

Java Programming, Comprehensive

Lecture 7

Bineet Sharma

Agenda: Working with Database

- ▶ SQL
- ▶ Derby RDBMS
- ▶ JDBC

Working with Database: Derby Database

Objectives

Applied

- Configure your system so you can work with a Derby database.
- Use the interactive JDBC tool to connect to a database and execute SQL statements.
- Given a SQL script that creates a Derby database, use the interactive JDBC tool to execute that script.
- Given the specifications for a table in a database, write SQL statements to retrieve, insert, update, and delete rows from the table.
- Start and stop the Derby database server.

Working with Database: Derby Database

Objectives

Knowledge

- Explain how data is organized in a relational database.
- Explain the difference between primary and foreign keys.
- Explain what the SELECT, INSERT, DELETE, and UPDATE statements do.
- Explain what a result set is.
- Explain what a join is, and describe the difference between an inner join and an outer join.
- Describe the characteristics of a Derby database.
- Explain what is meant by an embedded database.
- Describe the use of SQL scripts.

Working with Database: Relational Database

- ▶ **What is a database system?**
 - ▶ A database system consists of:
 - ▶ data
 - ▶ database management software, and
 - ▶ application programs
- ▶ **What is a database application programs**
 - ▶ Programs written on top of database management system
 - ▶ Provides access to DBMS for users of the database
- ▶ **Relational database management system**
 - ▶ Are based on relational data model
 - ▶ structure: representation of data
 - ▶ Integrity: constraints on the data
 - ▶ Language: means to access and manipulate data

Working with Database: Relational Database

- ▶ **Relational Structure:** a relational database consists of set of relations, consisting of
 - ▶ schema: relation itself
 - ▶ instance: snapshot of relation (schema) at any given time
- ▶ **Instance of a schema is a table of rows and columns**
 - ▶ So, an instance of relations is nothing but a table in relational database management system (RDBMBS)

Working with Database: Relational Database

How a table is organized

- A *relational database* uses **tables** to store and manipulate data.
- Each table contains one or more **records**, or **rows**, that contain the data for a single entry.
- Each row contains one or more **fields**, or **columns**, with each column representing a single item of data.
- Most tables contain a **primary key** that uniquely identifies each row in the table.
- The software that manages a relational database is called a *database management system (DBMS)*. Four popular database management systems today are Oracle, Microsoft's **SQL Server**, IBM's **DB2**, and **MySQL**.

Working with Database: Relational Database

The Products table in the MurachDB database

ProductCode	Description	Price
bvbn	Murach's Beginning Visual Basic .NET	49.50
cshp	Murach's C#	49.50
java	Murach's Beginning Java	49.50
jsps	Murach's Java Servlets and JSP	49.50
mcb2	Murach's Mainframe COBOL	59.50
sqls	Murach's SQL for SQL Server	49.50
zjcl	Murach's OS/390 and z/OS JCL	62.50

Working with Database: Relational Database

How the tables in a database are related

- The tables in a relational database are related to each other through their key columns. A column that identifies a related row in another table is called a *foreign key*.
- Three types of *relationships* can exist between tables. The most common type is a *one-to-many relationship*. However, two tables can also have a *one-to-one relationship* or a *many-to-many relationship*.

Working with Database: Relational Database

Two related tables: Products and Lineltems

ProductCode	Description	Price
bvbn	Murach's Beginning Visual Basic .NET	49.50
cshp	Murach's C#	49.50
java	Murach's Beginning Java	49.50
jsps	Murach's Java Servlets and JSP	49.50
mcb2	Murach's Mainframe COBOL	59.50
sqls	Murach's SQL for SQL Server	49.50
zjcl	Murach's OS/390 and z/OS JCL	62.50

LineltemID	InvoiceID	ProductCode	Quantity
1	1	java	5
2	1	jsps	5
3	2	mcb2	1
4	4	cshp	1
8	4	zjcl	2
9	6	sqls	1
10	6	java	1
11	7	mcb2	5

Working with Database: Relational Database

The design of the Products table

Column name	Data type
ProductCode	VARCHAR(10)–Java String
Description	VARCHAR(40)
Price	DOUBLE -Java double

How the columns in a table are defined

- A database management system requires a **name and data type** for each column in a table. Depending on the data type, the column definition can include **other properties** such as the column's size.
- Each column definition also indicates whether or not the column can contain **null values**.
- A column can be defined with a **default value**. Then, that value is used for the column if another value isn't provided when a row is added to the table.

Working with Database: SQL

- ▶ Use Structured Query Language (SQL) to communicate with RDBMS
 - ▶ Data Definition Language (DDL):
 - ▶ Used to define data structures in the database (table, columns etc.)
 - ▶ Data Manipulation Language (DML):
 - ▶ Used to work with the data stored in the database (query, update etc)
 - ▶ Few vocabulary: SELECT, INSERT, UPDATE, DELETE
 - ▶ Use **SELECT** statement of DML to retrieve data from a table
 - ▶ You can filter columns and rows of your choice to retrieve
 - ▶ The result of SELECT is yet another table (to view) and is called: *result table* or *result set*
 - ▶ Row pointer (**cursor**) identifies rows in result set and is used to insert update, delete a row (record)
 - ▶ Smaller result set is better for **efficiency**

Working with Database: SQL

SELECT syntax for selecting (extracting) data from one table

```
SELECT column-1 [, column-2] ...  
FROM table-1  
[WHERE selection-criteria]  
[ORDER BY column-1 [ASC|DESC]  
      [, column-2 [ASC|DESC]] ...]
```

Working with Database: SQL

A SELECT statement that gets selected columns and rows

```
SELECT ProductCode, Description, Price  
FROM Products  
WHERE Price > 50  
ORDER BY ProductCode ASC
```

The **result set** defined by the SELECT statement

ProductCode	Description	Price
mcb2	Murach's Mainframe COBOL	59.5
zjcl	Murach's OS/390 and z/OS JCL	62.5

A SELECT statement that returns all columns and rows

```
SELECT * FROM Products
```

Working with Database: SQL

SELECT syntax for joining two tables

```
SELECT column-1 [, column-2] ...  
FROM table-1  
    {INNER | LEFT OUTER | RIGHT OUTER} JOIN table-2  
    ON table-1.column-1 {=<|>|<=|>=|<>} table-2.column-2  
[WHERE selection-criteria]  
[ORDER BY column-1 [ASC|DESC] [, column-2 [ASC|DESC]] ...]
```

- You can **join** two or more tables in the SELECT to get a composite view of the data in one result set
- An **inner** join or equi-join gives rows from both the tables if the related **columns match**.
- An **outer join** returns rows from one table in the join even if it is not matching rows on other table.
- **LEFT OUTER** will return all rows from the table on left side of join and **RIGHT OUTER** will return all rows from the table on right side

Working with Database: SQL

A SELECT statement that retrieves data from the Products and LineItems tables

```
SELECT p.ProductCode, p.Price, li.Quantity,  
       p.Price * li.Quantity AS Total  
FROM Products p  
     INNER JOIN LineItems li  
     ON p.ProductCode = li.ProductCode  
WHERE p.Price > 50  
ORDER BY p.ProductCode ASC;
```

The result set defined by the SELECT statement

ProductCode	Price	Quantity	Total
mcb2	59.50	1	59.50
mcb2	59.50	5	297.50
zjcl	62.50	2	125.00

Working with Database: SQL

Another way to retrieve data from the Products and LineItems tables

```
SELECT p.ProductCode, p.Price, li.Quantity,  
       p.Price * li.Quantity AS Total  
FROM Products p, LineItems li  
WHERE p.ProductCode = li.ProductCode AND p.Price > 50  
ORDER BY p.ProductCode ASC;
```

The result set defined by the SELECT statement

ProductCode	Price	Quantity	Total
mcb2	59.50	1	59.50
mcb2	59.50	5	297.50
zjcl	62.50	2	125.00

Working with Database: SQL

How to **add** rows

INSERT syntax for adding a single row

```
INSERT INTO table-name [(column-list)]  
VALUES (value-list)
```

A statement that adds a row **with a column list**

```
INSERT INTO Products (ProductCode, Description, Price)  
VALUES ('casp', 'ASP.NET Web Programming with C#', 54.50)
```

A statement that adds a row **without a column list**

```
INSERT INTO Products  
VALUES ('casp', 'ASP.NET Web Programming with C#', 54.50)
```

Working with Database: SQL

How to **update** rows

UPDATE syntax

```
UPDATE table-name  
SET expression-1 [, expression-2] ...  
[WHERE selection-criteria]
```

A statement that updates **a single row**

```
UPDATE Products  
SET Description =  
    'Murach''s ASP.NET Web Programming with C#',  
    Price = 49.50  
WHERE ProductCode = 'casp'
```

A statement that updates **multiple rows**

```
UPDATE Products  
SET Price = 49.95  
WHERE Price = 49.50
```

Working with Database: SQL

How to **delete** rows

DELETE syntax

```
DELETE FROM table-name  
[WHERE selection-criteria]
```

A statement that deletes **a single row**

```
DELETE FROM Products WHERE ProductCode = 'casp'
```

A statement that deletes **multiple rows**

```
DELETE FROM Invoices WHERE AmountDue = 0
```

A statement that deletes **all rows**

```
DELETE FROM Invoices
```

Working with Database: Derby Database

- ▶ Apache Derby is a open source Relational Database Management System (RDBMS) – like Oracle, MySql etc.
 - ▶ Can embed in Java application (embedded database) or run as client/server (networked database)
 - ▶ JDK 1.6+ includes Derby – Java DB in the distribution
- ▶ Java Database Connectivity (JDBC) is a Java API that provides classes to manipulate data in a relational database management systems (RDBMS) like Derby

Working with Database: Derby Database

The Apache Derby website

`http://db.apache.org/derby/`

Derby pros

- Is inexpensive (free), uses only 2.6 MB
- Is platform-independent, included with JDK
- Is based on the Java, JDBC, and SQL standards.
- Can be run on the client and embedded in a Java application.
- Can be run on a server and accessed by multiple clients.

Mac OS X note

- Derby isn't included as part of the JDK for Mac OS X
- Install it manually

Working with Database: Derby Database

Derby cons

- Doesn't provide easy-to-use GUI tools.
 - Apache FAQ says: “No, Derby does not include a GUI. However, the web site includes a writeup on how to use SQuirreL SQL with Derby, and the UsesOfDerby Wiki page list GUIs that work with Derby.”
- Not designed for large, enterprise databases.

Working with Database: Derby Database

A typical Derby installation directory-Windows

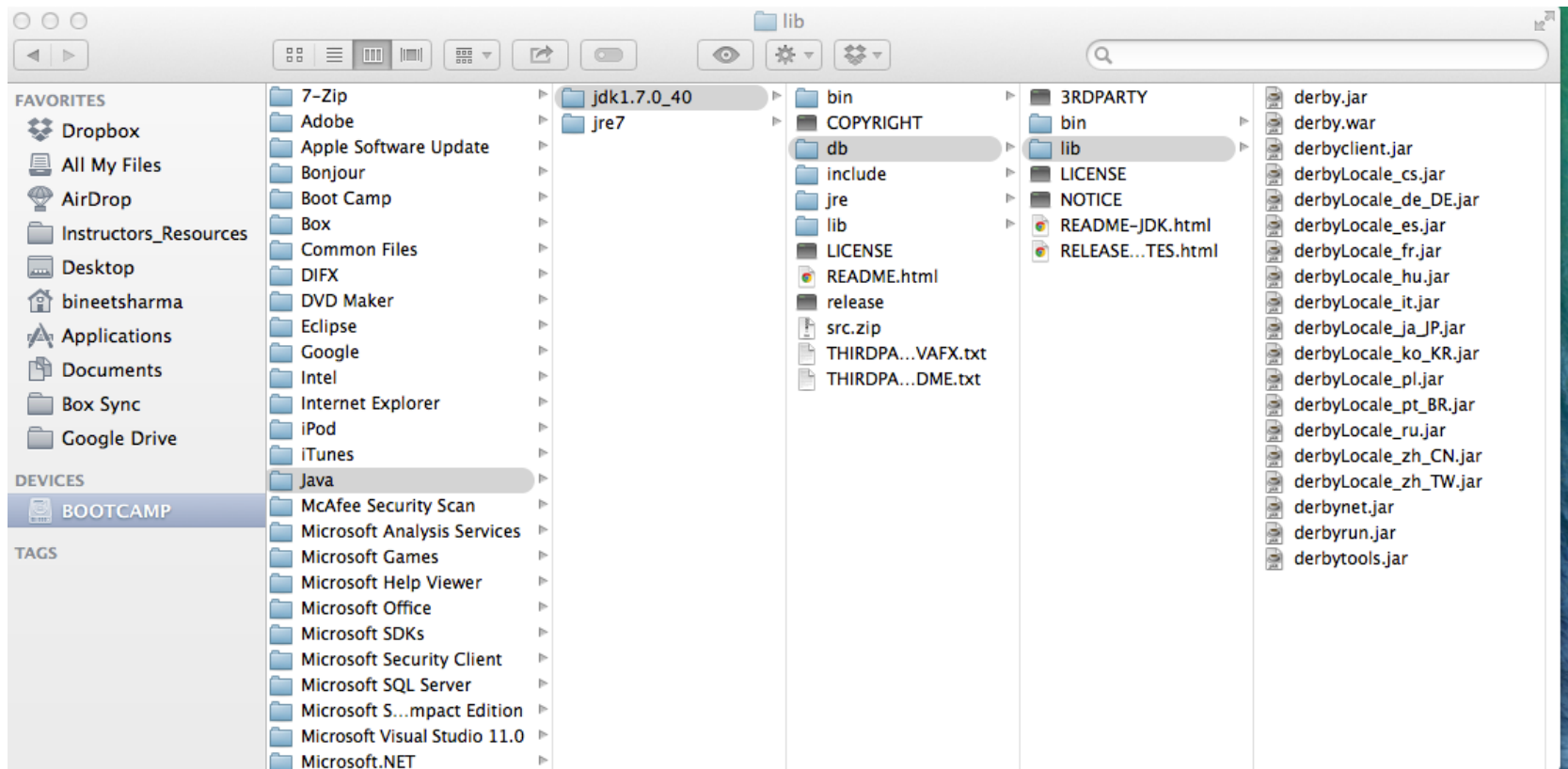
C:\Program Files\Java\jdk1.7.0\db

Four important Java Archive (JAR) files in the db\lib directory

Filename	Description
derby.jar	Needed to work with any Derby database.
derbytools.jar	Needed to run the ij tool described in the next figure.
derbynet.jar	Needed to start the server for a networked Derby database.
derbyclient.jar	Needed to allow a client to connect to a networked Derby database.

Working with Database: Derby Database

A typical Derby installation directory-**Windows**



Working with Database: Derby Database

CLASSPATH

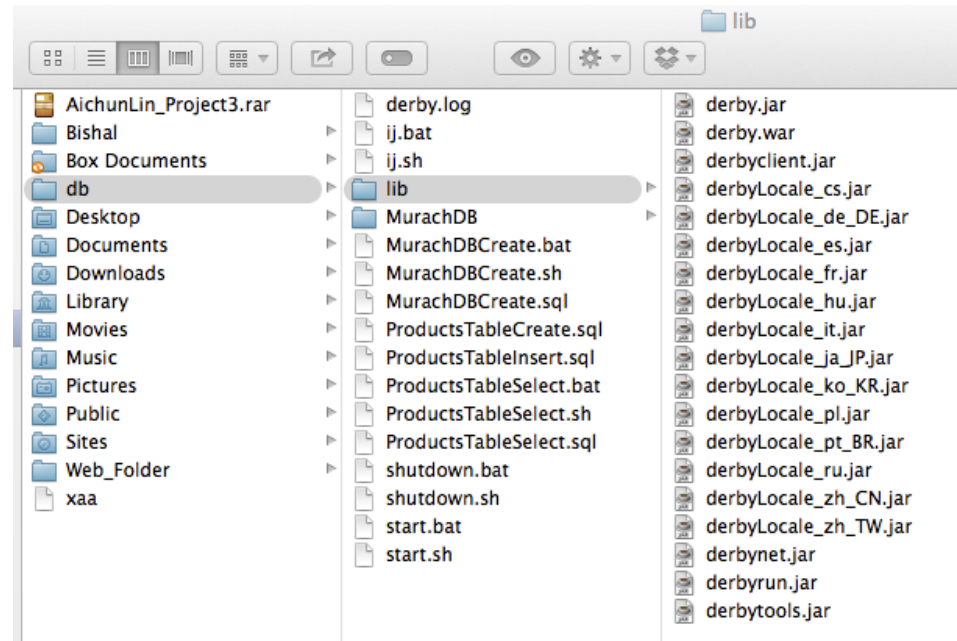
- **CLASSPATH** is required by JVM (when you issue **java** command), to locate a .class file.
- These JAR files are **not automatically included** in the JRE, so, you must include them in the class path, before you can work with the Derby
- You can include a “.”;” to include current directory in the class path

A class path for Windows that contains the current directory and four Derby JAR files (Windows)

```
.;  
C:\Program Files\Java\jdk1.7.0\db\lib\derby.jar;  
C:\Program Files\Java\jdk1.7.0\db\lib\derbytools.jar;  
C:\Program Files\Java\jdk1.7.0\db\lib\derbynet.jar;  
C:\Program Files\Java\jdk1.7.0\db\lib\derbyclient.jar;
```

For MAC it may look like this:

```
.;  
/Users/bineetsharma/db/lib/derby.jar;  
/Users/bineetsharma/db/lib/derbytools.jar;  
/Users/bineetsharma/db/lib/derbynet.jar;  
/Users/bineetsharma/db/lib/derbyclient.jar;
```



Working with Database: Derby Database

- ▶ JDK provides an interactive tool – **interactive JDBC (ij)** to work with the a Derby database (RDBMS)
- ▶ ij is fully written in java, so, you start the tool with java command (JVM).
- ▶ It opens an interactive session, you can exit ij, by entering exit.
- ▶ Use connect to connect to a database by providing a connection string (in a single quote)
- ▶ Use disconnect to disconnect from that database
- ▶ Use create to create a brand new database
- ▶ Each command needs to be followed by a semicolon

Working with Database: Derby Database

A typical ij session to connect to an existing database

```
C:\Users\Joel>cd \murach\java\db  
  
C:\murach\java\db>java org.apache.derby.tools.ij  
ij version 10.8  
ij> connect 'jdbc:derby:MurachDB';  
ij> disconnect;  
ij> exit;  
  
C:\murach\java\db>
```

The **error** message that's displayed if the database doesn't exist

```
C:\murach\java\db>java org.apache.derby.tools.ij  
ij version 10.8  
ij> connect 'jdbc:derby:BineetDB';  
ERROR XJ004: Database 'BineetDB' not found.  
ij> exit;
```

Working with Database: Derby Database

How to **create** a database and connect to it

```
C:\Users\Joel>cd \murach\java\db

C:\murach\java\db>java org.apache.derby.tools.ij
ij version 10.8
ij> connect 'jdbc:derby:MurachDB;create=true';
ij> disconnect;
ij> exit;

C:\murach\java\db>
```

Working with Database: Derby Database

How to run **SQL** statements against a database

```
C:\murach\java\db>java org.apache.derby.tools.ij
ij version 10.8
ij> connect 'jdbc:derby:MurachDB';
ij>

CREATE TABLE Products
(
    ProductCode VARCHAR(10),
    Description VARCHAR(40),
    Price DOUBLE
);
0 rows inserted/updated/deleted
ij>

INSERT INTO Products
VALUES ('java', 'Murach's Beginning Java', 49.50);
1 row inserted/updated/deleted
ij>
```

Working with Database: Derby Database

How to run SQL statements against a database (cont.)

```
SELECT * FROM Products;  
PRODUCTC&|DESCRIPTION
```

```
| PRICE
```

```
-----  
java      |Murach's Beginning Java
```

```
| 49.5
```

```
1 row selected
```

```
ij> exit;
```

```
C:\murach\java\db>
```

Working with Database: Derby Database

The SQL statement **stored** in the **ProductsTableCreate.sql** file

```
CREATE TABLE Products
(
    ProductCode VARCHAR(10),
    Description VARCHAR(40),
    Price DOUBLE
)
```

The SQL statements stored in the **ProductsTableInsert.sql** file

```
INSERT INTO Products
VALUES ('java', 'Murach's Beginning Java', 49.50);
```

```
INSERT INTO Products
VALUES ('jsp', 'Murach's Java Servlets and JSP', 49.50);
```


Working with Database: Derby Database

How to **run SQL scripts** from the ij prompt

```
C:\murach\java\db>java org.apache.derby.tools.ij
ij version 10.8
ij> connect 'jdbc:derby:MurachDB';
ij> run 'ProductsTableCreate.sql';
ij> CREATE TABLE Products
(
    ProductCode VARCHAR(10),
    Description VARCHAR(40),
    Price DOUBLE
);
0 rows inserted/updated/deleted
ij> run 'ProductsTableInsert.sql';
ij> INSERT INTO Products
VALUES ('java', 'Murach's Beginning Java', 49.50);
1 row inserted/updated/deleted
ij> INSERT INTO Products
VALUES ('jsp', 'Murach's Java Servlets and JSP', 49.50);
1 row inserted/updated/deleted
ij> disconnect;
ij> exit;
```

Working with Database: Derby Database

Part of the **MurachDBCreate.sql** file

```
CONNECT 'jdbc:derby:MurachDB;create=true';
```

```
CREATE TABLE Products
```

```
(  
    ProductCode VARCHAR(10),  
    Description VARCHAR(40),  
    Price DOUBLE  
);
```

```
INSERT INTO Products VALUES
```

```
('bvbn', 'Murach's Beginning Visual Basic .NET', 49.50);
```

```
INSERT INTO Products VALUES
```

```
('cshp', 'Murach's C#', 49.50);
```

```
INSERT INTO Products VALUES
```

```
('java', 'Murach's Beginning Java', 49.50);
```

```
INSERT INTO Products VALUES
```

```
('jsps', 'Murach's Java Servlets and JSP', 49.50);
```

```
INSERT INTO Products VALUES
```

```
('mcb2', 'Murach's Mainframe COBOL', 59.50);
```

Working with Database: Derby Database

Part of the MurachDBCreate.sql file (cont.)

```
INSERT INTO Products VALUES
('sqls', 'Murach's SQL for SQL Server', 49.50);
INSERT INTO Products VALUES
('zjcl', 'Murach's OS/390 and z/OS JCL', 62.50);

SELECT * FROM Products;
```

How to **run a script** from the command prompt

```
C:\murach\java\db>java
org.apache.derby.tools.ij MurachDBCreate.sql
```

How to run a script from a DOS batch file

```
cd \murach\java\db
java org.apache.derby.tools.ij MurachDBCreate.sql
pause
```

Working with Database: Derby Database

► A Typical ij session in a mac computer:

```
ij> cd /Users/bineetsharma/Downloads/db-derby-10.10.1.1-bin/bin
bineetsarmasmbp:bin bineetsharma$ pwd
/Users/bineetsharma/Downloads/db-derby-10.10.1.1-bin/bin
bineetsarmasmbp:bin bineetsharma$ ls -ltr
total 208
-rwxr-xr-x@ 1 bineetsharma  staff  1389 Mar 18  2013 sysinfo.bat
-rwxr-xr-x@ 1 bineetsharma  staff  1403 Mar 18  2013 stopNetworkServer.bat
-rwxr-xr-x@ 1 bineetsharma  staff  1397 Mar 18  2013 startNetworkServer.bat
-rwxr-xr-x@ 1 bineetsharma  staff  1273 Mar 18  2013 setNetworkServerCP.bat
-rwxr-xr-x@ 1 bineetsharma  staff  1075 Mar 18  2013 setNetworkServerCP
-rwxr-xr-x@ 1 bineetsharma  staff  1284 Mar 18  2013 setNetworkClientCP.bat
-rwxr-xr-x@ 1 bineetsharma  staff  1079 Mar 18  2013 setNetworkClientCP
-rwxr-xr-x@ 1 bineetsharma  staff  1278 Mar 18  2013 setEmbeddedCP.bat
-rwxr-xr-x@ 1 bineetsharma  staff  1073 Mar 18  2013 setEmbeddedCP
-rwxr-xr-x@ 1 bineetsharma  staff  1379 Mar 18  2013 ij.bat
-rwxr-xr-x@ 1 bineetsharma  staff  2426 Mar 18  2013 derby_common.bat
-rwxr-xr-x@ 1 bineetsharma  staff  1387 Mar 18  2013 dblook.bat
-rwxr-xr-x@ 1 bineetsharma  staff  1413 Mar 18  2013 NetworkServerControl.bat
-rwxr-xr-x@ 1 bineetsharma  staff  5789 Mar 19  2013 sysinfo
-rwxr-xr-x@ 1 bineetsharma  staff  5810 Mar 19  2013 stopNetworkServer
-rwxr-xr-x@ 1 bineetsharma  staff  5807 Mar 19  2013 startNetworkServer
-rwxr-xr-x@ 1 bineetsharma  staff  5876 Mar 19  2013 ij
-rwxr-xr-x@ 1 bineetsharma  staff  5740 Mar 19  2013 dblook
-rwxr-xr-x@ 1 bineetsharma  staff  5801 Mar 19  2013 NetworkServerControl
-rw-r--r--  1 bineetsharma  staff    701 Feb 27 16:16 derby.log
```

Working with Database: Derby Database

► A Typical ij session:

```
-rw-r--r--  1 bineetsharma  staff   701 Feb 27 16:16 derby.log
bineetsarmasmbp:bin bineetsharma$ ./ij
ij version 10.10
ij> connect 'jdbc:derby:/Users/bineetsharma/db/MurrachDB';
ERROR XJ004: Database '/Users/bineetsharma/db/MurrachDB' not found.
ij> connect 'jdbc:derby:/Users/bineetsharma/db/MurachDB';
ij> select * from products;
PRODUCTCO&|DESCRIPTION                                |PRICE
-----|-----
bvbvn      |Murach's Beginning Visual Basic .NET                 |49.5
cshp       |Murach's C#                                             |49.5
java       |Murach's Beginning Java                               |49.5
jsp        |Murach's Java Servlets and JSP                        |49.5
mcb2       |Murach's Mainframe COBOL                              |59.5
sqls       |Murach's SQL for SQL Server                           |49.5
zjcl       |Murach's OS/390 and z/OS JCL                          |62.5

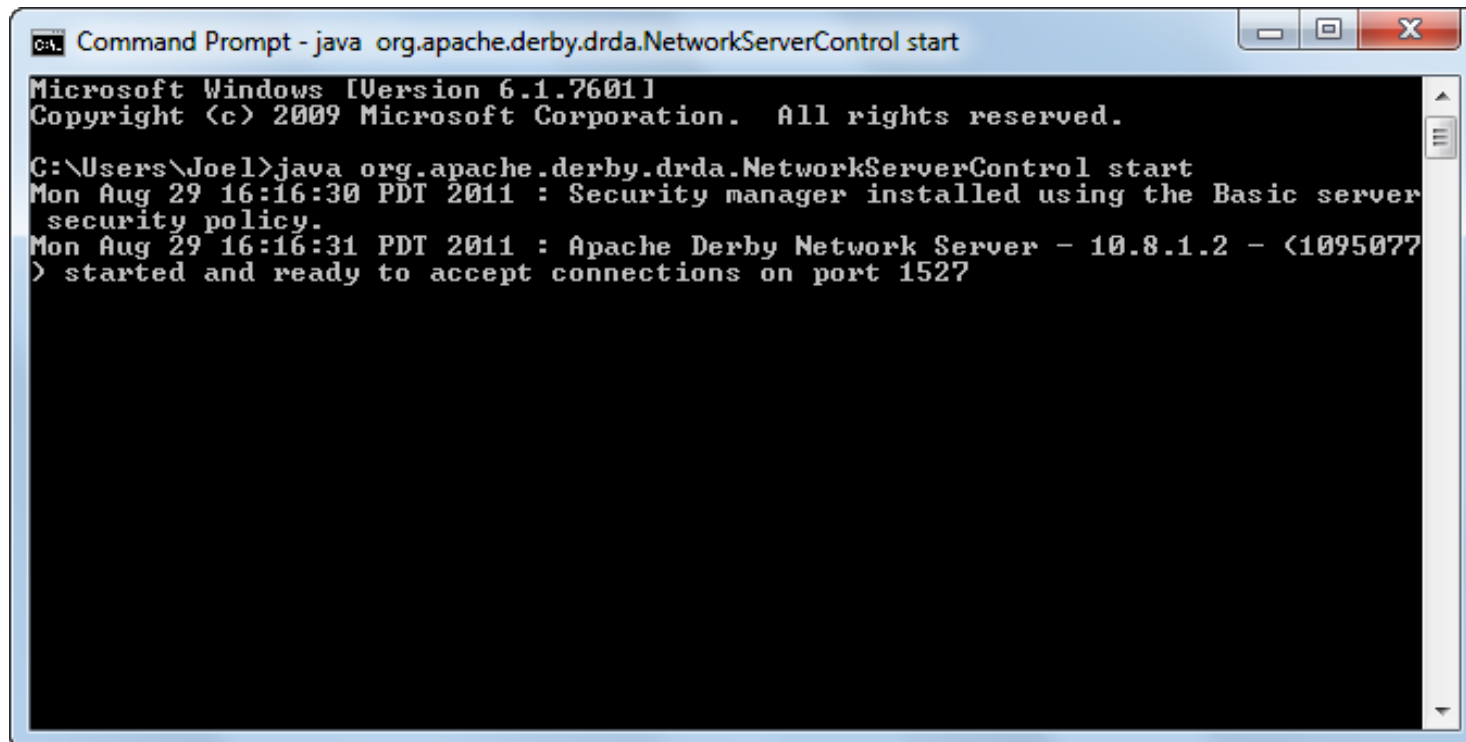
7 rows selected
ij> 
```

Working with Database: Derby Database

► The command to start the Derby server

```
java org.apache.derby.drda.NetworkServerControl start
```

The window that's displayed when the Derby server is running



```
Command Prompt - java org.apache.derby.drda.NetworkServerControl start

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

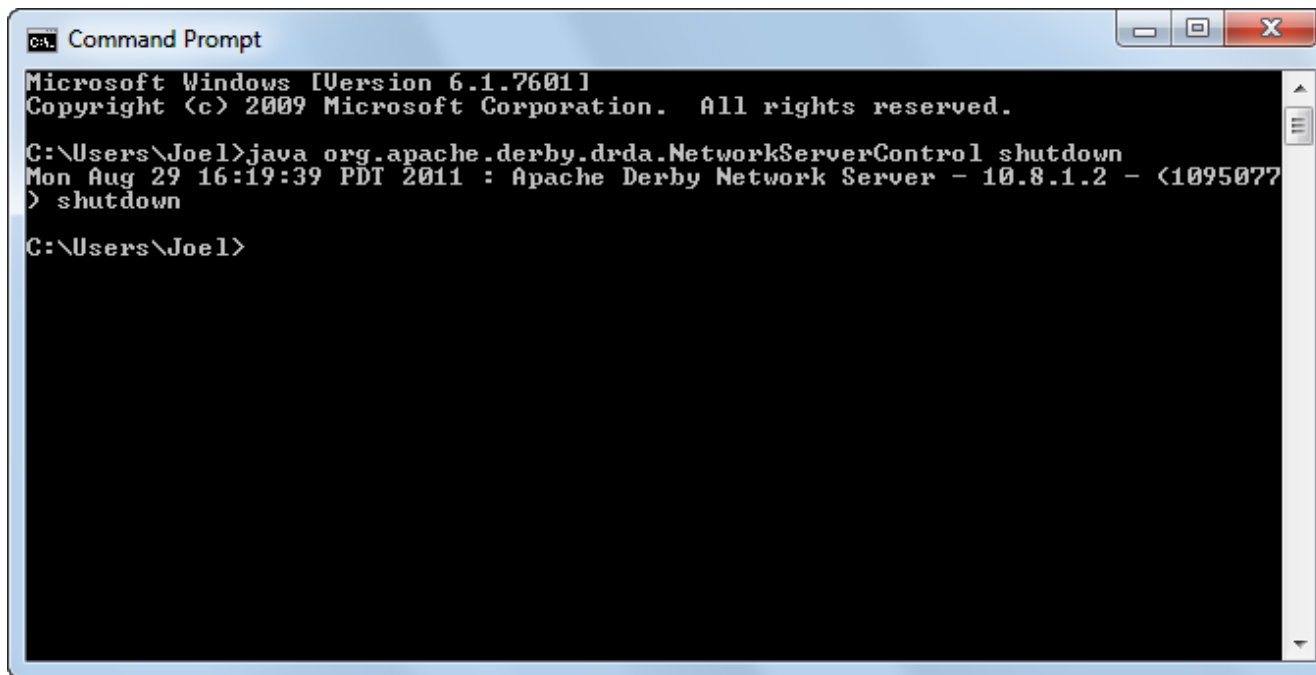
C:\Users\Joel>java org.apache.derby.drda.NetworkServerControl start
Mon Aug 29 16:16:30 PDT 2011 : Security manager installed using the Basic server
security policy.
Mon Aug 29 16:16:31 PDT 2011 : Apache Derby Network Server - 10.8.1.2 - <1095077
> started and ready to accept connections on port 1527
```

Working with Database: Derby Database

- ▶ The command to **stop** the Derby server

```
java org.apache.derby.drda.NetworkServerControl shutdown
```

The message that's displayed when the Derby server is stopped



```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Joel>java org.apache.derby.drda.NetworkServerControl shutdown
Mon Aug 29 16:19:39 PDT 2011 : Apache Derby Network Server - 10.8.1.2 - (1095077)
> shutdown

C:\Users\Joel>
```

Working with Database: **Derby Database**

The Derby documentation

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

<http://db.apache.org/derby/faq.html>

The Derby manuals that are available online

- Getting Started with Derby
- Derby Reference Manual, Derby Developer's Guide
- Tuning Derby
- Derby Server and Administration Guide
- Derby Tools and Utilities Guide

The Derby **tutorials**

- http://db.apache.org/derby/papers/DerbyTut/ij_intro.html
- https://db.apache.org/derby/papers/DerbyTut/install_software.html

Working with JDBC

Working with Database: JDBC

Objectives

Applied

- Add a database driver for any database to a project.
- Given the URL, username, and password required to connect to a database, write the code necessary to create a Connection object that connects to the database.
- Write code that executes a SELECT statement and processes the results using a forward-only or scrollable result set.
- Write code that inserts, updates, or deletes rows in a table.
- Given code that executes a SQL statement, modify the code so it uses a prepared statement and parameters.

Working with Database: JDBC

Objectives (cont.)

- Given a result set, write code that determines the name and data type of each of its columns.
- Given the specifications for an application that stores its data in a relational database, implement the program using the JDBC features presented in this chapter.
- Write the Java code for connecting to and disconnecting from an embedded Derby database.
- Given the specifications for an application that uses an embedded Derby database, develop the application.

Knowledge

- Identify the four types of Java database drivers and describe the benefits and drawbacks of each driver type.
- Describe the use of automatic driver loading and iterable SQL exceptions.

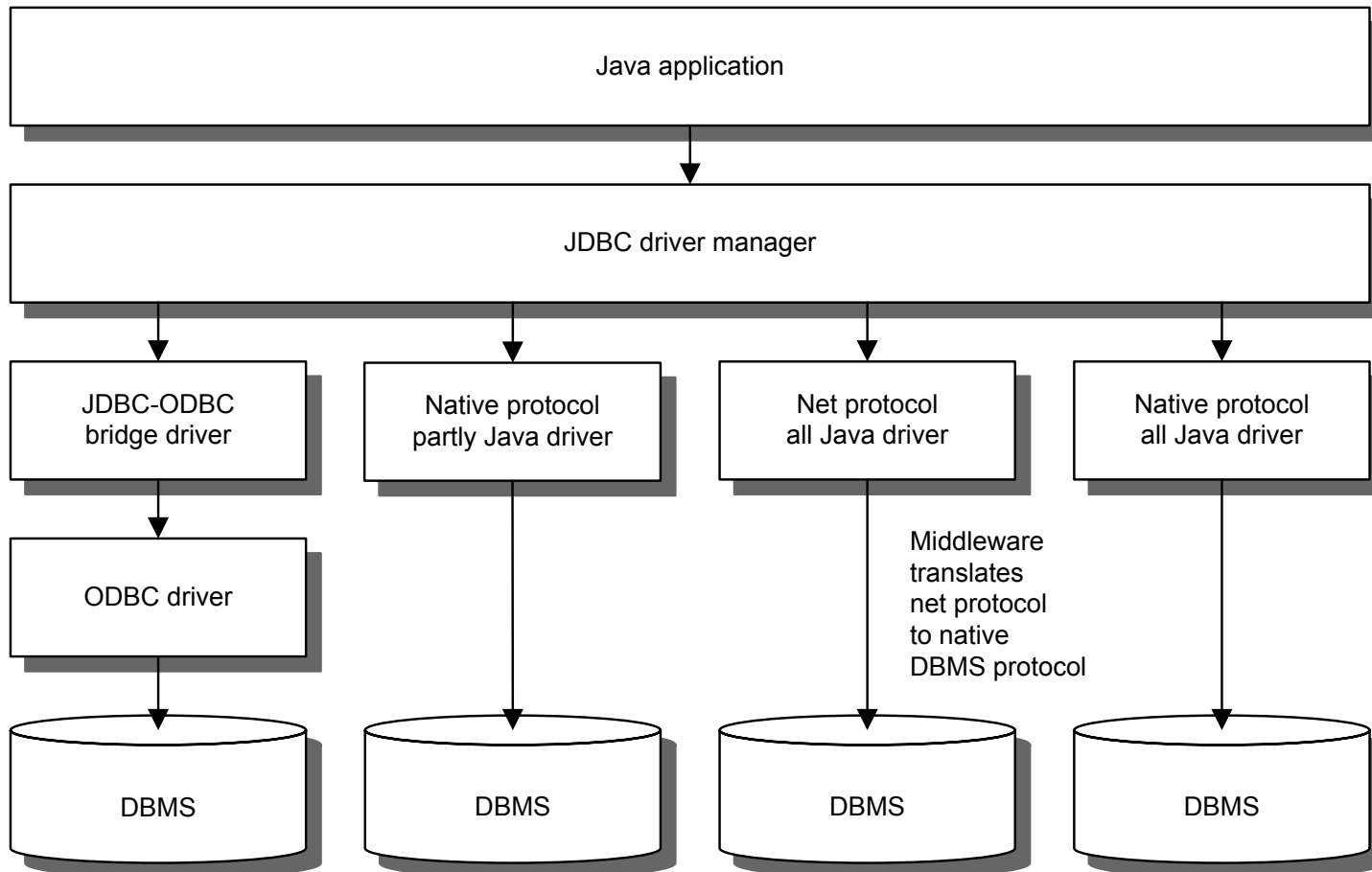
Working with Database: JDBC

Objectives (cont.)

- Explain the difference between a forward-only, read-only result set and a scrollable, updateable result set.
- Explain what metadata is and how it can be used in database applications.

Working with Database: JDBC

Four ways to access a database



Working with Database: JDBC

The four types of Java drivers

- Type 1 *A JDBC-ODBC (Open Database Connectivity) bridge driver* converts JDBC calls into ODBC calls that access the DBMS protocol. This data access method requires that the ODBC driver be installed on the client machines (driver exists for almost all current databases).
- Type 2 *A native protocol partly Java driver* converts JDBC calls into calls in the native DBMS protocol. Since this conversion takes place on the client, **some binary code must be installed** on the client machine.
- Type 3 *A net protocol all Java driver* converts JDBC calls into a net protocol that's **independent of any native DBMS** protocol. Then, **middleware software** running on a server **converts the net protocol to the native DBMS** protocol. Since this conversion takes place on the server side, no installation is required on the client machine.

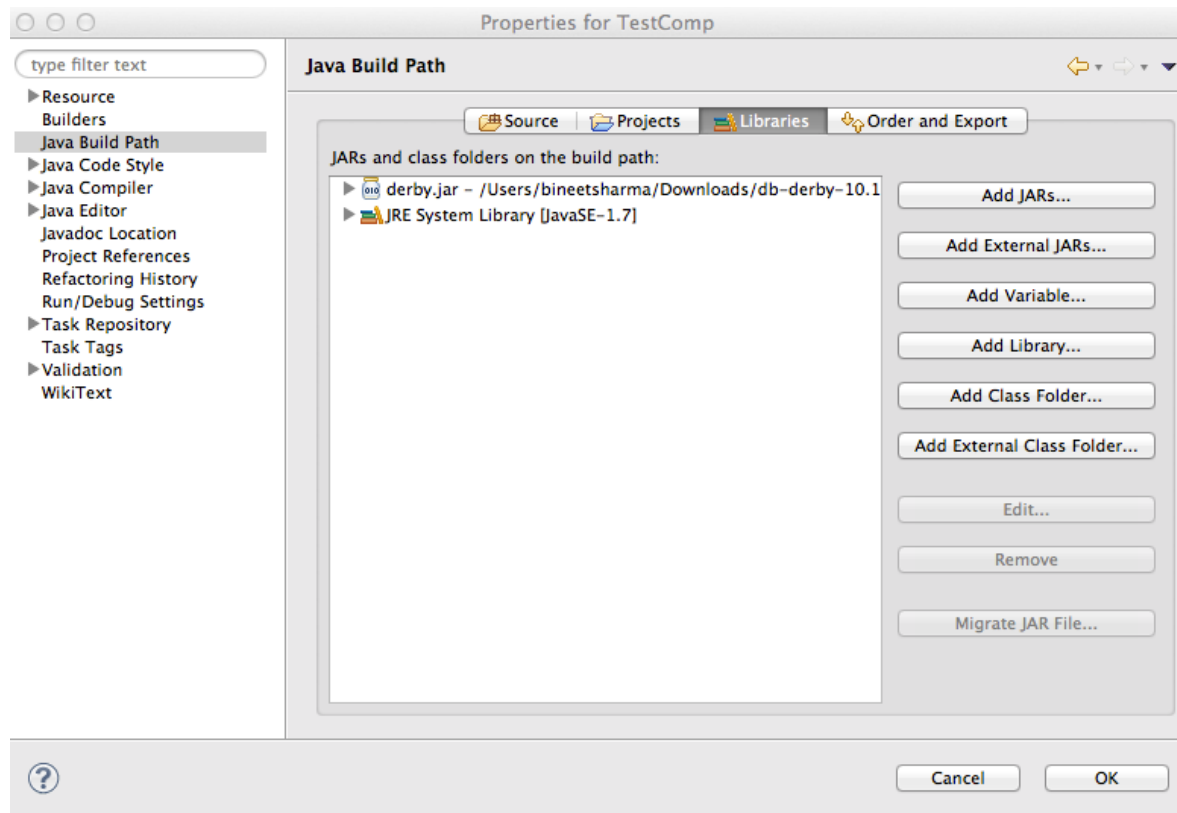
Working with Database: JDBC

The four types of Java drivers (cont.)

Type 4 *A native protocol all Java driver* converts JDBC calls into a native DBMS protocol. Since this conversion takes place on the server side, **no installation is required** on the client machine.

Working with Database: JDBC

A project after the **Derby database driver** has been added (**derby.jar** has classes to work with an embedded Derby database in a Java file)



Working with Database: JDBC

How to **connect** with a type-4 driver from java app

```
private Connection getConnection(){
    Connection connection = null;
    try {
        // if necessary, set the home directory for Derby
        String dbDirectory = "c:/murach/java/db";
        System.setProperty("derby.system.home",
            dbDirectory);
        // create and return the connection
        String dbUrl = "jdbc:derby:MurachDB";
        String username = "AppUser";
        String password = "sesame!";
        connection = DriverManager.getConnection(
            dbUrl, username, password);
        return connection;
    }
    catch (SQLException e) {
        for (Throwable t : e)
            e.printStackTrace();
        return null;
    }
}
```

Working with Database: JDBC

The URL for connecting to a database named MurachDB

```
String dbUrl = "jdbc:derby:MurachDB";
```

The URL for creating a database named MurachDB if it doesn't exist

```
String dbUrl = "jdbc:derby:MurachDB;create=true";
```

Working with Database: JDBC

How to **disconnect** from all databases and shut down the database engine

```
public boolean disconnect()
{
    try
    {
        // On a successful shutdown, this throws
        // an exception
        String shutdownURL = "jdbc:derby;;shutdown=true";
        DriverManager.getConnection(shutdownURL);
    }
    catch (SQLException e)
    {
        if (e.getMessage().equals(
            "Derby system shutdown."))
            return true;
    }
    return false;
}
```

Working with Database: JDBC

The URL for **disconnecting from all databases**

```
String shutdownURL = "jdbc:derby;;shutdown=true";
```

The URL for **disconnecting from a specific database**

```
String shutdownURL = "jdbc:derby:MurachDB;shutdown=true";
```

URL syntax

```
jdbc:subprotocolName:databasePath
```

A Derby database on a **local computer** in the default directory

```
jdbc:derby://localhost:1527/MurachDB
```

A Derby database on a **local computer** in a **specific directory**

```
jdbc:derby://localhost:1527/murach/java/db/MurachDB
```

A Derby database **on a server** in the default **directory**

```
jdbc:derby://DBSERVER:1527/Databases/MurachDB
```

Working with Database: JDBC

A **MySQL** database on a local computer

`jdbc:mysql://localhost:3306/MurachDB`

An **Oracle** database **on a local computer**

`jdbc:oracle:thin:@localhost:1521/XE`

An Oracle database **on a server**

`jdbc:oracle:thin:@//DBSERVER:1521/ORCL`

Working with Database

- ▶ Using Java to work with a database through JDBC API
 - ▶ JDBC API code works with any type of database
- ▶ Once the connection to database is established:
 - ▶ You typically retrieve bunch of data using a query into a result set
 - ▶ Result-set is a sub-set of table of rows and columns of data selected
 - ▶ You use the result set to insert, update and delete records
 - ▶ There are forward-only, scrollable, and updateable result sets.
- ▶ Use `createStatement` method of connection object to get the statement object which helps in executing various SQL statements

Working with Database: JDBC

How to create a forward-only, read-only result set

```
Statement statement = connection.createStatement();  
ResultSet rs = statement.executeQuery(  
    "SELECT * FROM Products");
```

How to create a scrollable, updateable result set

```
Statement statement = connection.createStatement(  
    ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);  
  
String query = "SELECT ProductCode, Description, Price "  
    + "FROM Products ORDER BY ProductCode ASC";  
  
ResultSet rs = statement.executeQuery(query);
```

Working with Database: JDBC

ResultSet fields that set type and concurrency

- **TYPE_FORWARD_ONLY** Cursor moves forward only
- **TYPE_SCROLL_INSENSITIVE** Moves any direction, but, the changes by others won't reflect
- **TYPE_SCROLL_SENSITIVE** Displays the changes too
- **CONCUR_READ_ONLY** Read-only (by default)
- **CONCUR_UPDATABLE** Can update

Which one to use? Most versatile would require most resources – so, they are slow as well.

Working with Database: JDBC

Methods of a ResultSet object that work with a result set

- `beforeFirst()` Moves cursor before the first row (that is the position when you get the result set)
- `afterLast()`
- `first()` Moves the cursor at first row returns true if successful, null if fails - row does not exists
- `previous()` Like `first()`
- `next()` Like `first()`
- `last()` Like `first()`
- `absolute(intRow)` Moves the cursor to the specified row
- `relative(intRow)` Moves the cursor relative to current
- `isBeforeFirst()` Returns true if cursor is before first
- `isAfterLast()`
- `isFirst()`
- `isLast()`
- `close()` Releases the JDBC and database resources

Working with Database: JDBC

How to work with a forward-only result set

```
while(rs.next())
{
    // code that works with each row
}
rs.close();
```

How to work with a scrollable result set

```
rs.first();
rs.last();

if (rs.isFirst() == false)
    rs.previous();

if (rs.isLast() == false)
    rs.next();

rs.absolute(4);
rs.relative(-2);
rs.relative(3);
```

Working with Database: JDBC

- ▶ Once you have the result set, you can use various methods to retrieve the data from the result set
 - ▶ `getString()`, `getInt()`, `getDouble()` etc returns respective data
 - ▶ It has methods for all 8 primitive data types and more
 - ▶ `getDate()`, `getTime()`, and `getTimestamp()` returns `Date`, `Time`, and `TimeStamp` object of `java.sql` package
- ▶ Result set even have methods to get large sets of data
 - ▶ `getBlob()`: returns BLOB objects (Binary Large Objects)
 - ▶ `getClob()`: returns CLOB objects (Character Large Objects)

Working with Database: JDBC

Methods of a ResultSet object that return data from a result set

- `getString(intColumnIndex)` Returns a string (uses col#)
- `getString(StringColumnName)` Returns a string (uses name)
- `getDouble(intColumnIndex)` Returns double (uses col#)
- `getDouble(StringColumnName)` (uses column name)

Code that uses column indexes to return fields from the Products result set

```
String code = rs.getString(1);  
String description = rs.getString(2);  
double price = rs.getDouble(3);
```

Code that uses column names to return the fields

```
String code = rs.getString("ProductCode");  
String description = rs.getString("Description");  
double price = rs.getDouble("Price");
```

Working with Database: JDBC

Code that creates a Product object from the Products result set

```
Product p = new Product(rs.getString("ProductCode"),  
                        rs.getString("Description"),  
                        rs.getDouble("Price"));
```

Working with Database: JDBC

- ▶ You can use `executeUpdate` method to modify data
 - ▶ INSERT, UPDATE, DELETE statements can be executed (JDBC 1.0+) using `executeUpdate`

How to use the `executeUpdate` method to modify data

How to add a record

```
String insertStatement = "INSERT INTO Products " +  
    "(ProductCode, Description, Price) " +  
    "VALUES ('" + p.getCode() + "', " +  
        "'" + p.getDescription() + "', " +  
            p.getPrice() + ")";  
int count = statement.executeUpdate(insertStatement);
```

How to update a record

```
String updateStatement =  
    "UPDATE Products SET " +  
        "ProductCode = '" + p.getCode() + "', " +  
        "Description = '" + p.getDescription() + "', " +  
        "Price = " + p.getPrice() + " " +  
    "WHERE ProductCode = '" + p.getCode() + "'";  
int count = statement.executeUpdate(updateStatement);
```

Working with Database: JDBC

How to use the executeUpdate method to modify data (cont.)

How to delete a record

```
String deleteStatement =  
    "DELETE FROM Products " +  
    "WHERE ProductCode = '" + p.getCode() + "'";  
int count = statement.executeUpdate(deleteStatement);
```

Working with Database: JDBC

Modify data **JDBC 2.0** (JDK 1.4+)

How to add a record

```
rs.moveToInsertRow();  
rs.updateString("ProductCode", p.getCode());  
rs.updateString("Description", p.getDescription());  
rs.updateDouble("Price", p.getPrice());  
rs.insertRow();  
rs.moveToCurrentRow(); /* to move the cursor to the  
row in the ResultSet that was the current row before  
the insert operation occurred.*/
```

How to update a record

```
rs.updateString("ProductCode", p.getCode());  
rs.updateString("Description", p.getDescription());  
rs.updateDouble("Price", p.getPrice());  
rs.updateRow();
```

How to delete a record

```
rs.deleteRow();
```


Working with Database: JDBC

- ▶ Instead of executing every SQL statements right away, you can choose to execute SQL statements in two steps
- ▶ First: prepare the statements (it is like compile it), leave room (a place holder) for any parameters which can be passed later
- ▶ Second: supply the values for all those parameters and then execute the statements.
- ▶ This is much more efficient and is followed in programming more often

Working with Database: JDBC

How to use a prepared statement to return a result set

```
String selectProduct =  
    "SELECT ProductCode, Description, Price " +  
    "FROM Products " +  
    "WHERE ProductCode = ?";  
PreparedStatement ps =  
    connection.prepareStatement(selectProduct);  
ps.setString(1, p.getCode());  
ResultSet rs = ps.executeQuery();
```

How to use a prepared statement to update a record

```
String updateProduct =  
    "UPDATE Products " +  
    "SET Description = ?, Price = ? " +  
    "WHERE ProductCode = ?";  
PreparedStatement ps =  
    connection.prepareStatement(updateProduct);  
ps.setString(1, p.getDescription());  
ps.setDouble(2, p.getPrice());  
ps.setString(3, p.getCode());  
int count = ps.executeUpdate();
```

Working with Database: JDBC

How to use a prepared statement to **insert** a record

```
String insertProduct = "INSERT INTO Products " +  
    "(ProductCode, Description, Price) " +  
    "VALUES (?, ?, ?)";  
PreparedStatement ps =  
    connection.prepareStatement(insertProduct);  
ps.setString(1, p.getCode());  
ps.setString(2, p.getDescription());  
ps.setDouble(3, p.getPrice());  
int count = ps.executeUpdate();
```

How to use a prepared statement to **delete** a record

```
String deleteProduct =  
    "DELETE FROM Products " +  
    "WHERE ProductCode = ?";  
PreparedStatement ps =  
    connection.prepareStatement(deleteProduct);  
ps.setString(1, p.getCode());  
int count = ps.executeUpdate();
```

Working with Database: JDBC

Dealing with issues with special characters in data

SQL statement that causes an error

```
"INSERT INTO Products " +  
    "(ProductCode, Description, Price) " +  
"VALUES ('java', 'Murach's Beginning Java 2', 49.50)";
```

A SQL statement that works

```
"INSERT INTO Products " +  
    "(ProductCode, Description, Price) " +  
"VALUES ('java', 'Murach's Beginning Java 2', 49.50)";
```

Working with Database: JDBC

A utility class for working with strings

```
public class DBUtils {
    // handle strings that contain one or more
    // apostrophes (')
    private static String fixDBString(String s)
    {
        // if the string is null, return it
        if (s == null)
            return s;

        // add an apostrophe before each existing
        // apostrophe
        StringBuilder sb = new StringBuilder(s);
        for (int i = 0; i < sb.length(); i++)
        {
            char ch = sb.charAt(i);
            if (ch == 39) // ASCII code for an apostrophe
                sb.insert(i++, "'");
        }
        return sb.toString();
    }
}
```

Working with Database: JDBC

Code that uses the utility class for working with strings

```
Statement statement = connection.createStatement();
String insert = "INSERT INTO Products " +
    "(ProductCode, Description, Price) " +
    "VALUES ('" + p.getCode() + "', " +
    "'" + DBUtils.fixDBString(p.getDescription()) +
    "', " + p.getPrice() + ")";
statement.executeUpdate(insert);
```

Putting it all together

Working with JDBC

Code walk through

Working with Database: JDBC

The code for the ProductDB class

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

public class ProductDB implements ProductDAO
{
    private Connection getConnection() {
        Connection connection = null;
        try
        {
            String dbDirectory = "c:/murach/java/db";
            System.setProperty(
                "derby.system.home", dbDirectory);

            String url = "jdbc:derby:MurachDB";
            String username = "";
            String password = "";
            connection = DriverManager.getConnection(
                url, username, password);
            return connection;
        }
    }
}
```


Working with Database: JDBC

The code for the ProductDB class (cont.)

```
        catch(SQLException e)
        {
            System.err.println(e);
            return null;
        }
    }

    public ArrayList<Product> getProducts() {
        String sql = "SELECT ProductCode, Description, "
            + "Price "
            + "FROM Products "
            + "ORDER BY ProductCode ASC";
        ArrayList<Product> products = new ArrayList<>();

        try (Connection connection = getConnection();
            PreparedStatement ps =
                connection.prepareStatement(sql);
            ResultSet rs = ps.executeQuery())
        {
```

Working with Database: JDBC

The code for the ProductDB class (cont.)

```
        while(rs.next())
        {
            String code = rs.getString("ProductCode");
            String description =
                rs.getString("Description");
            double price = rs.getDouble("Price");

            Product p =
                new Product(code, description, price);
            products.add(p);
        }
        return products;
    }
    catch(SQLException e)
    {
        System.err.println(e);
        return null;
    }
}
```

Working with Database: JDBC

The code for the ProductDB class (cont.)

```
public Product getProduct(String code) {
    String sql =
        "SELECT ProductCode, Description, Price " +
        "FROM Products " +
        "WHERE ProductCode = ?";
    try (Connection connection = getConnection();
        PreparedStatement ps =
            connection.prepareStatement(sql))
    {
        ps.setString(1, code);
        ResultSet rs = ps.executeQuery();
        if (rs.next())
        {
            String description =
                rs.getString("Description");
            double price = rs.getDouble("Price");
            Product p =
                new Product(code, description, price);
            rs.close();
            return p;
        }
    }
}
```

Working with Database: JDBC

The code for the ProductDB class (cont.)

```
        else
        {
            rs.close();
            return null;
        }
    }
    catch(SQLException e)
    {
        System.err.println(e);
        return null;
    }
}

public boolean addProduct(Product p) {
    String sql = "INSERT INTO Products " +
        "(ProductCode, Description, Price) " +
        "VALUES (?, ?, ?)";
    try (Connection connection = getConnection();
        PreparedStatement ps =
            connection.prepareStatement(sql))
    {
```

Working with Database: JDBC

The code for the ProductDB class (cont.)

```
        ps.setString(1, p.getCode());
        ps.setString(2, p.getDescription());
        ps.setDouble(3, p.getPrice());
        ps.executeUpdate();
        return true;
    }
    catch(SQLException e)
    {
        System.err.println(e);
        return false;
    }
}

public boolean deleteProduct(Product p)
{
    String sql = "DELETE FROM Products " +
                "WHERE ProductCode = ?";
    try (Connection connection = getConnection();
        PreparedStatement ps =
            connection.prepareStatement(sql))
    {
```

Working with Database: JDBC

The code for the ProductDB class (cont.)

```
        ps.setString(1, p.getCode());
        ps.executeUpdate();
        return true;
    }
    catch(SQLException e)
    {
        System.err.println(e);
        return false;
    }
}

public boolean updateProduct(Product p) {
    String sql = "UPDATE Products SET " +
        "Description = ?, " +
        "Price = ? " +
        "WHERE ProductCode = ?";
    try (Connection connection = getConnection();
        PreparedStatement ps =
            connection.prepareStatement(sql))
    {
```

Working with Database: JDBC

The code for the ProductDB class (cont.)

```
        ps.setString(1, p.getDescription());
        ps.setDouble(2, p.getPrice());
        ps.setString(3, p.getCode());
        ps.executeUpdate();
        return true;
    }
    catch(SQLException e)
    {
        System.err.println(e);
        return false;
    }
}
```

Working with Database: JDBC

At times, you need to know the definition of the result set itself – metadata of result set
Use **getMetaData** method to create a **ResultSetMetaData** object

```
ResultSetMetaData metaData = resultSet.getMetaData();
```

Methods of a ResultSetMetaData object for working with metadata

- **getColumnCount()** Returns count for this result set
- **columnName(intColumn)**
- **columnLabel(intColumn)**
- **columnType(intColumn)**
- **columnName(intColumn)**

Working with Database: JDBC

A method that returns the **column names of a result set**

```
public static ArrayList<String> getColumnNames(  
    ResultSet results) throws SQLException  
{  
    ArrayList<String> columnNames = new ArrayList<>();  
    ResultSetMetaData metaData = results.getMetaData();  
    int columnCount = metaData.getColumnCount();  
    for (int i = 1; i <= columnCount; i++)  
        columnNames.add(metaData.getColumnName(i));  
    return columnNames;  
}
```

Working with Database: JDBC

A method that returns the rows of a result set in a generic way (without knowing name or the index of the column)

```
public static ArrayList<ArrayList> getRows(  
    ResultSet results) throws SQLException  
{  
    ArrayList<ArrayList> rows = new ArrayList<>();  
    ResultSetMetaData metaData = results.getMetaData();  
    while (results.next())  
    {  
        ArrayList<Object> row = new ArrayList<>();  
        for (int i = 1; i <= metaData.getColumnCount();  
            i++)  
        {  
            if (metaData.getColumnType(i) ==  
                Types.VARCHAR)  
                row.add(results.getString(i));  
            else if (metaData.getColumnType(i) ==  
                Types.INTEGER)  
                row.add(new Integer(results.getInt(i)));  
        }  
    }  
}
```

Working with Database: JDBC

A method that returns the rows of a result set (cont.)

```
        else if (metaData.getColumnType(i) ==  
                Types.DOUBLE)  
            row.add(  
                new Double(results.getDouble(i)) );  
        }  
        rows.add(row) ;  
    }  
    return rows;  
}
```

Working with Database: JDBC

How SQL data types map to Java data types

SQL data type	Java data type
VARCHAR, LONGVARCHAR	String
BIT	boolean
TINYBIT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
DOUBLE	double
VARBINARY, LONGVARBINARY	byte[]
NUMERIC	java.math.BigDecimal
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Summary: Working with Database

- ▶ SQL
- ▶ Derby RDBMS
- ▶ JDBC

Next Lecture

- ▶ Working with Java Threads