

# Computer Science COMP2400/6240 2007

## Lab 6

### Programming with JDBC

October 16, 2007

## 1 Introduction

This lab follows on from the lecture on database programming, and gives you some skills working in JDBC. Not every student taking COMP2400/6240 is a programmer, so please don't panic if you aren't familiar with Java—we hope you will at least pick up the flavour of database programming.

*Please work in pairs!* Our experience is that people (even the best students) who work in pairs get more out of the labs than they do working by themselves. This will also help the tutors by allowing them to explain things 9 times rather than 18! If you aren't familiar with Java, please pair up with someone who is. Ask your tutor for assistance if necessary.

Some of the exercises below are optional. We recommend that you only attempt them *after* completing the mandatory parts of the lab.

This lab exercise involves modifying some sample applications. The sample code can be downloaded from <http://cs.anu.edu.au/student/comp2400/labs/lab6.zip>. Save this file in a directory on your local machine, such as `comp2400/lab6`, then do the following

```
$ cd ~/comp2400/lab6
$ unzip lab6.zip
```

This will create a directory called 'project'. We suggest you use Eclipse to work with these files, but feel free to work in the text editor of your choice.

### 1.1 Using Eclipse

1. Start Eclipse (K Menu>Development>Eclipse). Eclipse asks for a workspace (the 'Workspace Launcher' window)—just click 'OK'.
2. If this is the first time you have run eclipse, you will get a colourful screen with 4 or 5 icons. Click on the curly arrow at the top-right to go to the main Java view.
3. File > New > Project...
4. Click on Java Project (click next>).
5. Enter 'lab6' as the project name. Click on the radio button 'Create project from an existing source'.
6. Click 'Browse' and navigate to the project directory created above. This will be something like `comp2400/lab6/project`.
7. Click 'Finished'.

If you open the lab6 project (click on the triangle), you should see folder icons for 'src', 'data' and others. If you see 'project/src', you have imported one directory too high. Delete the project (without deleting its contents) and import it again.

## 1.2 Running JDBC programs

You can run your JDBC programs on your workstation without first logging in to the database server 'partch'. To simplify connecting to postgres, we will be connecting using a plain-text password. Before proceeding, please set your database password to something you won't mind people seeing, using the following SQL command (on 'partch'):

```
partch> psql
u1234567=> alter user u1234567 password 'abc123';
```

The project files include a Java class called Ping, which will simply connect to a database and indicate success. To run this class, (from the directory above the 'project' directory) the command is

```
$ java -cp project/bin:project/lib/pg_jdbc.jar ping.Ping
```

You also need to provide 4 parameters,

|          |                                                           |
|----------|-----------------------------------------------------------|
| host     | The name of the database server (partch)                  |
| db       | The name of the database (your user ID)                   |
| user     | Database user ID (your user ID)                           |
| password | Database password as set above (NOT your system password) |

The result should look like this

```
partch:~> java -cp project/bin:project/lib/pg_jdbc.jar ping.Ping
partch u3401953 u3401953 abc123
Loading postgres driver
Connecting to database //partch/u3401953
Connection to PostgreSQL 8.1.9 successful.
```

Take a look at the source code for this program. In Eclipse, go to the 'Package Explorer' on the left-hand side of the screen. Drill down on 'lab6', 'src', 'ping' and double-click on Ping.java to open it in the editor.

## 1.3 Resources

There are several resources available to help with JDBC. These include

- Lecture 28.
- The official Java documentation. <http://java.sun.com/j2se/1.5.0/docs/api/>
- El Masri and Navathe [1], Section 9.5 p279, or [2], Section 9.3, p315.

## 2 A simple upload program

The 'upload' package in the supplied files provides a simple program to upload data into the 'project' and 'works\_on' tables of the employee schema from lab 3. You might like to download the lab3 files again, and run the `drop_schema.sql` and `create_schema.sql` scripts to make sure the tables are there. Try running the upload program as follows:

```
partch:~> java -cp project/bin:project/lib/pg_jdbc.jar upload.UploadProject
partch u3401953 u3401953 abc123 project/data/project1.txt
Connecting to database //localhost/u3401953
Uploading project Uploaded Project, pnumber=9201, dno=1000
Uploading works_on eno=21286, pnumber=9201, hours=10.4
Uploading works_on eno=20765, pnumber=9201, hours=25.7
```

Note that if you repeat this command, it fails because of constraints in the database. Take a look at the sample data files in the directory `project/data`.

This program comprises 2 Java source files, `UploadProject.java`, which provides the main method, and parses the input file. The database interface code (which we are interested in) is in `DbInterface.java`—you don't need to look at how `UploadProject.java` works since this isn't a Java programming course.

## 2.1 Exercises

In `DbInterface.java`,

- (1) Modify the method `uploadProject` so that instead of failing when it can't insert into the database, it detects that the data exists and performs an update instead.  
*hint:* You should be able to achieve this by adding approx. 4 lines of code.  
*hint:* When this part is working, your program will still fail, just with a different error message indicating a constraint violation on the `works_on` table.
- (2) Modify the method `uploadHours` so that if the relevant tuple already exists, the hours from the input file are added to the existing hours.  
*hint:* How would you add 10 hours to a given `works_on` row if you were using `psql` ?
- (3) (*optional*) Modify the program so that each input file is uploaded as a single transaction. This will require modifying `UploadProject.java`.

## 3 An interactive program

You may find this section of the lab very time consuming. Don't worry if you can't complete it in the 2 hours of formal lab time.

The `editor` package in the example code is an interactive editor for employee details. Run the program using the following command line

```
partch:~> java -cp project/lib/pg_jdbc.jar:project/bin editor.Editor
```

and after filling in your connection details (the database url will be `jdbc:postgresql://partch/username`, you will see a window listing the employees in the database. Clicking on a row will bring up a dialog in which you can edit the details of that employee.

The database interactions for this editing dialog are performed in the class `EditEmployeeDb`. The `fetch` method retrieves data from the database and loads in into the user interface. The `update` method updates the database after the data has been edited. Currently the `update` method is empty, and the purpose of this exercise is to fill in this method. We will start very simply and work our way up.

- (4) Add code to update the first name field.
- (5) Add code to update the middle initial field, using a second SQL update statement. Note that we don't want to replace null values with non-null values.
- (6) Add code to update the department number field.
- (7) Add the remaining fields.

### 3.1 Concurrency

What will happen if two people try to update the same employee at the same time ? Read the description of the `Connection` interface on the Java API documentation, particularly the discussion of auto-commit mode.

- (8) Databases have been created for lab groups to share (they are named for the day and time, eg 'tue11'). Each member of the pair should connect to the shared database and experiment to see if you can produce an unexpected result from a concurrent update.
- (9) Modify your code to use explicit transactions to remove the update anomalies.
- (10) (*optional*) Even though we can solve the correctness issues using transactions, using multiple UPDATE statements when modifying a single tuple is inefficient. Rewrite the JDBC calls in the `update()` method so that it uses a single SQL statement.
- (11) (*optional*) Rewrite the `update()` method so that it only modifies fields that have been modified by the user.

## References

- [1] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Pearson, 4th edition, 2003.
- [2] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Pearson, 5th edition, 2006.

## A Example code

### A.1 JDBC Ping

```
package ping;

import java.sql.*;

/**
 * Postgres/JDBC 'ping'.  Connects to a database, and prints
 * out the software version of the server.
 *
 * @author Robin Garner
 * @date 11 Oct, 2007
 */
public class Ping {

    /**
     * Main method.  This program takes 4 arguments on the command line
     * host      - The host name of the database server
     * dbname    - Name of the database
     * username  - Postgres username for the connection
     * password  - Postgres password
     *
     * @param args Command-line arguments from the shell.
     */
    public static void main(String[] args) {
        if (args.length != 4) {
            System.err.println("Usage: Ping host db user pwd");
            System.exit(1);
        }
        String host = args[0];
        String database = args[1];
        String username = args[2];
        String password = args[3];

        try {
            /* Dynamically load the JDBC driver */
            System.out.println("Loading postgres driver");
            Class.forName("org.postgresql.Driver"); //load the driver

            /* Connect to the Postgres database */
            System.out.println("Connecting to database //" + host + "/" + database);
            Connection db = DriverManager.getConnection(
                "jdbc:postgresql://" + host + "/" + database,
                username,
                password); //connect to the db

            try {
                /* Ask the database for its metadata */
                DatabaseMetaData dbmd = db.getMetaData(); //get MetaData to confirm connection
                System.out.println("Connection to " + dbmd.getDatabaseProductName() + " " +
                    dbmd.getDatabaseProductVersion() + " successful.\n");
            } finally {
                /* Close the connection */
                db.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## B Running applications from Eclipse

Eclipse allows you to run Java programs directly, without having to switch to a terminal window. It does however require a little bit of setup. To run the `Ping` program from within eclipse, do the following:

- Locate the file `Ping.java` in the explorer window on the left-hand side of the screen.
- Right-click on this file.
- Choose `Run As > Run...`
- Highlight 'Java Application' in the list at the top left. Click 'New' at the bottom of the screen.
- Find the 'Arguments' tab. Enter the program arguments (database name, username etc).
- Click 'Apply'.
- Click 'Run'.

You should now be able to run this same configuration again by clicking the 'Run' button (looks like the 'play' symbol from a music player).

## C Setting the Eclipse JRE

Initially, Eclipse may start up referencing a Java 1.4 environment, and you will need to switch to the Java 1.5 environment. To do this,

1. From the 'Window' menu, choose 'Preferences...'
2. Expand the Java preferences section (click on the triangle).
3. Click on 'Installed JREs'.
4. Click 'Add' (on the right-hand side).
5. Enter a name for the JRE (eg 1.5).
6. In 'JRE home directory' enter `/usr/local/java`.
7. Click OK.
8. The 1.5 JRE should now be in your list. Click on the check-box next to the newly added one, to make it the default environment.
9. click 'OK'.
10. Eclipse may ask your permission to recompile all your classes - say 'yes'.

You should now find that the `lab6` project compiles correctly.