Database Application Development: JDBC

Slides based on “Database Management Systems” 3rd ed, Ramakrishnan and Gehrke
CREATE TABLE product(
    product_id INT NOT NULL,
    product_code VARCHAR(10) NOT NULL,
    product_description VARCHAR(100) NOT NULL,
    product_price FLOAT NOT NULL,  -- better to use decimal(10,2)
    UNIQUE (product_code),
    PRIMARY KEY (product_id)
);

How to create a result set from product with multiple columns and rows

Statement statement = connection.createStatement();
ResultSet products = statement.executeQuery("SELECT * FROM Product");

How to loop through a result set

while (products.next()) {
    // statements that process each row
}
Code that uses indexes to return columns

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

Code that uses names to return columns (usually a better way)

String code = products.getString("ProductCode");
String description = products.getString("ProductDescription");
double price = products.getDouble("ProductPrice");

Code that creates a Product object (see next slide) from the products result set

Product product = new Product(products.getString(1),
                              products.getString(2),
                              products.getDouble(3));
The Product class: one object can hold data from one row of the products table

```java
package murach.business;

import java.io.Serializable;
import java.text.NumberFormat;

public class Product implements Serializable {

    private String code;
    private String description;
    private double price;

    public Product() {
        code = "";
        description = "";
        price = 0;
    }

    public void.setCode(String code) {
        this.code = code;
    }

    public String getCode() {
        return code;
    }
}
```

In fact, to match the previous code, we need a constructor with parameters code, desc, price—we wrote this out in class.
The Product class (continued)

```java
public void setDescription(String description) {
    this.description = description;
}

public String getDescription() {
    return description;
}

public void setPrice(double price) {
    this.price = price;
}

public double getPrice() {
    return price;
}

public String getPriceCurrencyFormat() {
    NumberFormat currency = NumberFormat.getCurrencyInstance();
    return currency.format(price);
}
```
How to use the `executeUpdate` method to...

Add a row

```java
String query = "INSERT INTO Product (ProductCode, ProductDescription, ProductPrice) VALUES ('" + product.getCode() + ", ";
Statement statement = connection.createStatement(); int rowCount = statement.executeUpdate(query);
```

Update a row

```java
String query = "UPDATE Product SET ProductCode = '" + product.getCode() + ", " + 
"ProductDescription = '" + product.getDescription() + ", " + 
"ProductPrice = '" + product.getPrice() + ") WHERE ProductCode = '" + product.getCode() + ";
Statement statement = connection.createStatement(); int rowCount = statement.executeUpdate(query);
```
Using JDBC

• 3 steps to submit a database query:

1. Load the JDBC driver (not needed explicitly in JDBC 4.0)

2. Connect to the data source

3. Execute SQL statements
Building JDBCCheckup.java

• First compile JdbcCheckup.java.
  
  javac JdbcCheckup.java

• Now we have JdbcCheckup.class in the current dir.

• Use java to run it as follows. We need to add the driver jar file to the classpath to give the program access to the driver software:

  java –classpath driver.jar;. JdbcCheckup   (Windows)
  (change ‘;’ to ‘:’ for UNIX/Linux)

• Driver.jar: ojdbc6.jar or mysql-connector-java-5.1.43-bin.jar
What is a jar file?

- From the [Java tutorial intro](https):
  - The Java™ Archive (JAR) file format enables you to bundle multiple files into a single archive file. Typically a JAR file contains the class files and auxiliary resources associated with applets and applications.
  - It is the standard way in Java to distribute a set of related classes
  - It is a zip-compressed directory (with subdirectories)
  - Example: the JDBC driver of a certain database vendor, like ojdbc6.jar.
  - The **jar** command lets us examine and build jar files:
    - “jar tf ojdbc6.jar” shows all the class names and other files in the Oracle driver software, a huge list with 1670 lines
    - “jar xf ojdbc6.jar” expands all the files into a filesystem tree
    - “jar cf myprog.jar .” packs up everything below the current directory into a jar file
Using a jar file to provide classes to a program (i.e. a library)

• Where does Java look for classes needed for a program?
  • Example: Java sees `Work.process("somearg")` and needs to find the class “Work”

• Let’s concentrate on Java without packages for this course.

• Simple program: Java just looks for it in the current directory “.”

• Bigger program: Java uses the classpath, a list of directories to search
  • OK, the simple program uses a default classpath of just “.”.
  • Java looks for the needed class in each directory of the classpath in turn

• The classpath can be specified on the java and/or javac command line, or by a CLASSPATH environment variable, or config in an IDE

• We’ll look at the command line method here.
Building JDBCCheckup.java, revisited

• First compile JdbcCheckup.java. No special classpath needs to be set up for this:
  javac JdbcCheckup.java
• Now we have JdbcCheckup.class in the current dir.
• Use java to run it. We need to add the driver jar file to the classpath:
  java -classpath driver.jar:. JdbcCheckup  (Windows)
  (change ‘;’ to ‘:’ for UNIX/Linux)
• This classpath has two directories, driver.jar and “.”.
• The whole jar file is treated like a directory: it is a compressed directory (with subdirectories). Java decompresses it on the fly.
• Driver.jar: ojdbc6.jar or mysql-connector-java-5.1.39-bin.jar
Running JdbcTemplate to Oracle from pe07

pe07$ java -classpath ojdbc6.jar:. JdbcTemplate
Please enter information to test connection to the database
Using Oracle (o), MySql (m) or HSQLDB (h)? o
user: xxxxxx
password: xxxxxx
use canned Oracle connection string (y/n): y
host: dbs3.cs.umb.edu
port (often 1521): 1521
sid (site id): dbs3
using connection string:
jdbc:oracle:thin:@localhost:1521:dbs3
Connecting to the database...connected.
Hello World!
Your JDBC installation is correct.
In JdbcCheckup.java

- In the program, we find out what database the user wants to connect to, and their username and password (for Oracle or mysql)
- For Oracle:
  - the server host is "dbs3.cs.umb.edu"
  - port is 1521
  - sid is "dbs3"
- These are used in the "connection string" or "database url" connStr = "jdbc:oracle:thin:@dbs3.cs.umb.edu:1521/dbs3". The code ends up with strings connStr, user, and password.
- Then get a connection from the driver:
  
  ```java
  Connection conn =
      Driver.Manager.getConnection(connStr, user, password);
  ```
Create a statement using the Connection object:

```java
Statement stmt = conn.createStatement();
```

Do DB actions using Statement—

```java
stmt.execute("drop table welcome");
stmt.execute("create table welcome(msg char(20))");
stmt.execute("insert into welcome values ('Hello World!')");
ResultSet rest =
    stmt.executeQuery("select * from welcome");
```

Display row, close connection (and its associated objects)
```java
conn.close() ← important to free up TCP/IP connection into the DB
```
JdbcCheckup.java: Exceptions

• The previous slide ignored Exceptions
• JDBC coding involves messy Exception handling
• We need to use finally as well as try and catch
• Need to review Exceptions
Java Exceptions

• We looked at this tutorial: http://www.tutorialspoint.com/java/java_exceptions.htm
• Another tutorial: Java Tutorial: Exceptions
• Skip coverage on The try-with-resources Statement
Analyze JdbcCheckup for Exceptions

```java
static void tryWelcomeExperiment(Connection conn) throws SQLException {
    // Create a statement
    Statement stmt = conn.createStatement();
    ResultSet rset = null;
    try {
        <various DB actions>
    } finally { // Note: try without catch
        stmt.close(); // clean up statement resources
    }
}
```

- This method throws any exception occurring during the DB actions
- That’s good: in Java, a method is expected to report problems via an exception
Special handling for drop table action

// We treat this drop table specially to allow it to fail
// as it will the very first time we run this program
try {
    stmt.execute("drop table welcome");
} catch (SQLException e) {
    // assume not there yet, so OK to continue
}

• You should always double-check any catch clause that does nothing. It’s a “code smell”.
• Remember that a method is supposed to throw for any failure. This is arguably not a failure.
• The other actions: create table, etc., throw from the method on any DB error.
Handling getConnection, etc.

- Note that conn.close() can throw, so need to put it in an inner finally to maintain control. Here’s a pattern:

```java
try {
    conn = DriverManager.getConnection(connStr, user, password);
    try {
        <do-db-stuff> // this can throw SQLException
    }
    finally {
        conn.close();
    }
} catch (SQLException e) {
    // throw to report problem, or if in main, can exit in error
}
```