JDBC Tutorial

Phil Smith
P.Smith.7@cs.bham.ac.uk

17th October 2012

Tutorial Outline

- Setting up a VPN
- Using the Java Database Connectivity (JDBC) API
- PostgreSQL JDBC Driver
- Requesting a connection
- JDBC URLs
- Statement Objects
- Statement Methods

Setting up a VPN

- The SoCS database servers cannot be accessed externally
- But, your program should run as though it were internal to SoCS
- *dbteach* is the School's database server
- *dbteach* can be accessed via VPN
- Supportweb have compiled setup guides for most OS's
  - https://supportweb.cs.bham.ac.uk/remote_access/vpn/

Using the JDBC API

- The JDBC API makes it possible to do three things:
  - Connect to a database or access any tabular data source
  - Send SQL statements
  - Process the results
- “JDBC hides the complexity of many data access tasks, doing most of the "heavy lifting"for the programmer behind the scenes.”[1]

[1]http://www.oracle.com/technetwork/java/overview-141217.html

PostgreSQL JDBC Driver

- Needed to programatically access your database!
- Download from here:
  - http://jdbc.postgresql.org/download.html
- Add to your classpath
- Add as a library to your IDE of choice:
- Netbeans / Eclipse?

Registering the Driver

- We need to load the JDBC driver before we can use it
- Implicitly load driver:
- Class.forName(“org.postgresql.Driver”);
- Pass parameters to JVM:
- java -Djdbc.drivers=org.postgresql.Driver ClassYouWantToRun
- This lets the JDBC know which particular driver to use:
  - MySQL, PostgreSQL ….

Requesting a Connection

- A database is represented by a URL (Uniform Resource Locator)
- In PostgreSQL, it takes one of 3 forms:
  - *jdbc:postgresql:**database***
  - *jdbc:postgresql:**//host/database***
  - *jdbc:postgresql:**//host:port/database***
- The **host** is the name of the server
  - Default is **localhost**
- The **port** is the portnumber that the server is listening on
- The **database** is, well... the database.

URL for *dbteach*

- Externally:
  - jdbc:postgresql://dbteach.cs.bham.ac.uk/**<db-name>**
- Internally:
  - jdbc:postgresql://dbteach/**<db-name>**

Requesting a Connection

- Use the JDBC to get *Connection* instance
- Use *DriverManager.getConnection();*

Connection db = DriverManager.getConnection(

url, username, password);

Issuing a Query

- Two main objects:
  - *Statement*
  - *PreparedStatement*
- Once these have been instantiated, a query can be issued
- This will return a *ResultSet* object
- A *Statement* can be used as many times as you want – but only one *ResultSet* may exist
- You should close these once finished

Simple Query - *Statement*

```
Statement st = db.createStatement();

ResultSet rs = st.executeQuery("SELECT * FROM books
    WHERE date = 2013");

while (rs.next()) {

    System.out.print("First column returned ");

    System.out.println(rs.getString(1));

}

rs.close();

st.close();
```

Simple Query – *PreparedStatement*

```
int year= 2000;

PreparedStatement st = db.prepareStatement("SELECT * FROM books
    WHERE date = ?");

st.setInt(1, year);

ResultSet rs = st.executeQuery();

while (rs.next()) {

    System.out.print("First column returned ");

    System.out.println(rs.getString(1));

}

rs.close();

st.close();
```

ResultSet

- Before reading any values, *next();* must be called
- A *ResultSet* should be closed once you have finished using it
- Once you make a new query with the existing *ResultSet*, the currently open *ResultSet* is lost

Performing Updates

- Use the *executeUpdate();* method to:
  - Insert
  - Delete
  - Update
- This doesn't return a *ResultSet*
- It returns an integer, representing the number of rows effected

Deleting Rows

```
int year = 500;

PreparedStatement st = db.prepareStatement("DELETE
    FROM books WHERE date = ?");

st.setInt(1, year);

int rowsDeleted = st.executeUpdate();

System.out.println(rowsDeleted + " rows deleted");

st.close();
```

Creating a table - Serial

- Want to automatically increment that ID?
- Use the Serial keyword in your statement String:

```
CREATE TABLE person (
    id SERIAL,
    name TEXT
);
```

Dropping a table

```
Statement st = db.createStatement();
st.execute("DROP TABLE books");
st.close();
```