Lab Session 4 - Building Database-Driven Applications

By Mitchell Welch

University of New England

Reading

- Chapter 10 from ***Fundamentals of Database Systems*** by Elmazri and Navathe

---

Summary

- Introduction
- House Keeping
- Java Revision
- Connecting up JDBC
- Some Exercises for You
- Using JDBC to Create Records

Introduction

- Todays practical session is all about connecting java applications up to our database (hosted using the PostgreSQL DBMS)to build database-driven applications.
- A *Database* driven application is basically any program uses a centralised database to manage the storage of any form of persistent data.
- This is achieved by connecting your java application up to the database on the DBMS and executing SQL statements to retrieve and manipulate the data within.

House Keeping

- In order to connect your application up to the database, you will need to specify the username and password of the user you want to connect to the database as.
- Up until this point in the course, you have been connecting to the DBMS using the `psql` client, which automatically connects you up using your UNE username and password.
- To avoid having to specify your UNE username and password in plain-text within your java programs, we have created a new user account in the PosgreSQL sever on turing for each student dedicated for use in your applications.
    - The username for your apps account with by in form: `<your_une_username>_apps`
    - The password for the account will be your **UNE Student Number**

- For example, if my UNE username was *student1* and my student number was *122334455*, my apps account username will by **'student1_apps'** and the password for the account will be **'122334455'**
- You can create databases and log into your apps account using the `createdb` application and 'psql' client, however you will need to explicitly specify the username to stop the client from automatically logging in using you UNE account credentials.
- This is done using the `-U`, `-W` `-h` options:
  - `-U` Specifies the account username that you will be logging in with
  - `-W` Forces the `psql` client to prompt for the password
  - `-h` Sets the host - `127.0.0.1` is turing.une.edu.au
- For todays practical session, you will need to create a new database using you apps account called `<your_username>_apps_prac_5`
- Like this (Sub in your username and enter your student number as the password when prompted)

```
[mwelch8@turing ~]$ createdb -U mwelch8_apps -W -h 127.0.0.1 mwelch8_apps_prac_5
Password:
[mwelch8@turing ~]$ psql -U mwelch8_apps -W -h 127.0.0.1 mwelch8_apps_prac_5
Password for user mwelch8_apps:
psql (9.5.4)
Type "help" for help.

mwelch8_apps_prac_5=⇒ \dt
No relations found.
mwelch8_apps_prac_5=⇒
```

- Once you have successfully logged into to your newly created prac_05 database using your apps account, you will need to build your COMPANY database.
- We can do this by importing a pg_dump of practical ones database, we will call this prac_01.sql.
- Before we can do this, we need to change the owners name to <your_username>_apps, as this is your apps account which is a different account to your standard psql account. This is required to access our database from our application.
- The easiest way to do this is to use a text editor such as gedit and use the find and replace command to replace all instances of your username. If you miss this step you can use dropdb with the syntax below to drop the db and start again.

```
[mwelch8@turing ~]$ createdb -U mwelch8_apps -W -h 127.0.0.1 mwelch8_apps_prac_5
[mwelch8@turing ~]$ psql -U mwelch8_apps -W -h 127.0.0.1 mwelch8_apps_prac_5
psql (9.4.4)
Type "help" for help.

mwelch8_apps_prac_5=⇒ \i ~/prac_01.sql

 ...

mwelch8_apps_prac_5=⇒ \dt
             List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | department      | table | mwelch8
 public | dependent       | table | mwelch8
 public | dept_locations  | table | mwelch8
 public | employee        | table | mwelch8
 public | project         | table | mwelch8
 public | works_on        | table | mwelch8
(6 rows)
```

- **NOTE** Running the sql script in this way may produce some errors when you run this script. These errors are from the commands that attempt to change the owner of the relations to account under which they were originally created.
- These can be ignored as we want the owner of these relations to default to the current account.
- Now you have your database for todays practical session set up.

Java Revision

- As quick revision exercise we will review, compile and run a couple of simple Java programs just to get everyone up to speed.
- The first program will look at is the ubiquitous `Hello World` exmaple:

```
/*************************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 ************************************************************************/

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }

}
```

- Recall that compiling a java program is done using the `javac` appliction and we can run our application by invoking the Java Virtual Machine using the `java` application:

```
21641:prac_5 mwelch8$ javac HelloWorld.java
21641:prac_5 mwelch8$ ls
HelloWorld.class    HelloWorld.java     prac_5.md
21641:prac_5 mwelch8$ java HelloWorld
Hello, World
21641:prac_5 mwelch8$
```

- For processing user input from the keyboard we can use the `InputStreamReader` and `BufferedReader` classes.
- The static functions `parseInt` on the `Integer` class has bee used to convert the String-type input into the `int` types required.

```
import java.io.* ;


class InputExample {
     public static void main(String args[])
     {
          // Create a new InputStreamReader and connecting to STDIN
          InputStreamReader istream = new InputStreamReader(System.in) ;

          // Create a new BufferedReader and connect it to the InputStreamReader
          // Now the plumbing is done, we can read from the BufferedReader
          BufferedReader bufRead = new BufferedReader(istream) ;

          /*********************************************************
          * The BufferedReader class a several methods for reading
```

```
            * input:
            *  readLine() - reads a line of input from the stream
            *  read() -     returns the integer representation of the
            *               next character in the stream.
            *  read(char[] cbuf,int off,int len)-
            *               Reads len characters to the buf.
            *
            ********************************************************/


            System.out.println("Welcome To My First Java Program");

            try {
                 System.out.println("Please Enter In Your First Name: ");
                 String firstName = bufRead.readLine();

                 System.out.println("Please Enter In The Year You Were Born: ");
                 String bornYear = bufRead.readLine();

                 System.out.println("Please Enter In The Current Year: ");
                 String thisYear = bufRead.readLine();

                 int bYear = Integer.parseInt(bornYear);
                 int tYear = Integer.parseInt(thisYear);

                 int age = tYear-bYear;
                 System.out.println("Hello " + firstName + " You are "
                     + age + " years old");
            }
            catch (IOException err) {
                 System.out.println("Error reading line");
            }
            catch(NumberFormatException err) {
                 System.out.println(err);
            }
       }
  }
```

- This code was adapted from a tutorial available at:http://www.codeproject.com/Articles/2853/Java-Basics-Input-and-Output
- Compile and run these examples and the code.

Connecting up JDBC

- Now we are ready to write a Java program that connects to our database.
- First we need to download the PostgreSQL JDBC driver. This available here: http://turing.une.edu.au/~cosc210/workshops/prac_5/postgresql-42.6.0.jar
- Download this to your current working directory and extract the pre-compiled .class files (and the file structure):

```
[cosc210@turing prac_5]$ jar -xf postgresql-42.6.0.jar
[cosc210@turing prac_5]$ ls
HelloWorld.java  META-INF  org  postgresql-42.6.0.jar
[cosc210@turing prac_5]$
```

- This will create two directories (META-INF and org). Extracting the files in this way will make them accessible to you java programs at runtime.
- Now you are ready to run your first database connected Java program.
- The first program that we will look simply connects up to your COMPANY database and lists the first and last names of all employees:

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

```
import java.util.*;

public class  DbTester{

    public static void main(String [] argv) throws Exception{

        Connection conn = null;
        try
        {

          Class.forName("org.postgresql.Driver");
          String url = "jdbc:postgresql://localhost/<une_username>_apps_prac_5";
          conn = DriverManager.getConnection(url,"<une_username>_apps", "<une_student_number>");

        }
        catch (ClassNotFoundException e)
        {
          e.printStackTrace();
          System.exit(1);
        }
        catch (SQLException e)
        {
          e.printStackTrace();
          System.exit(2);
        }
        //Now we're connected up lets retrieve a list of employees


        System.out.println("\n****Employees Currently Within the Database*****");
        System.out.println();
        //Print the column headings to the console
       System.out.printf("%-12s %-12s %n", "First Name","Last Name");
        System.out.println("————————————————————");

        // First we specify our query
        String query = "SELECT fname, lname FROM employee;";
        Statement stmt = null;
        try {
        //Create an sql statement object
        stmt = conn.createStatement();
        //Execute the query
        ResultSet rs = stmt.executeQuery(query);
        //Iterate through the results and print to the console
        while (rs.next()) {
            String fName = rs.getString("fname");
            String lName = rs.getString("lname");
            System.out.printf("%-12s %-12s %n", fName,lName);
        }
        } catch (SQLException e ) {
        System.out.println(e);
        conn.close();
        System.exit(1);
        }
        System.out.println("————————————————————");
        System.out.println("\nQuery Executed Successfully … exiting");
        //Close the database connection
        conn.close();
      }

}
```

- This program can be downloaded from http://turing.une.edu.au/~cosc210/workshops/prac_5/DbTester.java
- Before you can run this program, you will need to enter the details in for the database URL and for the database connection in the lines shown below:

```
String url = "jdbc:postgresql://localhost/<une_username>_apps_prac_5";
conn = DriverManager.getConnection(url,"<une_username>_apps", "<une_student_number>");
```

- The url will need to list the database that you are attempting to connect to - in this situation it will be the prac_5 database that you created earlier in the prac.
- In the call to the `getConnection( … )`, you will need to specify the name of your apps account (which will be in the form "\<une_username>_apps") and the password for this account (which we have set to be your UNE student number).
- Once you have entered these items, compile and run the program. If you have correctly set your database up and updated the details in the source-code, the output should look something like this:

```
[mwelch8@turing prac_5]$ javac DbTester.java
[mwelch8@turing prac_5]$ java DbTester

****Employees Currently Within the Database*****

First Name   Last Name
————————————————————
Alex         Freed
Bob          Bender
Evan         Wallis
James        Borg
Jared        James
John         James
Kim          Grace
Ahmad        Jabbar
Alicia       Zelaya
Franklin     Wong
Jennifer     Wallace
Red          Bacher
Sammy        Hall
Carl         Reedy
Naveen       Drew
Ray          King
Billie       King
Jon          Kramer
Arnold       Head
Gerald       Small
Helga        Pataki
Lyle         Leslie
Jill         Jarvis
Kate         King
Nandita      Ball
Alec         Best
Bonnie       Bays
Sam          Snedden
John         Smith
Joyce        English
Ramesh       Narayan
Jeff         Chase
Chris        Carter
Jenny        Vos
Andy         Vile
Josh         Zell
Tom          Brand
Brad         Knight
Jon          Jones
Justin       Mark
————————————————————

Query Executed Successfully … exiting
[mwelch8@turing prac_5]$
```

- Now lets walk though our example:
- The first section sets the database driver and sets up the connection details:

```
Connection conn = null;
Class.forName("org.postgresql.Driver");
String url = "jdbc:postgresql://localhost/<une_username>_apps_prac_5";
```

```
conn = DriverManager.getConnection(url,"<une_username>_apps",
```

- The `Class.forName` call loads the `Driver` class at runtime
- The call to the static function `getConnection( … )` on the static `DriverManager` object creates the connection to our database and stores the connection information in the `Connection` object `conn`.
- We then specify the SQL query we want to execute on the database and store it as a string. In this example, we are simply selecting the `fname` and `lname` columns for all employees.

```
// First we specify our query
String query = "SELECT fname, lname FROM employee;";
```

- We then package our query into a `Statement` object and execute the query:

```
stmt = conn.createStatement();
//Execute the query
ResultSet rs = stmt.executeQuery(query);
```

- The results returned from the DBMS are stored in a `ResultSet` object, that we can iterate through using the `next()` method. The `ResultSet` object :

```
while (rs.next()) {
            String fName = rs.getString("fname");
            String lName = rs.getString("lname");
            System.out.printf("%-12s %-12s %n", fName,lName);
        }
```

- The `getString( … )` methods can be used to return the value for an individual column. You original query must return the column of data in order for you to access it! (e.g. Our query only returns the `fname` and `lname` so they are the only columns we will be able to access)
- The `System.out.printf( … )` call prints the `fname` and `lname` variables to STDOUT.
  - This also includes a formatting string `"%-12s %-12s %n"`. Review http://www.homeandlearn.co.uk/java/java_formatted_strings.html to see how this works.

Exercises for You

1. Modify the `DbTester` program so that it displayes the `fname`, `lname`, `bdate`, `sex` and salary from the employee table in a formatted list with columns that are 12 characters wide with appropriate column headings.
   - To achieve this you will need to modify the query your program is executing to return the columns required.
   - You will need to add calls to the `getString( … )` function to retrieve the appropriate columns for each row of data returned.
2. Modify the `DbTester` program so that it displays the department name (`dname`)and department location (`dlocation`) in addition to the attributes displayed in question 1.

Using JDBC to Create Records

- Now that we have Java program connecting up and bringing data down form our database, we can look at developing a java program that modifies some of you data.
- First we will look at a simple example that creates a new record in the employee table.

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

public class  DbInsert{
```

```
      public static void main(String [] argv) throws Exception{

          Connection conn = null;
          try
          {
            Class.forName("org.postgresql.Driver");

              String url = "jdbc:postgresql://localhost/<une_username>_apps_prac_5";

            conn = DriverManager.getConnection(url,"<une_username>_apps", "<une_student_number>");
          }
          catch (ClassNotFoundException e)
          {
            e.printStackTrace();
            System.exit(1);
          }
          catch (SQLException e)
          {
            e.printStackTrace();
            System.exit(2);
          }
          //Now we're connected up lets retrieve a list of employees


          System.out.println("\n****Inserting a New Employee*****");
          System.out.println();


          // First we specify our query
          Statement stmt = null;
          try {
              //Create a new statement object - notice the additional arguments for inserting
              stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
              //Get all record in the employee table
              ResultSet uprs = stmt.executeQuery("SELECT * FROM employee");
              /* - Employee table. Here to show column list
                  CREATE TABLE employee (
                      fname character varying(15) NOT NULL,
                      minit character varying(1),
                      lname character varying(15) NOT NULL,
                      ssn character(9) NOT NULL,
                      bdate date,
                      address character varying(50),
                      sex character(1),
                      salary numeric(10,2),
                      superssn character(9),
                      dno integer
                  );

              */
              //Create a new row in the ResultSet object
              uprs.moveToInsertRow();
              //Add new employee's information to the new row of data
              uprs.updateString("fname", "New_fname");
              uprs.updateString("minit", "S");
              uprs.updateString("lname", "New_lname");
              uprs.updateString("ssn", "112233445");
              uprs.updateInt("dno", 5);
              //Insert the new row of data to the database
              uprs.insertRow();
              //Move the cursor back to the start of the ResultSet object
              uprs.beforeFirst();
          } catch (SQLException e ) {
          System.out.println(e);
          conn.close();
          System.exit(1);
          }

          System.out.println("\nQuery Executed Successfully ... exiting");
          //Close the database connection
          conn.close();
        }
```

```
}
```

- After running this program on your apps database:

```
mwelch8_apps_prac_5⟹ select fname,lname from employee;

...

 Jon       | Jones
 Justin    | Mark
 New_fname | New_lname
(41 rows)

mwelch8_apps_prac_5⟹
```

- This program connects up to our database using the same process as in the first example.
- The new code sits within the central `try{ … }` block:

```
//Create a new row in the ResultSet object
 uprs.moveToInsertRow();
 //Add new employee's information to the new row of data
 uprs.updateString("fname", "New_fname");
 uprs.updateString("minit", "S");
 uprs.updateString("lname", "New_lname");
 uprs.updateString("ssn", "112233445");
 uprs.updateInt("dno", 5);
 //Insert the new row of data to the database
 uprs.insertRow();
```

- Here we create a new row in the result set that has been returned from the `SELECT` query.
- We then use the `updateString` and `updateInt` members to add the data for each attribute to the new row of data.
- Finally, we call the `insertRow` member to copy the data back to the database.
- What happens when you run this program a second time? (Connected to the same database)
  - Hint: Can you insert records into a database with duplicate primary keys?

Exercises for You

1. Update the following program to insert the employee information entered through the console session.

```
import java.io.* ;


class CreateEmployee {


  public static void main(String args[]){

    // Database connection stuff as per the examples

    Connection conn = null;
    try
    {
      Class.forName("org.postgresql.Driver");
      String url = "jdbc:postgresql://localhost/<une_username>_apps_prac_5";
      conn = DriverManager.getConnection(url,"<une_username>_apps", "<une_student_number>");
    }
    catch (ClassNotFoundException e)
    {
```

```
        e.printStackTrace();
        System.exit(1);
      }
      catch (SQLException e)
      {
        e.printStackTrace();
        System.exit(2);
      }

      /* - Employee table. Here to show column list
      CREATE TABLE employee (
      fname character varying(15) NOT NULL,
      minit character varying(1),
      lname character varying(15) NOT NULL,
      ssn character(9) NOT NULL,
      bdate date,
      address character varying(50),
      sex character(1),
      salary numeric(10,2),
      superssn character(9),
      dno integer
      );

      */

      try {
        System.out.println("Please Enter the employee's First Name: ");
        String firstName = bufRead.readLine();

        System.out.println("Please Enter the employee's middle initial: ");
        String minit = bufRead.readLine();

        System.out.println("Please Enter the employee's Last Name: ");
        String lastName = bufRead.readLine();

        System.out.println("Please Enter the employee's Ssn: ");
        String ssn = bufRead.readLine();

        System.out.println("Please Enter the employee's Department Number: ");
        String dno = bufRead.readLine();

        int dno_int = Integer.parseInt(dno);

        /*

        Add the additional data fields here

        */


      }catch (IOException err) {
        System.out.println(err);
      }catch(NumberFormatException err) {
        System.out.println(err);
      }

      //Now lets insert a new row of data

      System.out.println("\n****Inserting a New Employee*****");
      System.out.println();


      // First we specify our query
      Statement stmt = null;
      try {
        //Create a new statement object - notice the additional arguments for inserting
        stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
        //Get all record in the employee table
        ResultSet uprs = stmt.executeQuery("SELECT * FROM employee");

        //Create a new row in the ResultSet object
        uprs.moveToInsertRow();
        //Add new employee's information to the new row of data
```

```
        /*********************************************************/
        // This is where you will need to update the code to include
        // the data entered by the user.

        uprs.updateString("fname", ... );
        uprs.updateString("minit", ... );
        uprs.updateString("lname", ... );
        uprs.updateString("ssn", ... );
        uprs.updateInt("dno", ... );
        /*********************************************************/
        //Insert the new row of data to the database
        uprs.insertRow();
        //Move the cursor back to the start of the ResultSet object
        uprs.beforeFirst();
      }catch (SQLException e ) {
        System.out.println(e);
        conn.close();
        System.exit(1);
      }

      System.out.println("\nQuery Executed Successfully ... exiting");
      //Close the database connection
      conn.close();

    }


  }
```