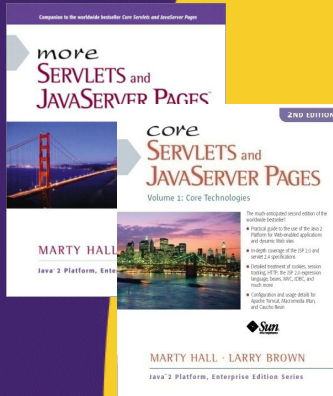




# Database Access with JDBC

Originals of Slides and Source Code for Examples:  
<http://courses.coreservlets.com/Course-Materials/msajsp.html>

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live Java EE training, please see training courses  
at <http://courses.coreservlets.com/>.**

**Servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax (with jQuery, Dojo, Prototype, Ext-JS, Google Closure, etc.), GWT 2.0 (with GXT), Java 5, Java 6, SOAP-based and RESTful Web Services, Spring, Hibernate/JPA, and customized combinations of topics.**



**Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact [hall@coreservlets.com](mailto:hall@coreservlets.com) for details.**

# Overview

- Overview of JDBC technology
- JDBC design strategies
- Using Apache Derby (Java DB)
- Seven basic steps in using JDBC
- Using JDBC from desktop Java apps
- Using JDBC from Web apps
- Prepared statements (parameterized commands)
- Meta data
- Transaction control

5

© 2010 Marty Hall



# Overview

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# JDBC Introduction

- **JDBC provides a standard library for accessing relational databases**
  - API standardizes
    - Way to establish connection to database
    - Approach to initiating queries
    - Method to create stored (parameterized) queries
    - The data structure of query result (table)
      - Determining the number of columns
      - Looking up metadata, etc.
  - API does not standardize SQL syntax
    - JDBC is not embedded SQL
  - JDBC classes are in the java.sql package
- **Note: JDBC is not officially an acronym;**
  - Unofficially, “Java DataBase Connectivity” is commonly used

7

# On-line Resources

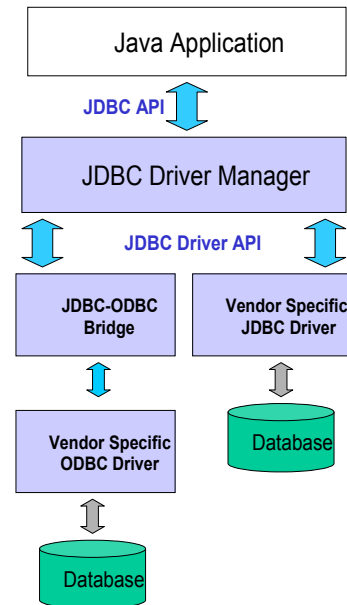
- **Sun’s JDBC Site**
  - <http://java.sun.com/javase/6/docs/technotes/guides/jdbc/>
- **JDBC Tutorial**
  - <http://java.sun.com/docs/books/tutorial/jdbc/>
- **API for java.sql**
  - <http://java.sun.com/javase/6/docs/api/java/sql/package-summary.html>
- **List of Available JDBC Drivers**
  - <http://developers.sun.com/product/jdbc/drivers>
    - Or, just look in your database vendor’s documentation

8

# JDBC Drivers

- **JDBC consists of two parts:**

- JDBC API, a purely Java-based API
- JDBC Driver Manager, which communicates with vendor-specific drivers that perform the real communication with the database.
  - Point: translation to vendor format is performed on the client
    - No changes needed to server
    - Driver (translator) needed on client



# JDBC Data Types

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	
BINARY	byte[]
VARBINARY	
LONGVARBINARY	
CHAR	String
VARCHAR	
LONGVARCHAR	

JDBC Type	Java Type
NUMERIC	BigDecimal
DECIMAL	
DATE	java.sql.Date
TIME	java.sql.Timestamp
TIMESTAMP	
CLOB	Clob
BLOB	Blob
ARRAY	Array
DISTINCT	mapping of underlying type
STRUCT	Struct
REF	Ref
JAVA_OBJECT	underlying Java class



# Steps for Using JDBC

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## JDBC Design Strategies

- **In general, plan for changes to data access**
  - Limit the data access to single area of code
    - Don't distribute JDBC calls throughout the code
    - Plan ahead for changing from JDBC to Hibernate or another tool
  - Don't return JDBC-specific (or Hibernate-specific) objects from the data-access layer
    - Return ordinary Java objects instead
- **In JDBC, plan for changes**
  - Limit the definition of driver, URL, etc. to single location
- **Let database experts do their stuff**
  - If database is complex, let database expert design the database and design the queries

# Seven Basic Steps in Using JDBC

- 1. Load the driver**
  - Not required in Java 6, so Java 6 needs only 6 steps.
- 2. Define the Connection URL**
- 3. Establish the Connection**
- 4. Create a Statement object**
- 5. Execute a query**
- 6. Process the results**
- 7. Close the connection**

13

## JDBC Step 1: Load the Driver

- **Not required in Java 6**
  - In Java SE 6.0 and later (JDBC 4.0 and later), the driver is loaded automatically.
- **Java 5 and earlier**
  - Load the driver *class* only. The class has a static initialization block that makes an instance and registers it with the DriverManager.

```
try {  
    Class.forName("com.mysql.jdbc.Driver");  
    Class.forName("oracle.jdbc.driver.OracleDriver");  
} catch (ClassNotFoundException cnfe) {  
    System.out.println("Error loading driver: " cnfe);  
}
```

14

## JDBC Step 2: Define the Connection URL

- **Remote databases**

- Format is “`jdbc:vendorName:...`”
  - Address contains hostname, port, and database name
  - Exact details given by supplier of JDBC driver

- **Embedded Derby database**

- The “Java DB” (i.e., Apache Derby) is bundled with Java 6 and can be used for a database embedded in the same Java VM that runs the app server.
- Format is “`jdbc:derby:databaseName`”

- **Examples**

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String mySqlUrl = "jdbc:mysql://" + host + ":" + port +
                 "/" + dbName;
String embeddedDerbyUrl = "jdbc:derby" + dbName;
```

15

## JDBC Step 3: Establish the Connection

- **Get the main connection**

```
Properties userInfo = new Properties();
userInfo.put("user", "jay_debese");
userInfo.put("password", "secret");
Connection connection =
    DriverManager.getConnection(mySqlUrl, userInfo);
```

- **Optionally, look up info about the database**

```
DatabaseMetaData dbMetaData =
    connection.getMetaData();
String productName =
    dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
String productVersion =
    dbMetaData.getDatabaseProductVersion();
System.out.println("Version: " + productVersion);
```

16

## JDBC Step 4: Make a Statement

- **Idea**

- A Statement is used to send queries or commands

- **Statement types**

- Statement, PreparedStatement, CallableStatement
  - Details on other types given later

- **Example**

```
Statement statement =  
    connection.createStatement();
```

17

## JDBC Step 5: Execute a Query

- **Idea**

- statement.executeQuery("SELECT ... FROM ...");
  - This version returns a ResultSet
- statement.executeUpdate("UPDATE ...");
- statement.executeUpdate("INSERT ...");
- statement.executeUpdate("DELETE...");
- statement.execute("CREATE TABLE...");
- statement.execute("DROP TABLE ...");

- **Example**

```
String query =  
    "SELECT col1, col2, col3 FROM sometable";  
ResultSet resultSet =  
    statement.executeQuery(query);
```

18

## JDBC Step 6: Process the Result

- **Important ResultSet methods**

- `resultSet.next()`
  - Goes to the next row. Returns false if no next row.
- `resultSet.getString("columnName")`
  - Returns value of column with designated name in current row, as a String. Also `getInt`, `getDouble`, `getBlob`, etc.
- `resultSet.getString(columnIndex)`
  - Returns value of designated column. First index is 1 (ala SQL), not 0 (ala Java).
- `resultSet.beforeFirst()`
  - Moves cursor before first row, as it was initially. Also `first`
- `resultSet.absolute(rowNum)`
  - Moves cursor to given row (starting with 1). Also `last` and `afterLast`.

19

## JDBC Step 6: Process the Result

- **Assumption**

- Query was “SELECT first, last, address FROM...”

- **Using column names**

```
while(resultSet.next()) {
    System.out.printf(
        "First name: %s, last name: %s, address: %s%n",
        resultSet.getString("first"),
        resultSet.getString("last"),
        resultSet.getString("address"));
}
```

- **Using column indices**

```
while(resultSet.next()) {
    System.out.printf(
        "First name: %s, last name: %s, address: %s%n",
        resultSet.getString(1),
        resultSet.getString(2),
        resultSet.getString(3)); } }
```

20

# JDBC Step 7: Close the Connection

- **Idea**

- When *totally* done, close the database connection. However, opening a new connection is typically much more expensive than sending queries on existing connections, so postpone this step as long as possible.
- Many JDBC drivers do automatic connection pooling
- There are also many explicit connection pool utilities

- **Example**

```
connection.close();
```

21

© 2010 Marty Hall



## Using Apache Derby

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Apache Derby: Overview

- **Overview**

- Written in Java. Good for small/medium applications (less than gigabyte database size, few queries/second)
- Bundled with Java 6, but latest version can be downloaded from Apache for Java 1.4 and later.

- **Embedded mode**

- Database runs in same VM as Java app. Does not accept network connections.
- Perfect for Web apps
  - Incredibly easy to set up. Just drop in one JAR file
  - Good if database is accessed only from the Web apps

- **Standalone mode**

- Runs in separate VM. Accepts network connections.

23

# Downloading and Installing Derby

- **Downloading and documentation**

- <http://db.apache.org/derby/>

- **Use in embedded mode**

- Code
  - For Web apps, drop derby.jar in WEB-INF/lib
  - For other apps, put derby.jar in CLASSPATH
- URL
  - jdbc:derby:*databaseName*
- Driver class name
  - org.apache.derby.jdbc.EmbeddedDriver
  - Not needed in Java 6!
- Username/password
  - Any are legal since DB runs inside the application.

- **Use in standalone mode**

- See <http://db.apache.org/derby/papers/DerbyTut/>
- Consider MySQL, Oracle, etc. as alternatives

24

# Setup Summary

- **Setup summary**

- Downloaded latest Derby ZIP from [db.apache.org/derby/](http://db.apache.org/derby/)
  - db-derby-10.5.3.0-bin.zip, but any recent version is fine
  - You can also use version bundled with JDK 1.6.x
- Unzipped
- Put *install\_dir/lib/derby.jar* into CLASSPATH
  - Many options in Eclipse, but if you make a Dynamic Web Project, drop derby.jar in WebContent/WEB-INF/lib
- That's it (for embedded usage!)
  - Compare this 90 second process to installing Oracle

- **Creating database**

- Can be created directly from Java using SQL commands via JDBC. See later slides. Need to run creator code at least once before accessing database.

25

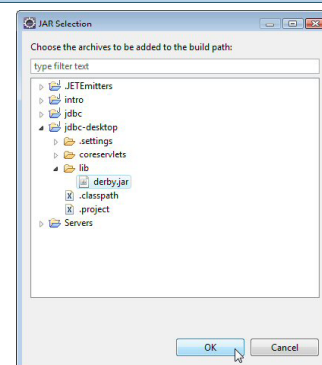
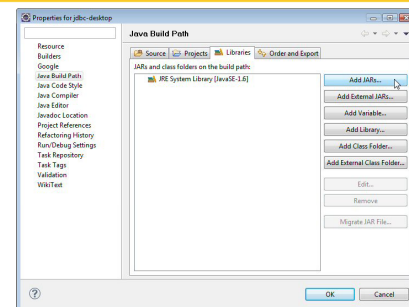
# Setup Details: Desktop Apps

- **Putting derby.jar in CLASSPATH**

- R-click on project in Eclipse, create new folder called “lib”. Put derby.jar in lib.
- R-click on project. Properties → Java Build Path → Libraries → Add JARs
- Navigate to project/lib/ and select derby.jar. Press OK.

- **Creating database**

- Manually run database creation code. See “Prepared Statements” section for code, but this needs to be done once only. In most real apps, you have to *query* database, but someone else *creates* the database.



26

# Setup Details: Web Apps

- **Putting derby.jar in CLASSPATH**
  - Copy derby.jar to WEB-INF/lib
- **Creating database**
  - Run database creation code. See “Prepared Statements” section for code . This needs to be done once only, so best way is to do it automatically in ServletContextListener. In most real apps, you have to *query* database, but someone else *creates* the database.
    - Reminder: full code can be downloaded from <http://courses.coreservlets.com/Course-Materials/msajsp.html>, so you can get db creation code there.

27

© 2010 Marty Hall



## Using JDBC from Desktop Java

Customized Java EE Training: <http://courses.coreservlets.com/>  
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Approach

- **Same basic steps**
  - Load the driver
  - Define the Connection URL
  - Establish the Connection
  - Create a Statement object
  - Execute a query
  - Process the results
  - Close the connection
- **Differences from JDBC in Web apps**
  - Results are often put into Swing or AWT interfaces
  - If you use value in calculation, use `getInt`, `getDouble`, etc.
  - If value is only used for display in GUI, you can use `getString` even if value is another type

29

# Sample Database

- **Table name**
  - `employees`
- **Column names**
  - `id` (int). The employee ID.
  - `firstname` (varchar/String). Employee's given name.
  - `lastname` (varchar/String). Employee's family name.
  - `position` (varchar/String). Corporate position (eg, "ceo").
  - `salary` (int). Yearly base salary.
- **Database name**
  - `myDatabase`
- **Note**
  - See "Prepared Statements" section for code that created DB

30

## Example: Printing Employee Info

```
package coreservlets;

import java.sql.*;
import java.util.*;

public class ShowEmployees {
    public static void main(String[] args) {
        String url = "jdbc:derby:myDatabase";
        Properties userInfo = new Properties();
        userInfo.put("user", "someuser");
        userInfo.put("password", "somepassword");
        String driver =
            "org.apache.derby.jdbc.EmbeddedDriver";
        showSalaries(url, userInfo, driver);
    }
}
```

The URL and the driver are the only parts that are specific to Derby. So, if you switch to MySQL, Oracle, etc., you have to change those two lines (or just one line in Java 6 with JDBC 4 driver, since the driver no longer needs to be declared in that situation). The rest of the code is database independent.

31

## Example: Printing Employee Info (Connecting to Database)

```
public static void showSalaries(String url,
                                Properties userInfo,
                                String driverClass) {
    try {
        // Load the driver. NOT NEEDED in Java 6!
        // Class.forName(driverClass);

        // Establish network connection to database.
        Connection connection =
            DriverManager.getConnection(url, userInfo);
        System.out.println("Employees\n=====");
        // Create a statement for executing queries.
        Statement statement = connection.createStatement();
        String query =
            "SELECT * FROM employees ORDER BY salary";
        // Send query to database and store results.
        ResultSet resultSet = statement.executeQuery(query);
    }
}
```

32

## Example: Printing Employee Info (Processing Results)

```
while(resultSet.next()) {
    int id = resultSet.getInt("id");
    String firstName = resultSet.getString("firstname");
    String lastName = resultSet.getString("lastname");
    String position = resultSet.getString("position");
    int salary = resultSet.getInt("salary");
    System.out.printf
        ("%s %s (%s, id=%s) earns $%,d per year.%n",
         firstName, lastName, position, id, salary);
}
connection.close();
} catch(Exception e) {
    System.err.println("Error with connection: " + e);
}
```

33

## Example: Printing Employee Info (Output)

### Employees

=====

```
Gary Grunt (Gofer, id=12) earns $7,777 per year.
Gabby Grunt (Gofer, id=13) earns $8,888 per year.
Cathy Coder (Peon, id=11) earns $18,944 per year.
Cody Coder (Peon, id=10) earns $19,842 per year.
Danielle Developer (Peon, id=9) earns $21,333 per year.
David Developer (Peon, id=8) earns $21,555 per year.
Joe Hacker (Peon, id=6) earns $23,456 per year.
Jane Hacker (Peon, id=7) earns $32,654 per year.
Keith Block (VP, id=4) earns $1,234,567 per year.
Thomas Kurian (VP, id=5) earns $2,431,765 per year.
Charles Phillips (President, id=2) earns $23,456,789 per year.
Safra Catz (President, id=3) earns $32,654,987 per year.
Larry Ellison (CEO, id=1) earns $1,234,567,890 per year.
```

34



# Using JDBC from Web Apps

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Approach

- **Same basic steps**
  - Load the driver
  - Define the Connection URL
  - Establish the Connection
  - Create a Statement object
  - Execute a query
  - Process the results
  - Close the connection
- **Differences from JDBC in desktop apps**
  - Results are often put into HTML
  - If value is inserted directly into HTML, you can use `getString` even if value is another type
  - To support concurrent access, you usually use a driver that supports connection pooling

# Sample Database

- **Table name**
  - employees
- **Column names**
  - **id** (int). The employee ID.
  - **firstname** (varchar/String). Employee's given name.
  - **lastname** (varchar/String). Employee's family name.
  - **position** (varchar/String). Corporate position (eg, "ceo").
  - **salary** (int). Yearly base salary.
- **Database name**
  - myDatabase
- **Note**
  - See "Prepared Statements" section for code that created DB

37

# A Servlet to Show Employee Info

- **Overview**
  - Same sample database as before
    - DB type: Derby in embedded mode
    - DB name: employees
    - Columns: id, firstname, lastname, position, salary
- **Goal**
  - Build HTML table that shows all employees
- **Approach**
  - Build HTML table directly in servlet
    - MVC combined with JSTL or custom tags might work better, but this lecture does not assume familiarity with those topics
    - Basic JDBC code is the same either way

38

# Employee Info Servlet

```
public class EmployeeServlet1 extends HttpServlet {
    //private final String driver =
    //    "org.apache.derby.jdbc.EmbeddedDriver";
    protected final String url = "jdbc:derby:myDatabase";
    protected final String tableName = "employees";
    protected final String username = "someuser";
    protected final String password = "somepassword";
```

The URL and the driver are the only parts that are specific to Derby. So, if you switch to MySQL, Oracle, etc., you have to change those two lines (or just one line in Java 6 with JDBC 4 driver, since the driver no longer needs to be declared in that situation). The rest of the code is database independent.

39

# Employee Info Servlet (Continued)

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
        "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
        \"Transitional//EN\" \"\n\"";
    String title = "Company Employees";
    out.print(docType +
        "<HTML>\n" +
        "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
        "<LINK REL='STYLESHEET' HREF='./css/styles.css'\n" +
        "    TYPE='text/css'>" +
        "<BODY><CENTER>\n" +
        "<TABLE CLASS='TITLE' BORDER='5'>" +
        "    <TR><TH>" + title + "</TABLE><P>");
    showTable(out);
    out.println("</CENTER></BODY></HTML>");
}
```

40

## Employee Info Servlet (Continued)

```
protected void showTable(PrintWriter out) {
    try {
        Connection connection = getConnection();
        Statement statement = connection.createStatement();
        String query = "SELECT * FROM " + tableName;
        ResultSet resultSet = statement.executeQuery(query);
        printTableTop(connection, resultSet, out);
        printTableBody(resultSet, out);
        connection.close();
    } catch(Exception e) {
        System.err.println("Error: " + e);
    }
}
```

41

## Employee Info Servlet (Continued)

```
protected Connection getConnection()
    throws Exception {
    // Load database driver if it's not already loaded.
    // Not needed in JDBC 4 (Java SE 6). Uncomment
    // for earlier versions.
    // Class.forName(driver);

    // Establish network connection to database.
    Properties userInfo = new Properties();
    userInfo.put("user", username);
    userInfo.put("password", password);
    Connection connection =
        DriverManager.getConnection(url, userInfo);
    return connection;
}
```

42

## Employee Info Servlet (Continued)

```
protected void printTableTop(Connection connection,
                             ResultSet resultSet,
                             PrintWriter out)
    throws SQLException {
    out.println("<TABLE BORDER='1'>");
    // Print headings from explicit heading names
    String[] headingNames =
        { "ID", "First Name", "Last Name",
          "Position", "Salary" };
    out.print("<TR>");
    for (String headingName : headingNames) {
        out.printf("<TH>%s", headingName);
    }
    out.println();
}
```

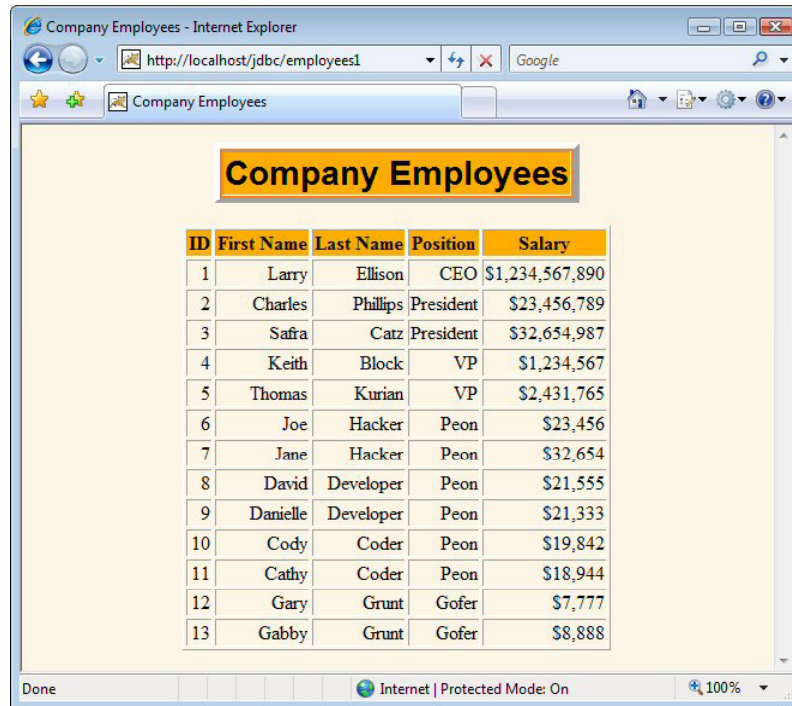
43

## Employee Info Servlet (Continued)

```
protected void printTableBody(ResultSet resultSet,
                              PrintWriter out)
    throws SQLException {
    // Step through each row in the result set and print cells
    while(resultSet.next()) {
        out.println("<TR ALIGN='RIGHT'>");
        out.printf(" <TD>%d", resultSet.getInt("id"));
        out.printf(" <TD>%s", resultSet.getString("firstname"));
        out.printf(" <TD>%s", resultSet.getString("lastname"));
        out.printf(" <TD>%s", resultSet.getString("position"));
        out.printf(" <TD>${%,d%n", resultSet.getInt("salary"));
    }
    out.println("</TABLE>");
}
}
```

44

# Employee Info Servlet (Results)



ID	First Name	Last Name	Position	Salary
1	Larry	Ellison	CEO	\$1,234,567,890
2	Charles	Phillips	President	\$23,456,789
3	Saffra	Catz	President	\$32,654,987
4	Keith	Block	VP	\$1,234,567
5	Thomas	Kurian	VP	\$2,431,765
6	Joe	Hacker	Peon	\$23,456
7	Jane	Hacker	Peon	\$32,654
8	David	Developer	Peon	\$21,555
9	Danielle	Developer	Peon	\$21,333
10	Cody	Coder	Peon	\$19,842
11	Cathy	Coder	Peon	\$18,944
12	Gary	Grunt	Gofer	\$7,777
13	Gabby	Grunt	Gofer	\$8,888

45

© 2010 Marty Hall



## Using MetaData

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Using MetaData

- **Idea**

- For most queries, you know column names and database version ahead of time. But you can discover this dynamically as well.

- **System-wide data**

- `connection.getMetaData().getDatabaseProductName()`
- `connection.getMetaData().getDatabaseProductVersion()`

- **Table-specific data**

- `resultSet.getMetaData().getColumnCount()`
  - When using the result, remember that the index starts at 1, not 0
- `resultSet.getMetaData().getColumnName()`

47

# Using MetaData: Example

```
public class EmployeeServlet2 extends EmployeeServlet1 {
    protected void printTableTop(Connection connection,
                                ResultSet resultSet,
                                PrintWriter out)
        throws SQLException {
        // Look up info about the database as a whole.
        DatabaseMetaData dbMetaData = connection.getMetaData();
        String productName =
            dbMetaData.getDatabaseProductName();
        String productVersion =
            dbMetaData.getDatabaseProductVersion();
        out.println("<UL>\n" +
            "    <LI><B>Database:</B>\n" + productName +
            "    <LI><B>Version:</B>\n" + productVersion +
            "</UL>");
    }
}
```

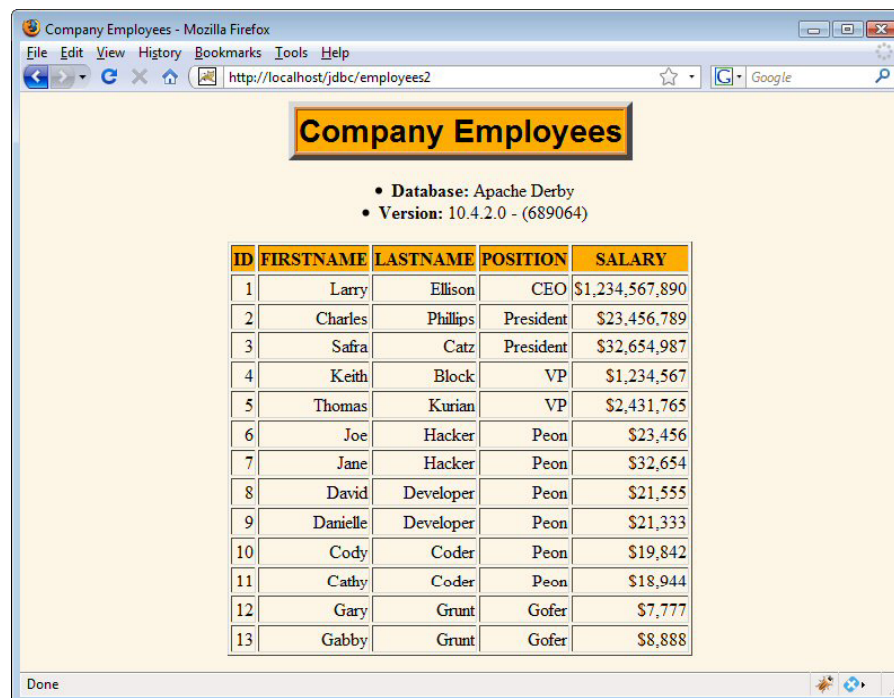
48

# Using MetaData: Example (Continued)

```
out.println("<TABLE BORDER='1'>");
// Discover and print headings
ResultSetMetaData resultSetMetaData =
    resultSet.getMetaData();
int columnCount = resultSetMetaData.getColumnCount();
out.println("<TR>");
// Column index starts at 1 (a la SQL), not 0 (a la Java).
for(int i=1; i <= columnCount; i++) {
    out.printf("<TH>%s", resultSetMetaData.getColumnName(i));
}
out.println();
}
}
```

49

# Using MetaData: Results



The screenshot shows a web browser window titled "Company Employees - Mozilla Firefox". The address bar displays "http://localhost/jdbc/employees2". The page content includes a title "Company Employees" and two bullet points: "Database: Apache Derby" and "Version: 10.4.2.0 - (689064)". Below this is a table with 5 columns: ID, FIRSTNAME, LASTNAME, POSITION, and SALARY. The table contains 13 rows of employee data.

ID	FIRSTNAME	LASTNAME	POSITION	SALARY
1	Larry	Ellison	CEO	\$1,234,567,890
2	Charles	Phillips	President	\$23,456,789
3	Safra	Catz	President	\$32,654,987
4	Keith	Block	VP	\$1,234,567
5	Thomas	Kurian	VP	\$2,431,765
6	Joe	Hacker	Peon	\$23,456
7	Jane	Hacker	Peon	\$32,654
8	David	Developer	Peon	\$21,555
9	Danielle	Developer	Peon	\$21,333
10	Cody	Coder	Peon	\$19,842
11	Cathy	Coder	Peon	\$18,944
12	Gary	Grunt	Gofer	\$7,777
13	Gabby	Grunt	Gofer	\$8,888

50



# Using Prepared Statements (Parameterized Commands)

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Overview

- **Situation**
  - You are repeatedly executing query or update where format stays consistent, but values change
  - You can make a parameterized query or update, then pass in values for the placeholders
- **Advantages**
  - More convenient than string concatenation
  - Significantly faster with most drivers and databases
  - If the values contain user data, much less susceptible to SQL injection attacks

## Steps

- **Make a placeholder**
  - Assume someTable has two columns (int and String)  
String template =  
"INSERT INTO someTable VALUES(?, ?)";  
PreparedStatement inserter =  
connection.prepareStatement(template);
- **Substitute in values**
  - Use setInt(index), setString(index), etc. Remember indices start with 1, not 0.  
for(...) {  
    inserter.setInt(1, someInt);  
    inserter.setString(2, someString)  
}
- **Execute command**
  - inserter.executeUpdate();

53

## Creating the Database

- **Roles**
  - Most JDBC developers do not need to create the database
  - That is the job of the database system admin
- **Apache Derby**
  - You can run Derby-specific scripts and create the database interactively
  - Or, you can use Java and JDBC
    - Using PreparedStatement simplifies the process
- **Simple “myDatabase” example**
  - Created with Java and JDBC
  - Triggered from a ServletContextListener so database creation code is executed at least once

54

## Creating Sample Database

```
public class EmbeddedDbCreator {
    // Driver class not needed in JDBC 4.0 (Java SE 6)
    // private String driver =
    //     "org.apache.derby.jdbc.EmbeddedDriver";
    private String protocol = "jdbc:derby:";
    private String username = "someuser";
    private String password = "somepassword";
    private String dbName = "myDatabase";
    private String tableName = "employees";
    private Properties userInfo;

    public EmbeddedDbCreator() {
        userInfo = new Properties();
        userInfo.put("user", username);
        userInfo.put("password", password);
    }
}
```

55

## Creating Sample Database (Continued)

```
public void createDatabase() {
    Employee[] employees = {
        new Employee(1, "Larry", "Ellison", "CEO",
                    1234567890),
        new Employee(2, "Charles", "Phillips", "President",
                    23456789),
        new Employee(3, "Safra", "Catz", "President",
                    32654987),
        ...
    };
};
```

56

## Creating Sample Database (Continued)

```
try {
    String dbUrl = protocol + dbName + ";create=true";
    Connection connection =
        DriverManager.getConnection(dbUrl, userInfo);
    Statement statement = connection.createStatement();
    String format = "VARCHAR(20)";
    String tableDescription =
        String.format
            ("CREATE TABLE %s" +
             "(id INT, firstname %s, lastname %s, " +
              "position %s, salary INT)",
             tableName, format, format, format);
    statement.execute(tableDescription);
}
```

57

## Creating Sample Database (Continued)

```
String template =
    String.format("INSERT INTO %s VALUES(?, ?, ?, ?, ?)",
                 tableName);
PreparedStatement inserter =
    connection.prepareStatement(template);
for(Employee e: employees) {
    inserter.setInt(1, e.getEmployeeID());
    inserter.setString(2, e.getFirstName());
    inserter.setString(3, e.getLastName());
    inserter.setString(4, e.getPosition());
    inserter.setInt(5, e.getSalary());
    inserter.executeUpdate();
    System.out.printf("Inserted %s %s.%n",
                      e.getFirstName(),
                      e.getLastName());
}
inserter.close();
connection.close();
```

58

# Triggering Database Creation

- **Database needs to be created once**

- Not for every request
- Not even for every time server restarts

- **Goal**

- Create table when Web app is loaded
  - try/catch block in creation code halts process if table already exists

- **Possible approaches**

- Servlet with creation code in init, and web.xml with `<load-on-startup>1</load-on-startup>`
- ServletContextListener
  - Same effect as above, but more straightforward
- For desktop apps, just manually run the code *once*.

59

# Triggering Database Creation: Listener

```
package coreservlets;

import javax.servlet.*;

public class DatabaseInitializer
    implements ServletContextListener {
    public void contextInitialized(ServletContextEvent event) {
        new EmbeddedDbCreator().createDatabase();
    }

    public void contextDestroyed(ServletContextEvent event) {}
}
```

60

# Triggering Database Creation: web.xml

```
...  
<listener>  
  <listener-class>  
    coreservlets.DatabaseInitializer  
  </listener-class>  
</listener>
```

61

© 2010 Marty Hall



## Advanced Features

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Transactions

- **Idea**

- By default, after each SQL statement is executed the changes are automatically committed to the database
- Turn auto-commit off to group two or more statements together into a transaction

```
connection.setAutoCommit(false);
```

- Call **commit** to permanently record the changes to the database after executing a group of statements
- Call **rollback** if an error occurs

63

# Transactions: Example

```
Connection connection =
    DriverManager.getConnection(url, userProperties);
connection.setAutoCommit(false);
try {
    statement.executeUpdate(...);
    statement.executeUpdate(...);
    ...
    connection.commit();
} catch (Exception e) {
    try {
        connection.rollback();
    } catch (SQLException sqle) {
        // report problem
    }
} finally {
    try {
        connection.close();
    } catch (SQLException sqle) { }
}
```

64

## Useful Connection Methods (for Transactions)

- **getAutoCommit/setAutoCommit**
  - By default, a connection is set to auto-commit
  - Retrieves or sets the auto-commit mode
- **commit**
  - Force all changes since the last call to commit to become permanent
  - Any database locks currently held by this `Connection` object are released
- **rollback**
  - Drops all changes since the previous call to commit
  - Releases any database locks held by this `Connection` object

65

## Using JNDI and DataSource

- **Idea**
  - Use abstract name to get connection from a data source
- **Advantages**
  - Lets you change data source without changing code
  - Available in multiple Web apps for network databases (not embedded Derby!)
- **Disadvantage**
  - Requires server-specific registration of data source
- **Code for steps 1-3 replaced by:**

```
Context context = new InitialContext();
DataSource dataSource = (DataSource)context.lookup
    ("java:comp/env/jdbc/dbName");
Connection connection = dataSource.getConnection();
```

66

## DataSource: Tomcat Configuration (META-INF/context.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource
    name="jdbc/myDatabase"
    driverClassName="org.apache.derby.jdbc.ClientDriver"
    url="jdbc:derby:myDatabase"
    type="javax.sql.DataSource"
    username="someuser"
    password="somepassword"
    auth="Container"
    maxActive="8" />
</Context>
```

67

## More JDBC Options

- **Stored procedures**
- **Changing buffer size**
- **Connection pooling**
- **Hibernate/JPA and other ORM tools**
  - See tutorials at [coreservlets.com](http://coreservlets.com)
- **JSP Standard Tag Library (JSTL)**
  - Custom tags to hide JDBC details.
    - Some developers use JSTL to loop down the ResultSet. Disadvantage: ties presentation layer to JDBC.
    - Some developers use JSTL to actually do queries. Simple. Disadvantage: very inflexible and violates MVC principle.

68



## Wrap-Up

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## JDBC and Derby References

- **Books**
  - *JDBC API Tutorial and Reference* by Maydene Fisher, Jon Ellis, and Jonathan Bruce (Prentice Hall)
  - *Expert Oracle JDBC Programming* by R.M. Menon (APress)
  - *JDBC Recipes: A Problem-Solution Approach* by Mahmoud Parsian (APress)
- **Sun JDBC tutorial**
  - <http://java.sun.com/docs/books/tutorial/jdbc/>
- **Derby references**
  - Derby beginner's tutorial
    - <http://db.apache.org/derby/papers/DerbyTut/>
  - Derby 10.4 manuals
    - [http://db.apache.org/derby/manuals/index.html#docs\\_10.4](http://db.apache.org/derby/manuals/index.html#docs_10.4)

# Summary

- 1. Load the driver**
  - Not required in Java 6; use `Class.forName` otherwise
- 2. Define the Connection URL**
  - `jdbc:vendor:blah` (vendor gives exact format)
- 3. Establish the Connection**
  - `DriverManager.getConnection`
- 4. Create a Statement object**
  - `connection.createStatement`
- 5. Execute a query**
  - `statement.executeQuery`
- 6. Process the results**
  - Loop using `resultSet.next()`
  - Call `getString`, `getInt`, etc.
- 7. Close the connection**

71

© 2010 Marty Hall



## Questions?

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.