME LIKE ?");
updateSales.setInt(1, 75);
```

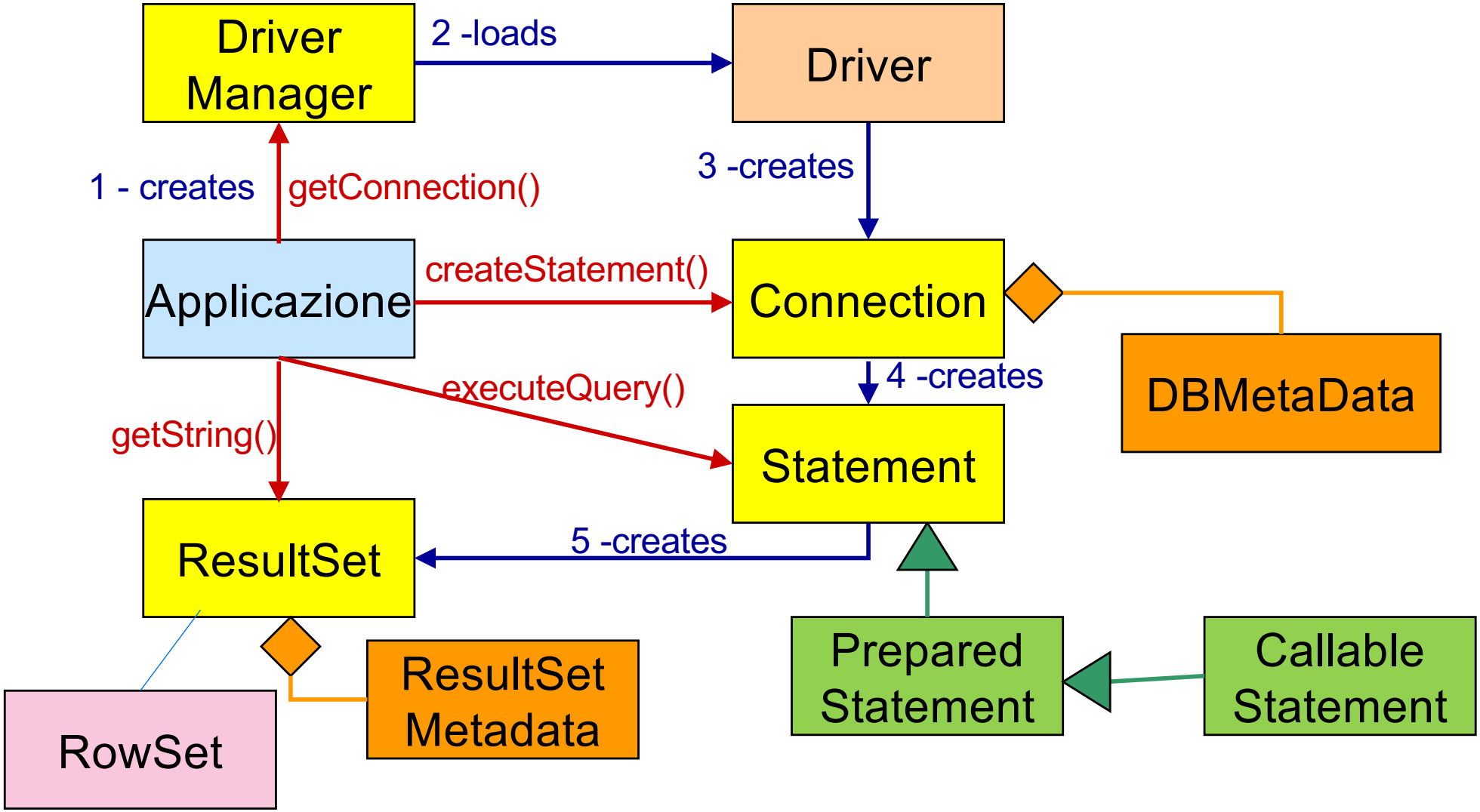
JDBC – Callable statements

A **stored procedure** is a group of SQL statements that form a logical unit and perform a particular task. Stored procedures are used to encapsulate a set of operations or queries to execute on a database server. For example, operations on an employee database (hire, fire, promote, lookup) could be coded as stored procedures executed by application code. Stored procedures can be compiled and executed with different parameters and results, and they may have any combination of input, output, and input/output parameters.
Stored procedures are supported by most DBMSs, but there is a fair amount of variation in their syntax and capabilities.

If you want to call stored procedures, you must use a CallableStatement (subclass of PreparedStatement).

WARNING: stored procedures move the business logic WITHIN THE DB!

The java.sql Object Model: RowSet

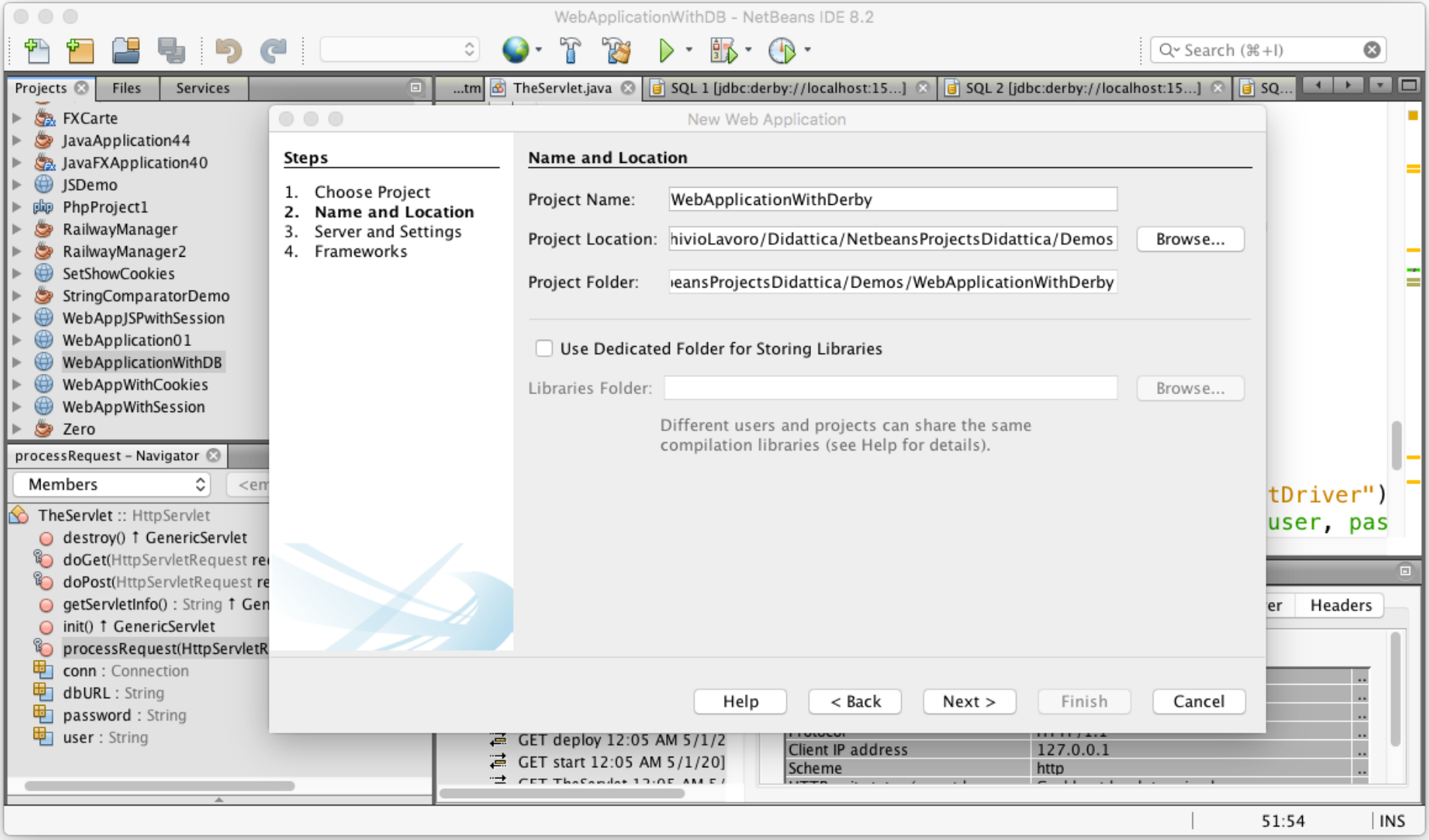


for RowSet see https://docs.oracle.com/javase/tutorial/jdbc/basics/jdbcrowset.html

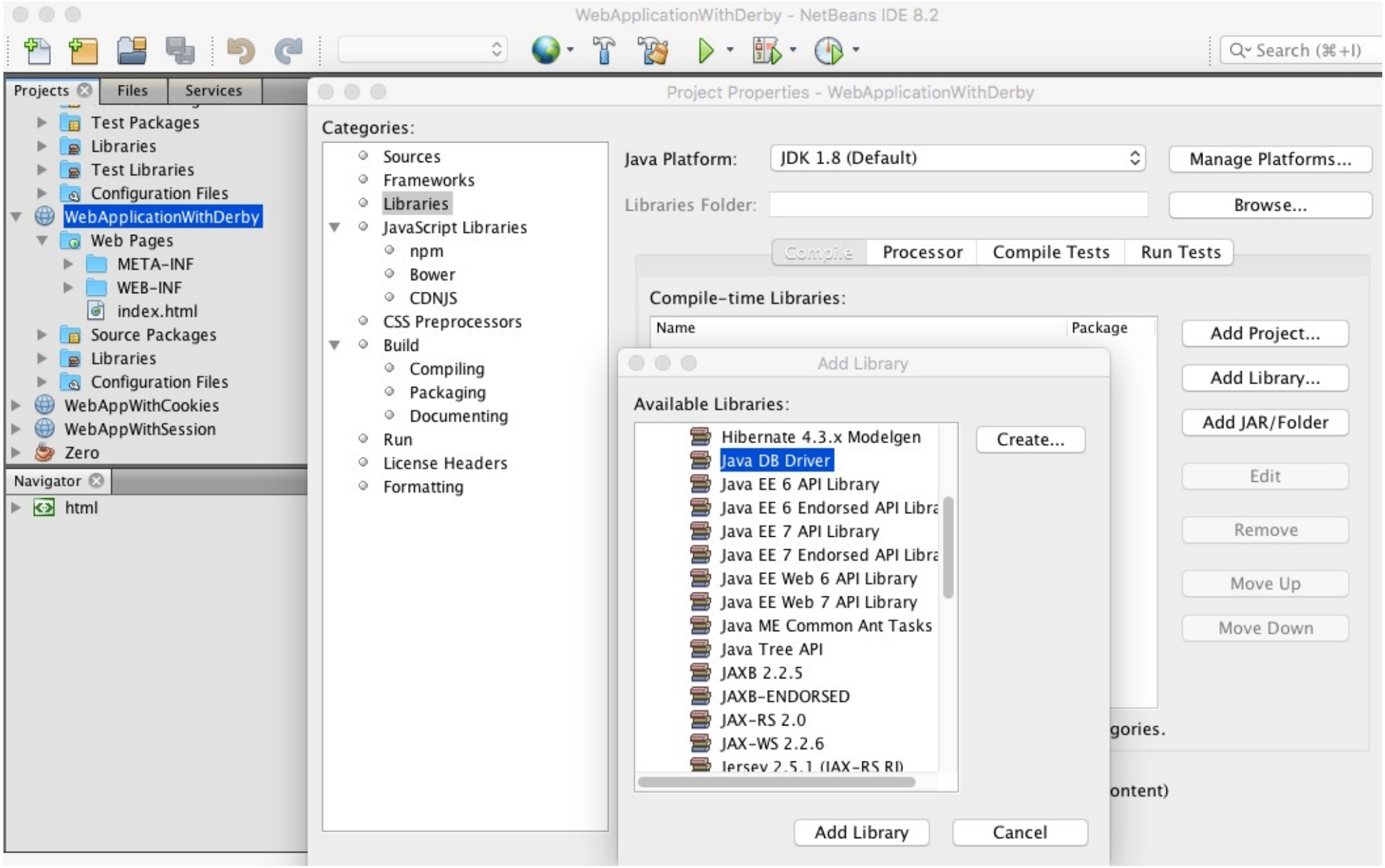
Q

What do I create a Netbeans project that uses a DB?

Netbeans configuration – 1 Create WebApp

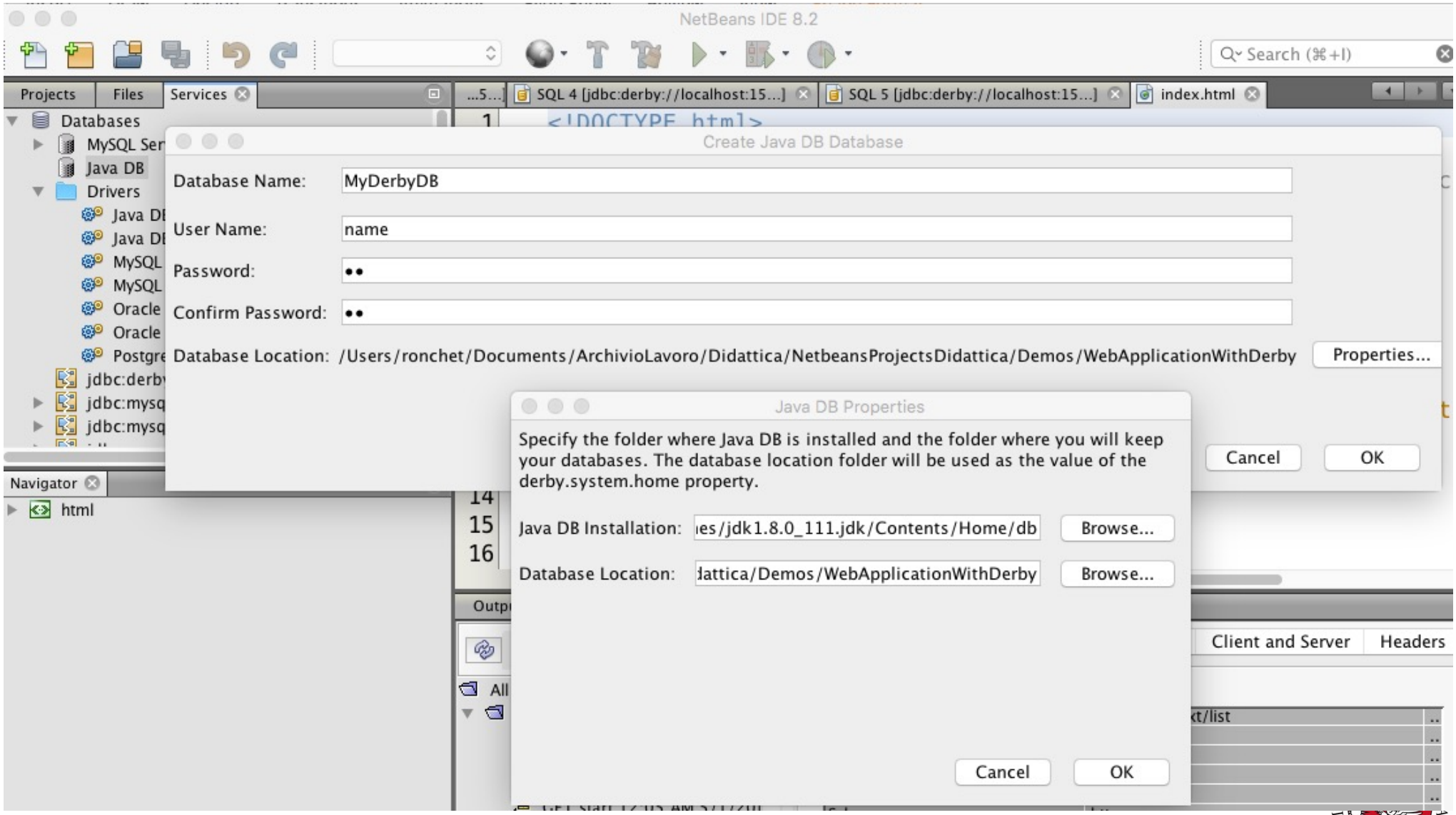


Netbeans configuration – 2
in Project Properties add Library "Java DB"

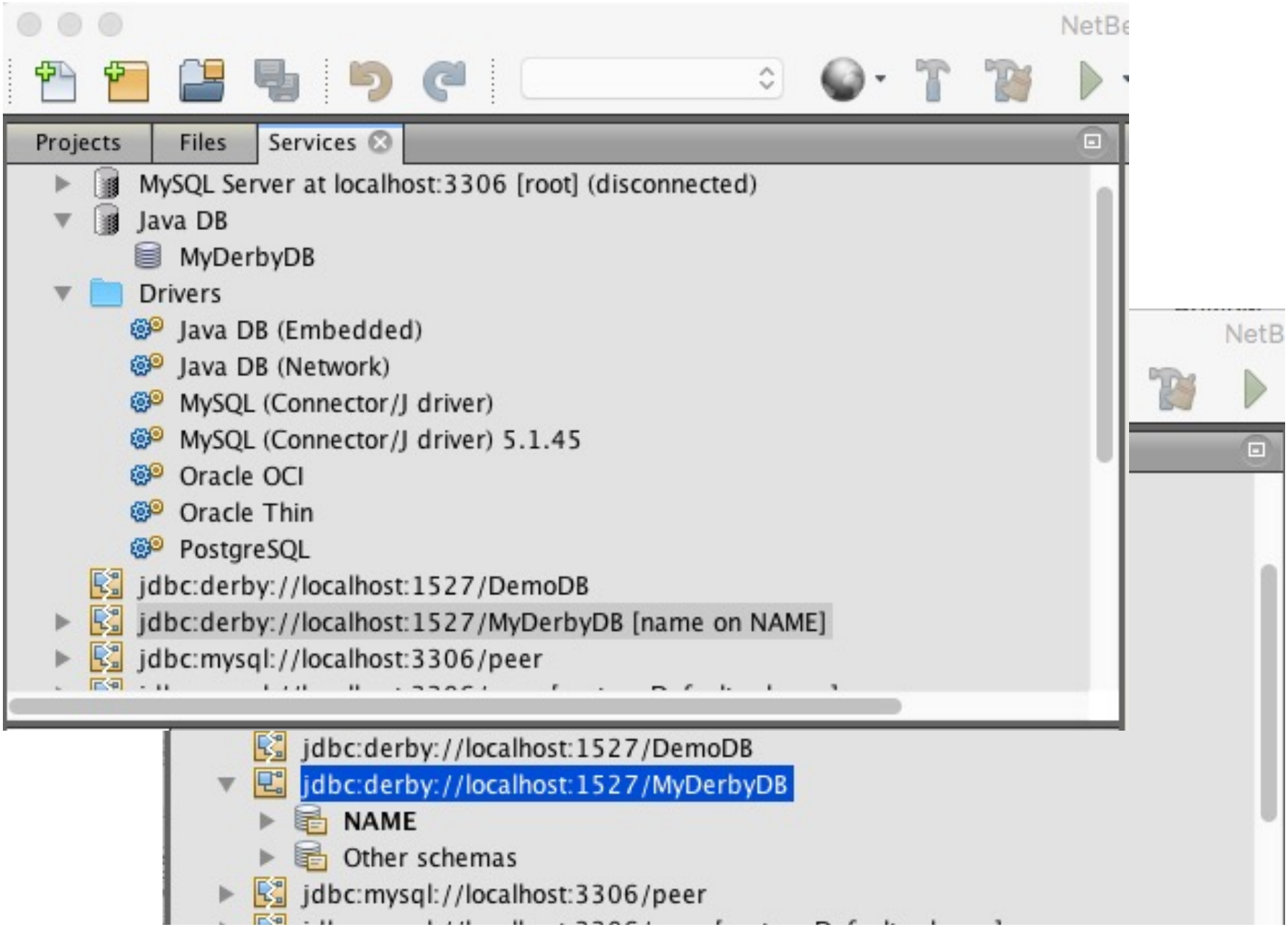


Netbeans configuration – 3
in Services create Database

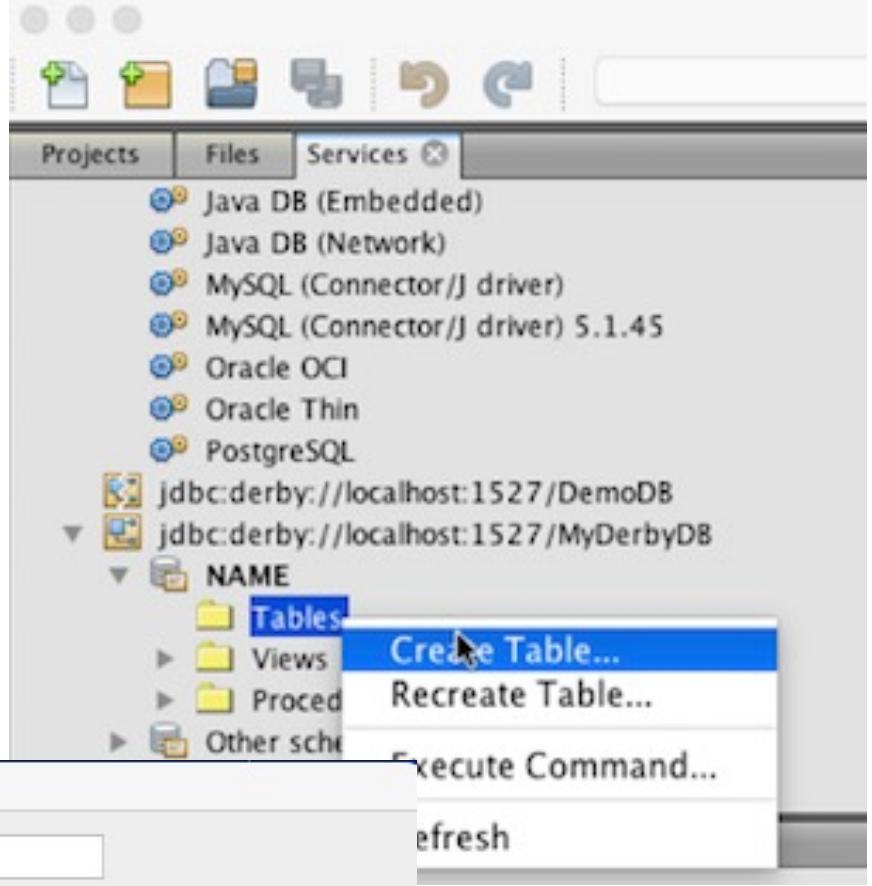
right cllck on JavaDB, choose "create", add requested info, click on properties and make the DB Location within your project

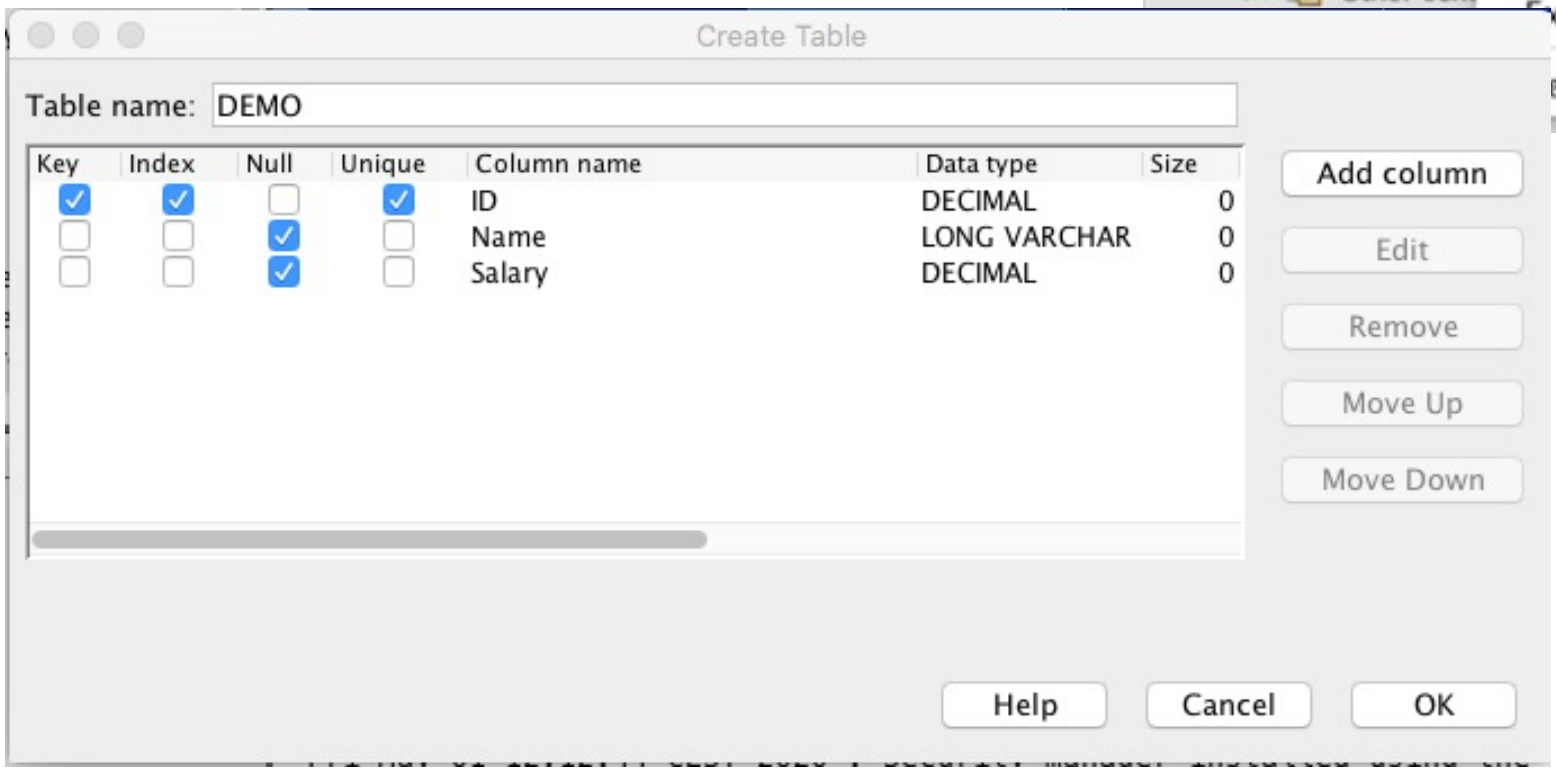


Netbeans configuration – 4
rename the created connection (if needed), right-click it and "connect"



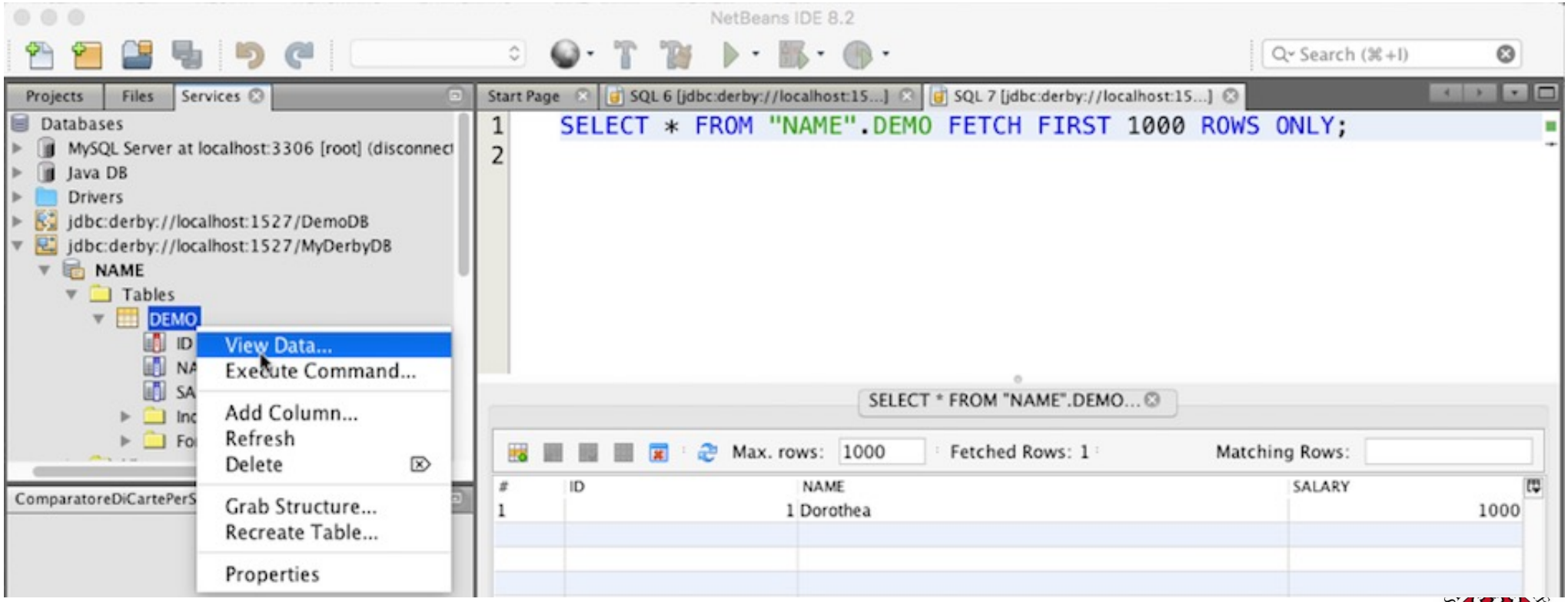
Netbeans configuration – 5 create table





Netbeans configuration – 6 populate DB

right click on command and choose "Run Statement"



Right click on "DEMO " and select "Show Dada"

Create servlet

Go to the project, and create a servlet called "TheServlet"

Edit it as shown in the next slides

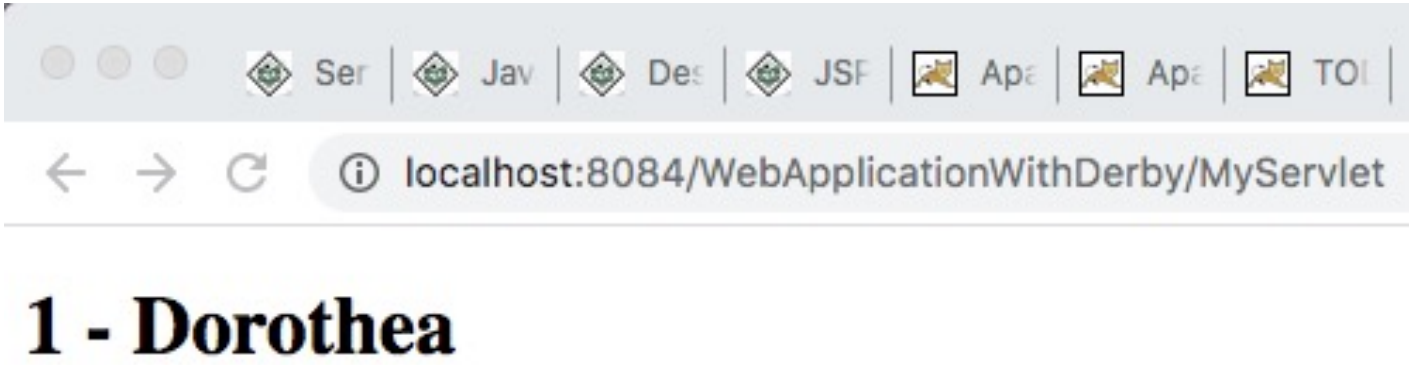
Example

```
@WebServlet(urlPatterns = {"/TheServlet"})
public class TheServlet extends HttpServlet {

    String dbURL = "jdbc:derby://localhost:1527/MyDerbyDB";
    String user = "name";
    String password = "pw";
    Connection conn = null;

    @Override
    public void init() {
        try {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            conn = DriverManager.getConnection(dbURL, user, password);
        } catch (ClassNotFoundException | SQLException ex) {
            ex.printStackTrace();
        }
    }
    @Override
    public void destroy() {
        try {
            conn.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
```

Example

```
protected void processRequest(HttpServletRequest request,
     HttpServletResponse response) throws ServletException, IOException {
   response.setContentType("text/html;charset=UTF-8");
   StringBuffer dbOutput = new StringBuffer("<h1>");
   try {
     Statement stmt = conn.createStatement();
     String sql = "SELECT ID, NAME FROM DEMO";
     ResultSet results = stmt.executeQuery(sql);
     while (results.next()) {
        dbOutput.append(results.getString(1)).append(" - ");
        dbOutput.append(results.getString(2)).append("</h1>");
     }
   } catch (SQLException ex) {
        dbOutput.append(ex.toString()).append("</h1>");
   }
   try (PrintWriter out = response.getWriter()) {
      out.println("<!DOCTYPE html><html><head>");
      out.println("<title>Servlet TheServlet</title>");
      out.println("</head><body>");
      out.println(dbOutput.toString());
      out.println("</body><html>");
         }
    }
… doGet, doPost, getServletInfo: leave them as they are
}
```

Run file…


localhost:8084/WebApplicationWithDerby/MyServlet
1 - Dorothea

Q

What is the best way to manage DB connections?

Connection management

We created the connection in the init method, and closed it in the destroy ("per Servlet connection"). Is this a good idea?

Alternatives:

create the connection in the doXXX (or processRequest) method ("per Request connection")

perServlet:

- many connections simultaneously open
- concurrency bottleneck (Connection's methods are synchronozed)

perRequest

- lots of open/close (slow!)

Connection management

We could create "per Session" connection.

perSession:

- every user has one connection, and reuses it
- potentially many connections , with low usage each
- sessions reman alive as long as the connection lives
- you should use HttpSessionBinding interface to monitor the closing of sessions due to timeout

Connection pooling

- servlets share a set of existing connection
- more complex
- infrastructures exist to allow it

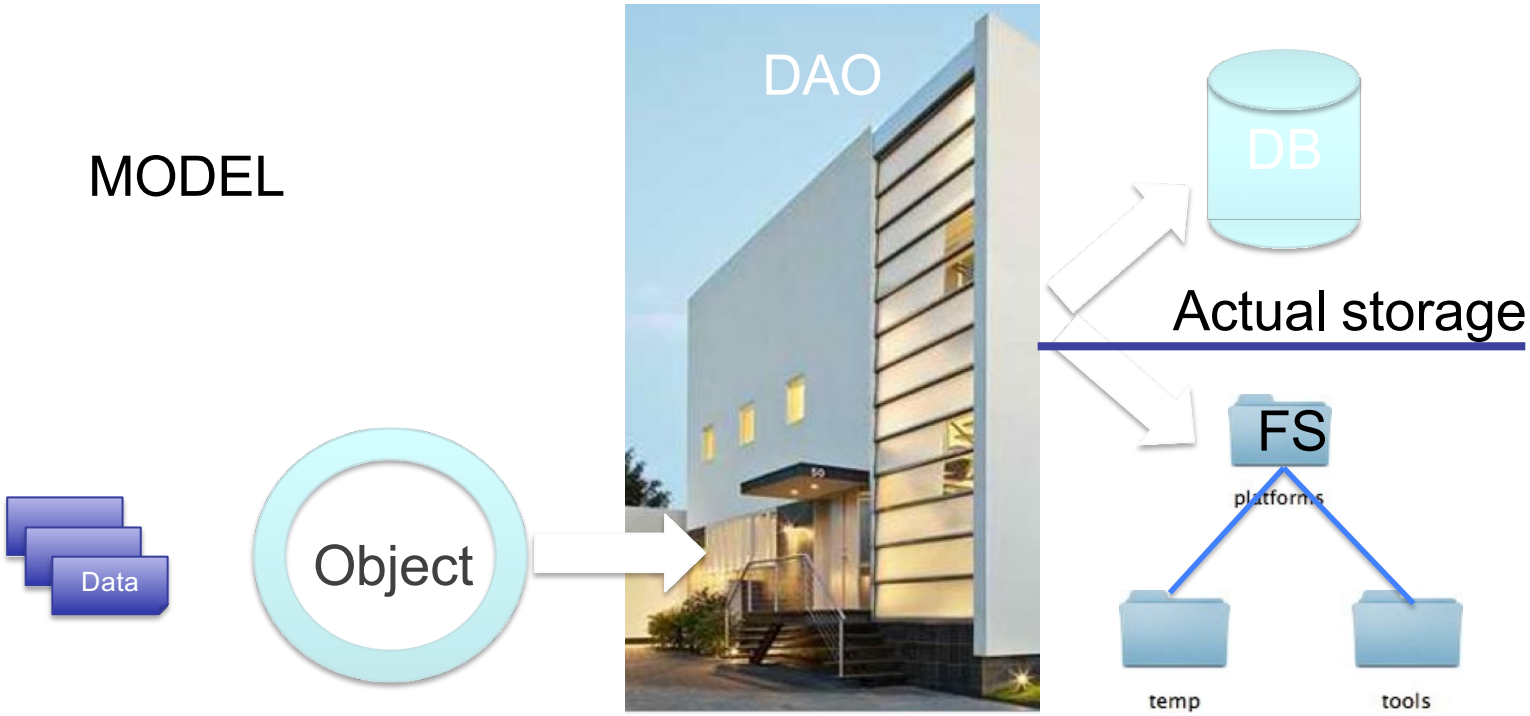
(Yet another possibility would have been "one connection per Web App". How could you have implemented it? What are its advantages and disadvantages?)

Connection management

In depth discussion, with examples

https://www.oreilly.com/library/view/java-programming-with/059600088X/ch04s02.html

Data Access Object



References about jdbc

https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

https://www.journaldev.com/2471/jdbc-example-mysql-oracle

https://www.tutorialspoint.com/servlets/servlets-database-access.htm

Extra references

if you need to refresh your SQL:

https://www.w3schools.com/sql/default.asp

If you need to install JavaDB (Derby)

https://www.codejava.net/java-se/jdbc/how-to-get-started-with-apache-derby-javadb