Java Programming
Unit 12

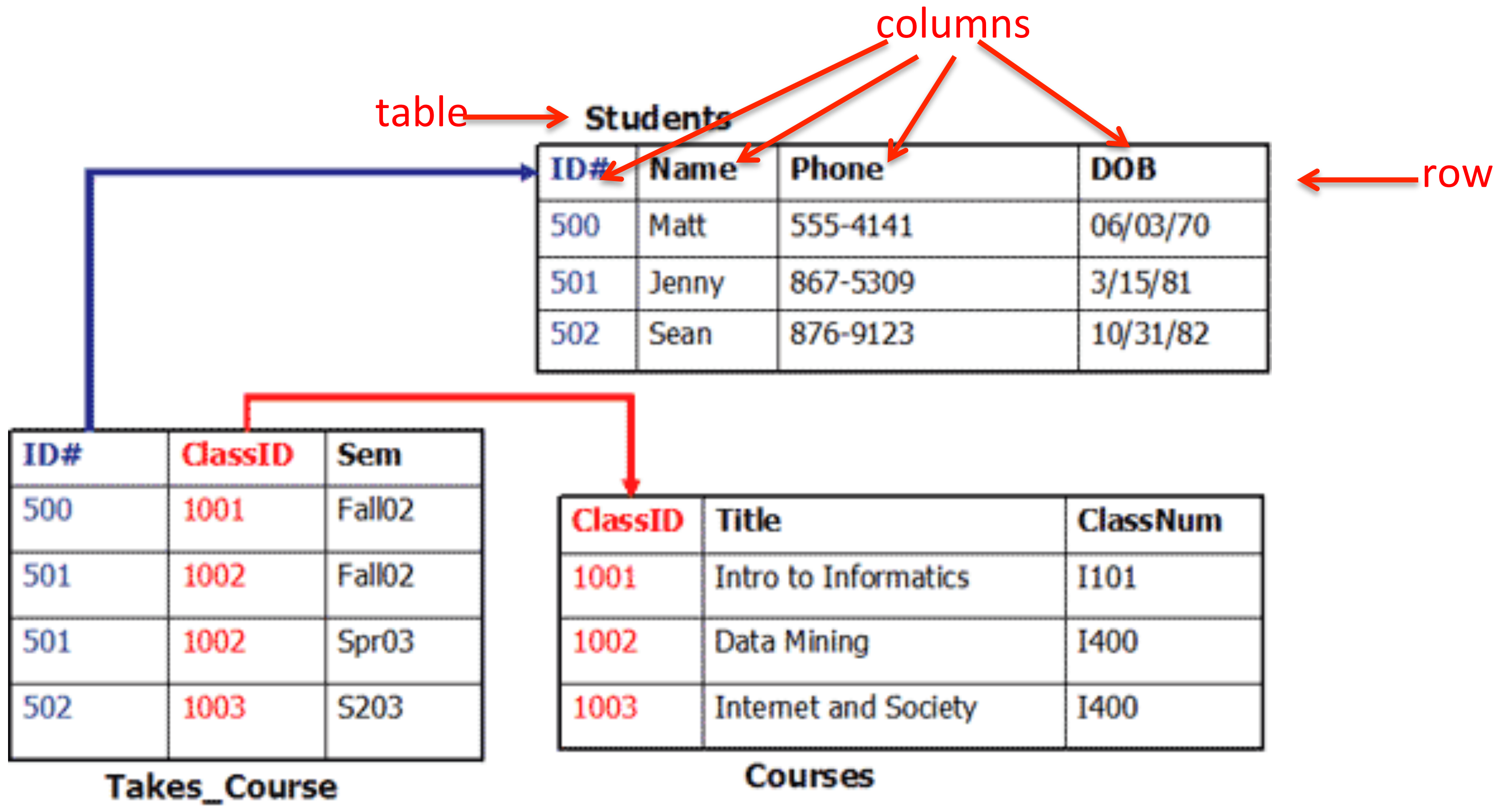
Working with Relational Databases using JDBC

Relational Database Management Systems

RDBMS store data in *tables* as *rows* and *columns*.

One row is one record (e.g. Students, Courses, Customers, Orders).

A column represents a property of the record (e.g. name, address, price).



Students

ID#	Name	Phone	DOB
500	Matt	555-4141	06/03/70
501	Jenny	867-5309	3/15/81
502	Sean	876-9123	10/31/82

Takes_Course

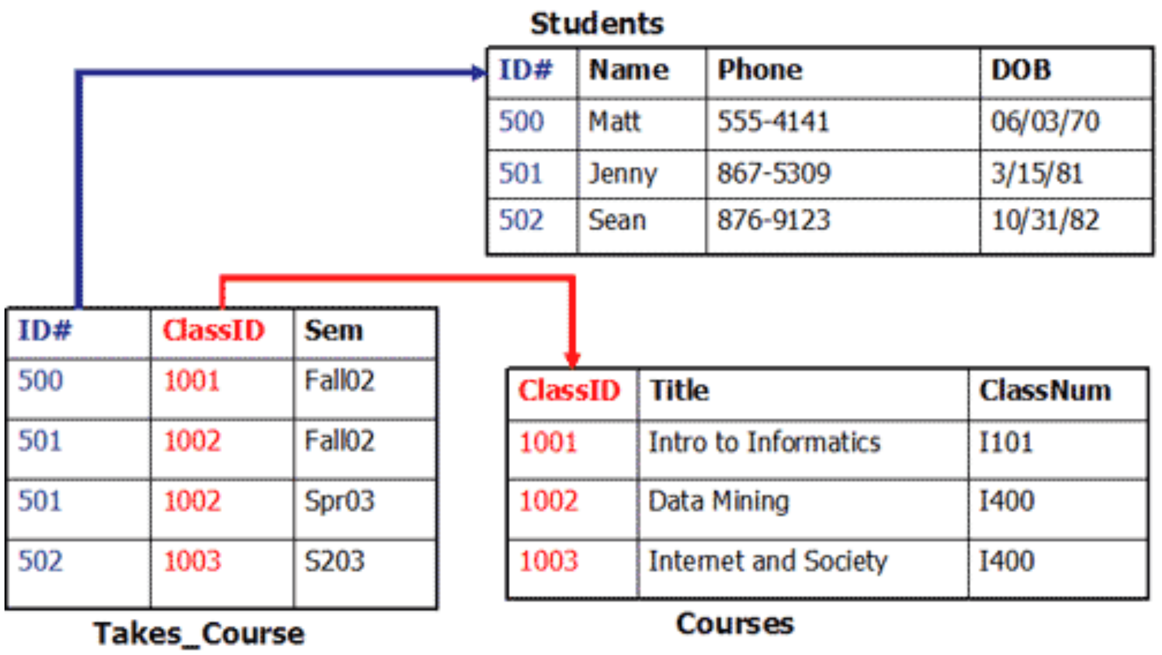
ID#	ClassID	Sem
500	1001	Fall02
501	1002	Fall02
501	1002	Spr03
502	1003	S203

Courses

ClassID	Title	ClassNum
1001	Intro to Informatics	I101
1002	Data Mining	I400
1003	Internet and Society	I400

Structured Query Language (SQL)

```
1.  Select * from Students
2.  Select phone from Students where name = "Matt"
3.  Select name, classid from Students, Takes_Course
           where Students.ID = Takes_Course.ID and Name="Matt"
4.  Select Name, Title from Students, Takes_Course, Courses
            where  Students.ID = Takes_Course.ID
                  and Takes_Course.classid = Courses.classid
5.  Select count(*) from Takes_Course where classid=1002
```



Students

ID#	Name	Phone	DOB
500	Matt	555-4141	06/03/70
501	Jenny	867-5309	3/15/81
502	Sean	876-9123	10/31/82

ID#	ClassID	Sem
500	1001	Fall02
501	1002	Fall02
501	1002	Spr03
502	1003	S203

Takes_Course

ClassID	Title	ClassNum
1001	Intro to Informatics	I101
1002	Data Mining	I400
1003	Internet and Society	I400

Courses

Popular RDBMS

The most popular Relational DBMS's are Oracle, DB2, Microsoft SQL Server, and MySQL Server.

We'll use JavaDB (a.k.a. Derby DB), which is included with your JDK.

Java contains classes required for work with DBMS in packages java.sql and javax.sql.

Non-relational DBMS don't store data as rows and columns. They're known as NoSQL databases (e.g. MongoDB, Cassandra, Couchbase et al). See http://nosql-database.org.

JDBC

JDBC is an API for working with all RDBMS using the same Java classes, e.g. `Connection`, `Statement`, and `ResultSet` .

Embedded in Java SQL is converted by JDBC driver into a form that DBMS understands.

An alternative way of working with DBMS is by using object-relational mapping (ORM) frameworks (e.g. Hibernate), or Java Persistense API (JPA) ORM frameworks map *Java entities* to corresponding DBMS tables.

Another approach is to map a *result set* produced by SQL to a Java objects, e.g. MyBatis framework.

Four types of JDBC Drivers

- **Type 1** driver is a JDBC-ODBC bridge that enables Java programs to work with the database using ODBC drivers from Microsoft. Windows only.

- **Type 2** driver: native drivers (C, C++) are wrapped in Java. These must be installed on the computer that runs client Java program accessing DBMS.

- **Type 3** driver consists of two parts: the client portion relays a DBMS independent SQL to middleware server, which then translates it to a specific DBMS protocol by the server portion of the driver.

- **Type 4** driver is a pure Java thin driver, which comes as a .jar file and performs direct calls to the database server. It does not need any configuration on the client's machine.

JavaDB (a.k.a. Derby DB)

If you installed JDK as instructed in Lesson 1, you already have Derby DB under *c:\Program Files\Sun\JavaDB*

If you don't have it there or use MAC OS, install it from http://db.apache.org/derby.

Configuration instructions: http://goo.gl/Q5a01N

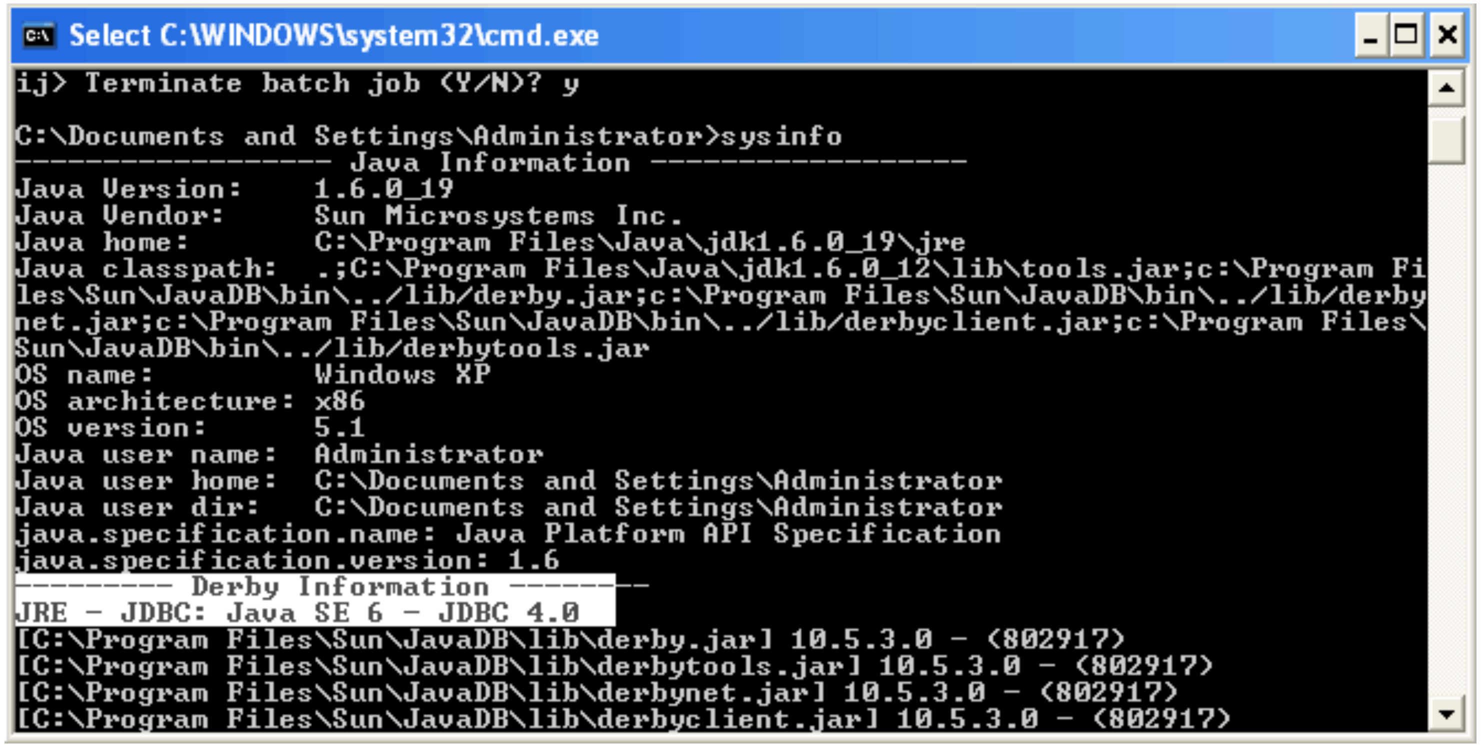
GlassFish server includes Derby DB (see lesson 26 from textbook).

In Windows, add *c:\Program Files\Sun\JavaDB\bin* to environment variable *PATH.*

Walkthrough 1

- Download the latest Derby DB (binary distribution) from http://db.apache.org/derby/derby_downloads.html

  Unzip it in any directory and set *DERBY_HOME* and *PATH* variables according to instructions at http://bit.ly/11pe06H
- From the command prompt run *sysinfo* command:



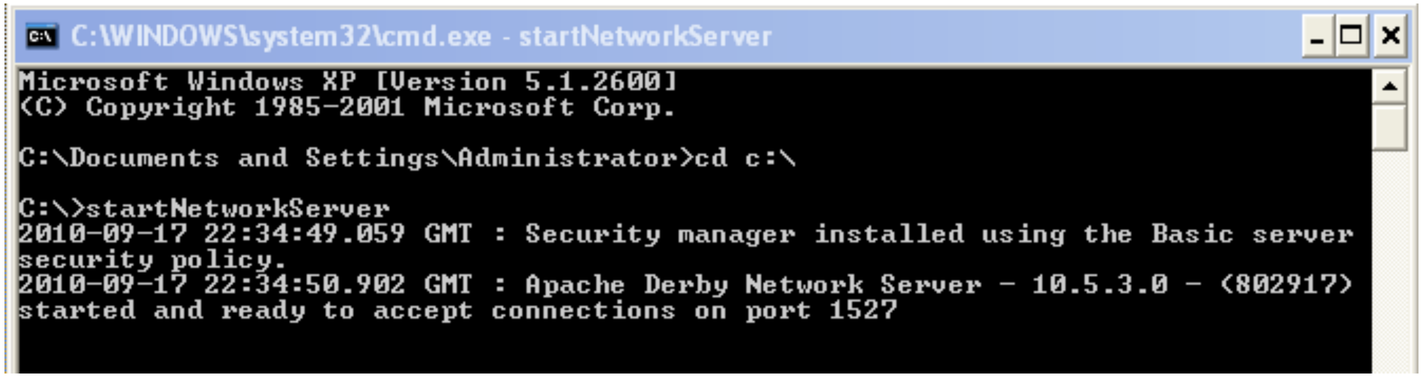
```
ij> Terminate batch job (Y/N)? y

C:\Documents and Settings\Administrator>sysinfo
------------------ Java Information ------------------
Java Version:    1.6.0_19
Java Vendor:     Sun Microsystems Inc.
Java home:       C:\Program Files\Java\jdk1.6.0_19\jre
Java classpath:  .;C:\Program Files\Java\jdk1.6.0_12\lib\tools.jar;c:\Program Fi
les\Sun\JavaDB\bin\../lib/derby.jar;c:\Program Files\Sun\JavaDB\bin\../lib/derby
net.jar;c:\Program Files\Sun\JavaDB\bin\../lib/derbyclient.jar;c:\Program Files\
Sun\JavaDB\bin\../lib/derbytools.jar
OS name:         Windows XP
OS architecture: x86
OS version:      5.1
Java user name:  Administrator
Java user home:  C:\Documents and Settings\Administrator
Java user dir:   C:\Documents and Settings\Administrator
java.specification.name: Java Platform API Specification
java.specification.version: 1.6
--------- Derby Information --------
JRE - JDBC: Java SE 6 - JDBC 4.0
[C:\Program Files\Sun\JavaDB\lib\derby.jar] 10.5.3.0 - (802917)
[C:\Program Files\Sun\JavaDB\lib\derbytools.jar] 10.5.3.0 - (802917)
[C:\Program Files\Sun\JavaDB\lib\derbynet.jar] 10.5.3.0 - (802917)
[C:\Program Files\Sun\JavaDB\lib\derbyclient.jar] 10.5.3.0 - (802917)
```

to be continued

Walkthrough 1 (cont.)

- Start Derby in a command window by typing *startNetworkServer*. You should see something like this:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd c:\

C:\>startNetworkServer
2010-09-17 22:34:49.059 GMT : Security manager installed using the Basic server
security policy.
2010-09-17 22:34:50.902 GMT : Apache Derby Network Server - 10.5.3.0 - (802917)
started and ready to accept connections on port 1527
```

- Open <u>another</u> command window, start Derby's utility **ij**, and connect to your database server creating the Lesson22 database:

  connect 'jdbc:derby:Lesson22;create=true';

  or

  connect 'jdbc:derby://localhost:1527/Lesson22;create=true';

to be continued

Walkthrough 1 (end)

- Using the **ij** utility create the table Employee:

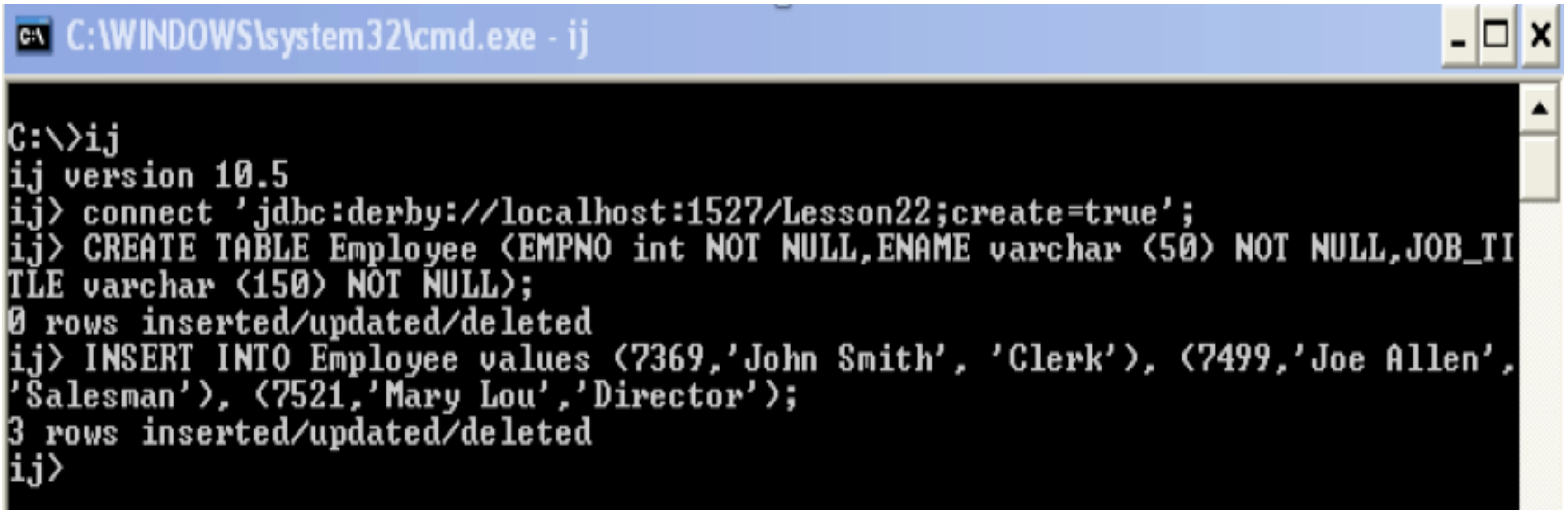
```
CREATE TABLE Employee ( EMPNO int NOT NULL, ENAME varchar (50)
NOT NULL, JOB_TITLE varchar (150) NOT NULL );
```

- Insert three rows into the table Employee:

```
INSERT INTO Employee values (7369,'John Smith', 'Clerk'),
(7499,'Joe Allen','Salesman'), (7521,'Mary Lou','Director');
```

- Test that the data were saved in the db:

```
Select * from Employee;
```



```
C:\>ij
ij version 10.5
ij> connect 'jdbc:derby://localhost:1527/Lesson22;create=true';
ij> CREATE TABLE Employee (EMPNO int NOT NULL,ENAME varchar (50) NOT NULL,JOB_TI
TLE varchar (150) NOT NULL);
0 rows inserted/updated/deleted
ij> INSERT INTO Employee values (7369,'John Smith', 'Clerk'), (7499,'Joe Allen',
'Salesman'), (7521,'Mary Lou','Director');
3 rows inserted/updated/deleted
ij>
```

`DriverManager` – the old way to do JDBC

1. Load the JDBC driver using the method `forName()` of the Java class `Class`.
For example, load Oracle Type 4 JDBC driver:

```
  Class.forName("oracle.jdbc.driver.OracleDriver");
```

In the case of **JavaDB**, which was installed and registered with your JDK, you can skip this step.

2. Obtain the database connection to the database **Lesson22**:

```
  DriverManager.getConnection(url, user, password);
```

In the case of Derby DB you don't have to supply the user and the password;

```
  DriverManager.getConnection("jdbc:derby:Lesson22");
```

3. Create an instance of the Java class Statement:

```
     Connection.createStatement();
```

As an alternative, you can create PreparedStatement or CallableStatement.

4. Execute the SQL Select query statement:

```
Statement.executeQuery("Select …");
```

5. Write a loop to process the database result set, if any:

```
  while (ResultSet.next()) {…}
```

6. Release the system resources by closing `ResultSet`, `Statement`, and `Connection` obj.

Retrieving Employee records in Java

```
class EmployeeList {
public static void main(String argv[]) {

  Connection conn=null;
  Statement stmt=null;
  ResultSet rs=null;

 try {

   // Load the JDBC driver - Class
   // This can be skipped for Derby, but derbyclient.jar
   //has to be in the CLASSPATH

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

   conn = DriverManager.getConnection(
                        "jdbc:derby://localhost:1527/Lesson22");


   // Create an SQL query
   String sqlQuery = "SELECT * from Employee";

   // Create an instance of the Statement object
   stmt = conn.createStatement();

   // Execute SQL and get obtain the ResultSet object
   rs = stmt.executeQuery(sqlQuery);
```

```
// Process the result set - print Employees
   while (rs.next()){
   int empNo = rs.getInt("EMPNO");
      String eName = rs.getString("ENAME");
      String job = rs.getString("JOB_TITLE");

System.out.println(""+ empNo + ", " + eName + ", " + job );
   }
  } catch( SQLException se ) {
     System.out.println ("SQLError: " + se.getMessage ()
         + " code: " + se.getErrorCode());
  } catch( Exception e ) {

     System.out.println(e.getMessage());
     e.printStackTrace();
  } finally{

      // clean up the system resources
      try{
rs.close();
stmt.close();
conn.close();
      } catch(Exception e){
         e.printStackTrace();
      }
  }
 }
}
```

`DataSource`: Preferred way to do JDBC

- Connecting to DBMS is a slow process - connecting/disconnect for every SQL request is bad. Use connection pools.

- The package **`javax.sql`** includes the interface `DataSource`, which is an alternative to `DriverManager`.
  JDBC drivers implement this interface, and a `DataSource` is typically preconfigured for a certain number of pooled connections.

- The `DataSource` interface is typically used on the server side bound to JNDI (see Lesson 31 in the textbook).

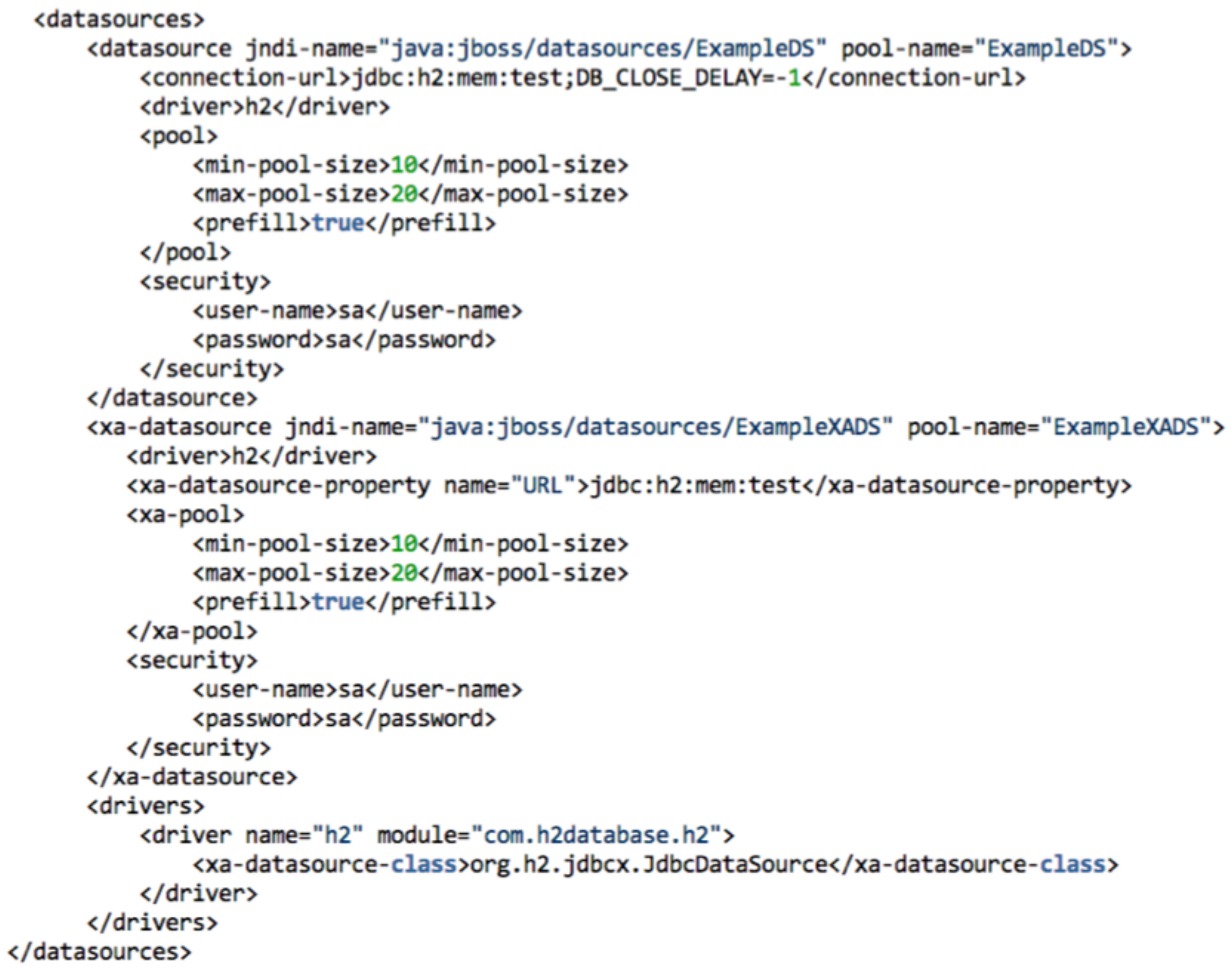
`DataSource` on the client and server

- Typically DataSource objects are preconfigured on the Java server and are bound to JNDI (see Lesson 31 in textbook)

- Server-side Java programs get this object either by JNDI lookup or by injection with @Resource annotation.

- A sample of a client's code connection to a Derby DB server using pooled connections is here: http://goo.gl/T74FYu

Sample DataSource config in JBoss Wildfly server



```
  <datasources>
      <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-name="ExampleDS">
          <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
          <driver>h2</driver>
          <pool>
              <min-pool-size>10</min-pool-size>
              <max-pool-size>20</max-pool-size>
              <prefill>true</prefill>
          </pool>
          <security>
              <user-name>sa</user-name>
              <password>sa</password>
          </security>
      </datasource>
      <xa-datasource jndi-name="java:jboss/datasources/ExampleXADS" pool-name="ExampleXADS">
         <driver>h2</driver>
         <xa-datasource-property name="URL">jdbc:h2:mem:test</xa-datasource-property>
         <xa-pool>
              <min-pool-size>10</min-pool-size>
              <max-pool-size>20</max-pool-size>
              <prefill>true</prefill>
         </xa-pool>
         <security>
              <user-name>sa</user-name>
              <password>sa</password>
         </security>
      </xa-datasource>
      <drivers>
          <driver name="h2" module="com.h2database.h2">
              <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
          </driver>
      </drivers>
</datasources>
```

Walkthrough 2

1. Download and import into Eclipse the sample code form Lesson 22. You'll see an error about the missing DB driver derbyclient.jar. Find it in the lib directory of your Derby installation and add it to the build path of your your project.

2. Review the code with the instructor

3. Run the application – it should display the list of employees:

```
7369, John Smith, Clerk
7499, Joe Allen, Salesman
7521, Mary Lou, Director
```

Note. If you see the error "Failed to start database 'Lesson22' … Another instance of Derby may have already booted the database", exit the ***ij*** utility with `exit;` command.

You can read more about specifics of running embedded Derby DB instance at http://db.apache.org/derby/papers/DerbyTut/embedded_intro.html

PreparedStatement

With PreparedStatement you can create SQL with parameters that are dynamically passed from the Java program to DBMS.

Suppose you need to execute the query "SELECT * from EMP WHERE empno=…" multiple times for each employee from the array empNumbers[].

```
// Using PreparedStatement
PreparedStatement stmt=conn.prepareStatement(
                       ” SELECT * from Employee WHERE empno=?”);

for (int i=0;i<empNumbers.length; i++){

  // pass the array’s value that substitutes the ?
  stmt.setInt(1,employees[i];)
  stmt.executeQuery(sqlQuery);
}
```

← SQL gets compiled once

```
// Using Statement
stmt = conn.createStatement();
for (int i=0;i<empNumbers.length; i++){
  sqlQuery=”SELECT * from Employee WHERE empno=” + employees[i];
  stmt.executeQuery(sqlQuery);
}
```

← SQL gets compiled for each employee

Transactional Updates

Transaction is a single unit of work.

Transaction can consist of several actions, and **all** of them must either finish successfully or be undone.

```
try{
  conn.setAutoCommit(false);

  Statement stmt = con.createStatement();

  stmt.addBatch("insert into Orders " +
              "values(123, ‘Buy’,’IBM’,200)”);
  stmt.addBatch("insert into OrderDetail " +
              "values('JSmith', ‘Broker131’, ‘05/20/02’)");
  stmt.executeBatch();

  conn.commit();    // Transaction succeeded

}catch(SQLException e){
  conn.rollback();  // Transaction failed
  e.printStackTrace();
}
```

Homework

1. Study the materials from the lessons 22.

2. Do the assignment from the Try It section of Lesson 22.

3. Study SQL: http://www.sqlcourse.com/

4. Get familiar with working with DerbyDB from Eclipse: http://www.vogella.de/articles/EclipseDataToolsPlatform/

Design Patterns: DAO, DTO, VO

- Data Transfer Object (a.k.a. Value Object) – a POJO to store data (e.g. `Customer.java`)

- Data Access Object perform operations on working with the data source, e.g. `getCustomers()`, `getCustomer(int custID)` etc. The DTO objects are given to such methods as arguments or are returned as a query results.

Additional Reading

10 common mistakes: http://goo.gl/C4aj3H

10 easy steps to understand SQL: http://goo.gl/CAeAPp

Connecting to DBMS using `DataSource`: http://goo.gl/8QY3w

Derby tutorials http://db.apache.org/derby and tips: http://bit.ly/1dYs4xn.

An advanced book: "SQL for Smarties" by Joe Celko: http://goo.gl/WQQLZK

Database Interaction with DAO and DTO: http://goo.gl/zXsoZB

Challenge Yourself

1. Download and install DBMS Oracle 11g Express edition from http://bit.ly/Qmkzpt . During the installation enter (and memorize) the passwords for SYS and SYSTEM accounts.

2. Download Oracle SQL Developer from http://bit.ly/1bAq4Y7. Unzip it into any folder. SQL Developer doesn't support Java 8 yet, so you'll need to install Java 7.

3. Run the sqldeveloper.exe located in sqldeveloper subfolder. Specify where your JDK is installed, e.g C:\Program Files\Java\jdk1.7.0_55.

4. For instructions on using SQL Developer follow this video: https://www.youtube.com/watch?v=ZINT9tCl20g.

5. Use Eclipse plugin for Oracle: http://bit.ly/rYIDid

6. You can get ojdbc7.jar, the thin Oracle JDBC driver at http://bit.ly/1d4rn1Z