SCHOOL OF COMPUTER SCIENCE

PRIFYSGOL BANGOR UNIVERSITY

# Java Technologies Mini-Project 2

## Java Database Connectivity

**Dave Perkins**

## Introduction

In this laboratory session we explore the topic of Java database connectivity, i.e. how can we get a Java application to 'speak' to a database. More particularly, by working your way systematically through these laboratory notes you will learn how to:

- connect a Java application to a database;
- use a Properties file to maintain configuration information;
- maintain a Properties file through the use of Maven;
- generate and populate a multi-table database;
- build a console style menu system to access the database;
- maintain the separation of interface and core functionality.

## Accessing the MySQL Server

MySQL is a well known, open source DBMS which runs as a server providing multi-user access to one or more databases. Although this software is open source it is nonetheless the property of the Oracle Corporation who supply paid editions for commercial use. To obtain a free copy of the Community Edition visit the site below and download and install *MySQL Community Server* (5.6.13).

http://dev.mysql.com/downloads/

Although working your way through the installation process is worthwhile there is no need to do so because a MySQL server is already installed on the university network. The *hostname* for the server is **mysql.cs.bangor.ac.uk** whilst the *port number* is 3306.

To access the server you must have an account with appropriate privileges. To obtain your *password* and *user name* contact the system administrator, currently Cameron Gray(c.gray@bangor.ac.uk).

You should also note that MySQL is not the only advanced open source database; have a look for example at Postgres SQL or Microsoft's Server Express.

Before reading these notes any further you are strongly advised to work your way through Chapter 21 in *Big Java* (4th edition).

## University of Utopia Database

The database used in this mini-project consists of five tables which hold data relating to the staff, students and modules of the University of Utopia. The relations which constitute this database are listed below:

```
student(student_Id,student_name,degree_scheme)

Primary Key student_Id


staff(staff_Id,staff_name,staff_grade)

Primary Key staff_Id


module(module_Id, module_name, credits)

Primary Key module_Id


registered(student_Id, module_Id)

Primary Key student_Id, module_Id

Foreign Key student_Id references Student(student_Id)

Foreign Key module_Id references Module(module_Id)


teaches(staff_Id, module_Id)

Primary Key staff_Id, module_Id

Foreign Key staff_Id references Staff(staff_Id)

Foreign Key module_Id references Module(module_Id)
```

Pay particular attention to the key structure of this database. For more information concerning the integrity rules for the database see next section of this document.

## SAMPLE DATA

To assist you in this task some data has been provided. Please make sure that when you create the University of Utopia database the tables initially contain the data below. If we all standardise on the initial state of the database the process of testing and marking will be so much easier. For purposes of this module it is assumed that you have a working knowledge of SQL and in particular understand the following concepts:

- Primary Keys
- Foreign Keys
- Use of the **CASCADE** feature to enforce integrity

You will be expected to use SQL commands to mark attributes as either primary or foreign keys and to specify *cascades* where appropriate. For example, if a student is deleted from the **Student** table then any reference to that student in another table(e.g. **Registered**) must also be deleted.

**student**

| student_id | student_name | degree_scheme |
|---|---|---|
| S10345 | John Smith | BSc Computer Science |
| S10346 | Sian Evans | BSc Computer Science |
| S10347 | Sean Crossan | BSc Electronic Engineering |
| S10348 | Jamie McDonald | BSc Mathematics |

**module**

| module_id | module_name | credits |
|---|---|---|
| CS101 | Introduction to Computing | 10 |
| CS203 | Data Structures and Algorithms | 10 |
| CS204 | Computer Architecture | 10 |
| M101 | Foundation Mathematics | 20 |

**staff**

| staff_id | staff_name | grade |
|---|---|---|
| E10010 | Alan Turing | Senior Lecturer |
| E10011 | Tony Hoare | Reader |
| E10012 | Seymour Cray | Lecturer |

**registered**

| student_id | module_id |
|---|---|
| S10345 | CS101 |
| S10346 | CS203 |
| S10346 | CS204 |
| S10347 | CS204 |
| S10348 | M101 |
| S10348 | CS101 |

**teaches**

| staff_id | module_id |
|---|---|
| E10010 | CS101 |
| E10011 | CS203 |
| E10012 | CS204 |
| E10010 | CS204 |
| E10011 | M101 |
| E10011 | CS101 |

When you are asked to demonstrate your program please ensure that *all* of the above data has been loaded into the appropriate tables.

## Exercise 1: Making a Connection (10%)

Create a new Maeven project for this week's laboratory and install the official JDBC driver for MySQL as a dependency. Once this has been done your Maven project should look like this:

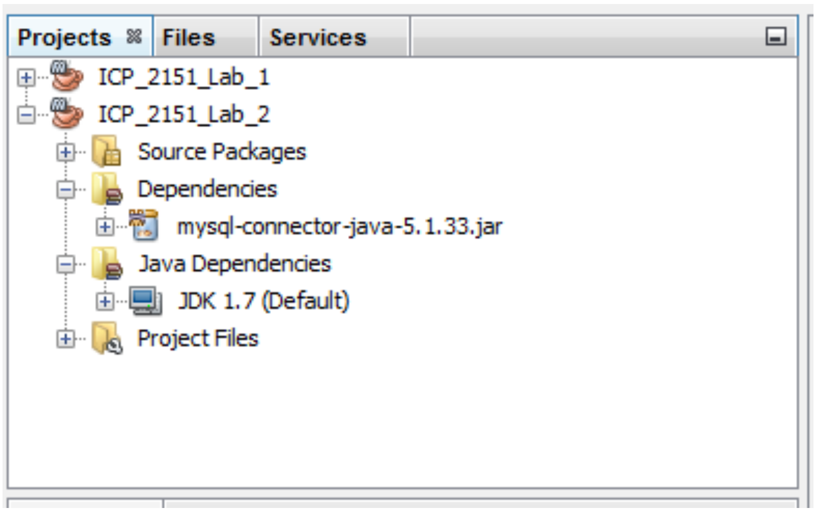


Figure 1JDBC DRiver Installed

Also use work from the previous week to ensure that you have the following files in place:

- **SimpleDataSource.java**
- **MySqLTester.java**
- **database.properties**

In addition, you will need to re-use the profiles section from the POM file of last week's project.

The test program should attempt to connect to the SQL server and then create a table containing two or three rows of data. The contents of the table should first be displayed and then the tables should be dropped.

Remember that If you are using the remote server provided by the School of Computer Science you will need an account with a login and password. If you have not been supplied with your account details speak to the lecturer or laboratory demonstrator.

## Exercise 2: Creating and Populating the Database (20%)

This exercise involves the use of SQL scripts to create the following tables:

- `student`
- `staff`
- `module`
- `teaches`
- `registered`

The sample data provided in the previous section *must* be used to populate the tables above. To do this you should follow the instructions below.

For every build of your project, you will be using the SQL Maven Plugin to run all of the scripts (instead of needing to write an independent Java class). To *include* this plugin in your project you *must* add the fragment of XML below to your POM. (This section does not need any alterations.)

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
     <artifactId>sql-maven-plugin</artifactId>
     <version>1.5</version>
    <configuration>
       <driver>${jdbc.driver}</driver>
       <url>${jdbc.url}</url>
       <username>${jdbc.username}</username>
       <password>${jdbc.password}</password>
    </configuration>
     <executions>
       <execution>
         <phase>generate-test-resources</phase>
         <goals>
           <goal>execute</goal>
         </goals>
         <configuration>
           <srcFiles>
             <srcFile>src/test/resources/1-destroy.sql</srcFile>
             <srcFile>src/test/resources/2-create.sql</srcFile>
             <srcFile>src/test/resources/3-insert.sql</srcFile>
           </srcFiles>
         </configuration>
       </execution>
     </executions>
</plugin>
```

This snippet must be included in the `project` > `build` > **`plugins`** element. You may need to create some of these elements if they do not already exist. More detailed documentation on this plugin can be found at http://mojo.codehaus.org/sql-maven-plugin/ should you wish to read more.

Having amended the POM file, you should create the three SQL scripts referenced in the **`<srcFiles>`** element. Each script performs a specific task; do not mix them up or include things that should be in another script. The files must be named *exactly* as they appear here and must be created in the directory **`src/test/resources`** (you may need to create this structure yourself).

| | |
|---|---|
| **1-destroy.sql** | This script is intended to *guarantee the initial state* of the database. To develop this script, you will need to undo all of the changes effected by the other two scripts. This means that for each proposed table there will be a conditional drop statement, as well as for any columns added to existing tables. In effect you are being asked to produce a script that "undoes" each and every change that the **2-create.sql script** will make to the schema. |
| **2-create.sql** | This script creates tables in the database and/or makes the structural changes to some existing tables. The script must not include any DML (INSERT, UPDATE or DELETE statements). You should assume that none of the creations or modifications have been made prior to running this script. |
| **3-insert.sql** | This script handles all the DML changes necessary before the application will run. This may include adding sample (also known as 'seed') data. You can assume that all necessary structural changes have been made at this stage. |

All three of the above scripts must be able to execute without any errors. Your build will fail if there are any errors in running these scripts. This means that the scripts are guaranteed to have executed on the database prior to your first test case being run. As a consequence, programmers can develop unit tests covering the full range of database operations.

Make sure that when creating the tables *all relevant integrity rules* are embodied in the design of the table. See explanatory notes relating to the sample database.

## Exercise 3: Building a Command Line Interface (20%)

Now that the Utopia University database has been created, develop a menu driven program to provide the following options:

- List all students;
- List all staff;
- List all modules;
- List all module registrations;
- List all courses on which staff teach;
- Quit application.

In addition, you should provide options to:

- Add a new student;
- Delete an existing student;
- Update existing student details;

The interface should take the form of a command line interface(CLI) as illustrated in **Appendix 1**.

In completing this part of the project pay particular attention to the following points:

- Your program should consist of three classes
  - `DatabaseMain`
  - `DatabaseTextInterface`
  - `DatabaseManager`
- The function of `DatabaseMain` is to provide a main method to serve as an entry point to the application as a whole.
- The function of `DatabaseTextInterface` is to generate menu screens, read input values (e.g. the identity number of a student who is to be deleted) and display output (e.g. table contents); this class does not directly access the database but instead uses methods provided by the class `DatabaseManager`.
- The function of the class `DatabaseManager` is to respond to requests, received via the interface, for the execution of specifc SQL commands. For example, if the user wishes to delete a specific student this request is forwarded to an appropriate method in the manager. See Appendix 2 for further guidance.

To ensure the strict separation of the interface from the core functionality of the application (i.e. the database operations) you are not permitted to use either a `Scanner` or print statements of any form (e.g. `print` or `println`) in the `DatabaseManager` class.

## Exercise 4: Full CRUD Functionality (20%)

As the program stands there is only support for basic CRUD (Create, Read, Update, Delete) operations[1] on the **student** table. However, it is relatively straightforward to provide functionality for adding, deleting and updating entries in the other tables by supplying extra methods in the **DatabaseManager** class. For example, to enable the program user to add a module to the **module** table, supply a method with the header below:

```
public void addModule(String id, String name, int credits)
                 throws SQLException
```

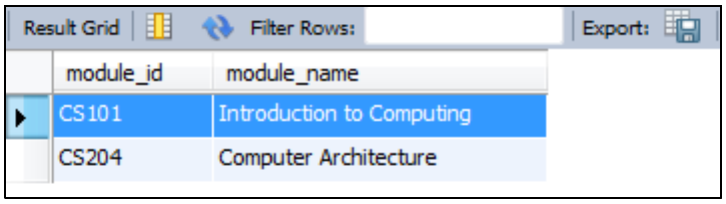
The problem with this approach is that whilst it will work it involves generating twelve more methods, three for each table. So here is a challenge; can you think of a more economic and elegant solution?

## Exercise 5: Complex Queries (30%)

In this exercise you are required to execute a set of queries involving the use of some more advanced features of SQL. For example, queries which involve the contents of more than one table will make use of some form of **join** operation. To complete this exercise, you should first develop SQL scripts to obtain the following information:

- all modules (module_Id and module_title) taught by a specified member of staff
- all students (student_Id and student name) registered on a specific module
- all staff (staff_Id, staff name, module_Id) who teach modules on which a specific student is registered
- all staff (staff_Id, staff_name) who teach on more than one module

It is suggested that *before* you embed the statements in your Java application you test your SQL scripts using an SQL tab in Workbench. For example, the script which lists the modules taught by a specific member of staff, in this case Alan Turing, was run and generated the following results.



| module_id | module_name |
|---|---|
| CS101 | Introduction to Computing |
| CS204 | Computer Architecture |

Figure 2 Running a Script File in Workbench

[1] For an explanation of this term see http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

If you are having difficulties with this question I suggest you try the following approach:

- Identify all tables required to formulate the query e.g. `TableA, TableB, TableC`

- Generate the Cartesian product using
```
SELECT * FROM TableA, TableB, TableC
```

- Next identify the conditions in a **`WHERE`** clause to extract precisley the rows required
```
SELECT *
FROM TableA, TableB, TableC
WHERE ...
```

- Finally, restrict the columns generated to those required using
```
SELECT TableA.Colm, TableB.Coln, TableC.Colo
FROM TableA, TableB, TableC
WHERE ...
```

Having got this far you it should be straightforward to produce queries based on the use of SQL JOIN and which generate the same result table as that delivered using the Cartesian Product.

Once these queries are working, use them to add extra functionality to your data base application. To do this, create a new option on your menu called **Reports** (see Appendix 2).

When the user selects this option, a submenu should be displayed with options corresponding to the above queries. The idea is illustrated below:

```
Reports
*******
1. Modules taught by
2. Students registered on
3. Staff who teach student
4. Staff who teach more than
```

Your report generator should be as generalised as possible thus if the user selects option 1 the application should generate a prompt asking the user to specify the *name of the member of staff* in whom they are interested. The application will then behave as illustrated below:

```
Member of staff :> Alan Turing

Module ID  Name
*********  *******************
CS101      Intro to Computing
CS204      Computer Architecture
```

The above illustrates output for Alan Turing but the report generator should work for whatever name is supplied. In fact, the first three reports, and not just the first one, should support generality. If in doubt about what this means ask your lecturer.

## Submission

Use **Blackboard** to submit your source code files. Each source code file must

- Contain a program header
- An appropriate level of Javadoc style comments
- Follow a consistent style of indentation
- Follow the usual Java conventions for class and variable names

Failure to comply with the above will result in a marks penalty of 5%.

The deadline for course work submission is published on Blackboard. Late submissions will be penalised in line with School policy.

When submitting work, it is your responsibility to ensure that all work submitted is

- consistent with stated requirements
- entirely your own work
- on time

Please note that there are **severe penalties** for submitting work which is not your own. *If you have used code which you have found on the Internet* or *from any other source,* then you **must** signal that fact with appropriate program comments. Note also that to obtain a mark you **must** attend a lab session and be prepared to demonstrate your program and to answer questions about the coding. Non-attendance at labs will result in your work not being marked.

**Dave Perkins**

## Useful Links

The last time I checked these notes all of the links below were correct and led to a functioning site. If a link fails please let me know and I will amend the laboratory notes.

- Oracle: Java™ Tutorials – JDBC(TM) Database Access - http://docs.oracle.com/javase/tutorial/jdbc/index.html
- Wikipedia - "Java Database Connectivity" – http://en.wikipedia.org/wiki/JDBC
- The Java Developers Almanac 1.4 – "Getting Data From A ResultSet" - http://www.exampledepot.com/egs/java.sql/GetRsData.html
- O'Reilly ONJava.com - "JDBC 4.0 Enhancements in Java SE 6" by SriniPenchikala – http://www.onjava.com/pub/a/onjava/2006/08/02/jjdbc-4-enhancements-in-java-se-6.html
- Developer.com - "Using JDBC with MySQL, Getting Started" – http://www.developer.com/java/data/article.php/3417381
- MySQL.com - "Using MySQL with Java" – http://dev.mysql.com/usingmysql/java/

Check out the last two entries for some useful information about MySQL. For Reference Manuals for MySQL see

http://dev.mysql.com/doc/

For a SQL syntax reference source try:

http://dev.mysql.com/doc/refman/5.0/en/sql-syntax.html

Finally, if you find any really good MySQL sites do let me know, and I will add them to the list.

# Appendix 1:

The screen shots below illustrate the operation of the interface. In this example the user of the program decides to select option 3 to obtain a listing of all modules

```
Main Menu
********************
1. List student
2. List staff
3. List modules
4. List registrations
5. List courses taught by staff
6. Add student
7. Delete student
8. Update student
9: Reports
0. Quit


:> 3
```

The system responds by displaying the contents of the tables **Modules**

```
Module ID  Name                            Credits
*********  ***************************     *******
CS101      Intro to Computing              10
CS203      Data Structures & Algorithms    10
CS204      Computer Architecture           10
M101       Maths I                         20
```

An alternative, and probably, better approach to having a single main menu with a large number of options is to have a smaller main menu providing access *to sub-menus*.

```
Main Menu
********************
1. Students
2. Modules
3. Registrations
4. Reports
0. Quit
:> 1

Sub-Menu (Students)
*******************
1. Add student
2. Remove student
3. Update student
4. List students
0. Return to main menu
```

# Appendix 2: Datbase Manager

To help you I have included a fragment from my version of the class D**atabaseManager**

```
public class DatabaseManager
{

    public DatabaseManager()
    {
        Intialise data source using properties file
    }

    public void addStudent(String id, String name, String degree)
            throws SQLException
    {
        Connection conn = SimpleDataSource.getConnection();
        try
        {
            // Create string with variable parts.
            String command = "INSERT INTO student VALUES(?,?,?)";
        }
        …
   }

   ...
}
```