CS348: Introduction to Database Management

JDBC Tutorial – Handout

Author: Stefan Büttcher
<sbuettch@uwaterloo.ca>

Modified by: Ahmed Ataullah
<aataulla@uwaterloo.ca>

University of Waterloo, January 2007

1 Compiling and Running Your Application

After you have obtained the DB2 JDBC driver from

http://www.student.cs.uwaterloo.ca/˜cs348/db2java.zip

you should copy this file to your working directory. DO NOT unzip this file. You can then compile Java programs containing JDBC-specific code with:

```
javac SQL.java
```

By default, java will look for all drivers and required (packed) files from your current directory and the program will compile successfully.
If `javac` was able to compile your program, you can execute it by typing:

```
java -classpath .:db2java.zip SQL
```

Here you should specify the class path since a lot of JRE's do not explicitly look for class libraries in your current working directory.

2 Using JDBC to Access the Database Server

In order to include the clases and interfaces of the JDBC API into your namespace, you have to include a line

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

into your program. If you do not do this, you will have to give a fully qualified class name (`java.sql.Connection`, `java.sql.Statement`, . . .) every time you refer to one of the JDBC classes.
Before your application can connect to the database server, the respective driver has to be registered with the `DriverManager` class, which is responsible to select the appropriate driver for a given connection URL. For the DB2 driver we are using for the assignment, you can do this with:

```
DriverManager.registerDriver(new COM.ibm.db2.jdbc.net.DB2Driver());
```

After the driver has been registered, your application can connect to the database server. In general, this is done by creating a `java.sql.Connection` object:

```
Connection conn = DriverManager.getConnection(url, user, password);
```

For the assignment, you have to use:

- `url = "jdbc:db2://rees.math.uwaterloo.ca:50448/cs448"`;
- `user = "db2guest"`;
- `password = "put password here"`.

By default, every DB connection represents one transaction. Using the `commit` and `rollback` methods, it is possible to divide a session into multiple transactions. Closing the connection finishes the current transaction. Transactions may not span over multiple connections.
After the connection has been established, you have to switch to the correct schema. This is done by the following sequence of commands:

```
Statement s = conn.createStatement();
s.executeUpdate("SET SCHEMA sbuettch");
s.close();
```

Switching to the correct schema is important. If you don't do this, your application will not be able to read data from the database. Please note that you do not have write permissions for any of the tables in the schema "sbuettch". Any attempt to change data inside the tables will cause a JDBC exception.

3 Using the JDBC API to Request Data

The most important classes in JDBC are `Statement`, `PreparedStatement`, and `ResultSet`.
Instances of the `Statement` class are created by

```
Statement s = conn.createStatement();
```

All resources associated with a `Statement` are released by

```
s.close();
```

The results to an SQL query are managed by an instance of the `ResultSet` class:

```
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("select * from class");
// do something with rs
rs.close();
s.close();
```

You can walk through a `ResultSet` object using the `next()` function. `next()` returns `true` as long as there is at least one more row available in the result set. Otherwise, it returns `false`. The contents of a row can be accessed by the `getString(...)` function (or `getInt`, or `getDate`, or ...). `getString` can either be called with an integer parameter or with a string parameter:

- `getString(1)` returns the string value of the first column of the current row.
- `getString("name")` returns the string value of the column with name ”name” inside the current row.

Example:

```
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("SELECT name AS n FROM class");
while (rs.next())
  System.out.println(rs.getString("n"));
rs.close();
s.close();
```

For every SQL statement being processed at the same time, you need a separate `Statement` object:

```
Statement s = conn.createStatement();
ResultSet rs1 = s.executeQuery("SELECT * FROM class");
ResultSet rs2 = s.executeQuery("SELECT * FROM course");
// do something with rs1
// do something with rs2
```

does not work, because `rs1` and `rs2` refer to the same object. Instead, you have to do:

```
Statement s1 = conn.createStatement();
ResultSet rs1 = s1.executeQuery("SELECT * FROM class");
Statement s2 = conn.createStatement();
ResultSet rs2 = s2.executeQuery("SELECT * FROM course");
// do something with rs1
// do something with rs2
```

If you have to send a sequence of very similar SQL statement, you can increase query processing performance by using the `PreparedStatement` class. An instance of `PreparedStatement` contains an SQL query with placeholders. Example:

```
PreparedStatement ps = conn.prepareStatement("SELECT x FROM y WHERE z=?");
for (int i = 1; i < 1024; i += i) {
  ps.setInt(1, i);
  ResultSet rs = ps.executeQuery();
  ...
  rs.close();
}
ps.close();
```

4 Exceptions

If you use any functions from the JDBC API, you have to take care of the exceptions they might throw. For most functions, `SQLException` is the only exception that can be thrown. In general, there are two ways to

deal with exceptions: catching them or passing them one level higher.
You can catch an exception using the usual `try...catch` construction:

```
try {
  ...
}
catch (SQLException sqle) {
  ...
}
```

If you don't want to explictly catch all possible exceptions, you can also pass them to the calling function by declaring your methods with the additional qualifier `throws SQLException`. If you do this for all functions in your class (including `main`), `javac` won't complain when it tries to compile your program, and any SQLException thrown inside the JDBC implementation will make your program stop immediately.
For the assignment, either way to deal with exceptions is fine.

5 Examples

There is a sample java program, (SQL.java) available through the course website, which should compile seamlessly and will let you run queries onto the db2 server in rees using an easy to use prompt. The purpose of this program is to make sure you can understand how JDBC works and give you a head start on your assignment by providing an interface as an example. All you need to run this program is to make sure that you fill in the connection parameters (mostly there except the password) and compile using the instructions in this guide.
As a first step make sure that you can compile this program on the unix servers. There is also another tutorial availble from the website which goes into details of JDBC and 'how it all works' if you are interested.

6 Submitting Your Assignment - A2, CS348 Winter 2007

As a first step make sure that you can compile your program on the cs enviornment. Electronic submissions are required along with written submissions - Your assignment will not be marked if you do not submit BOTH the written/printed and electronic versions.

We will be compiling and testing your assignments on rees so please make sure that your program runs and compiles perfectly according to specification. If your program fails to compile and run on the school enviornment you WILL be penalized. As usual commenting your code and proper presentation (indentation, meaningful variable naming etc.) will get you marks.

We will be looking for the following files in the electronic submission:

A2.java (the main program)
*.java (all other files)