Database Connectivity with JDBC

Objective
This lab will help you learn how to configure a Java project to include the necessary libraries for connecting to a DBMS. The lab will then give you the opportunity to experiment with sending queries to a database and executing stored procedures in the database. When you are done with the lab, you should have and understand sample code that you can use to manipulate a database from a Java program.

Required Materials
Laptop, network connection

Other Resources
- SQL Server 2005 JDBC Driver: *http://www.microsoft.com/downloads/details.aspx?familyid=6D483869-816A-44CB-9787-A866235EFC7C&displaylang=en*

Assignment Details
The sample code that you'll be working with in this lab is quite long. In order to encourage you to examine particular parts of the code, this lab requires that you answer several questions as you work through the lab. Questions, Result Sets, and SQL Queries appear in line and are marked by "**Question #:**", "**Result Set #:**", or "**SQL Query #:**". As with other written Lab work, please create a text file or PDF file from a word document with the answers to each question clearly marked (e.g. Question 1:, Result Set 2:, etc.). When you have completed this lab you will have answered 7 questions, copied in 4 result sets, and saved 1 SQL Query.

PLEASE NOTE: Lab 7 must be completed properly before starting this lab – Portions of this Lab require certain Stored Procedures be in place. Consult your Instructor or a TA if you require assistance with Lab 7.

You may continue working with your partner from Lab 7.

Task 1: Configure a Java Project to connect to SQL Server

1. Download the zipped Eclipse project file `JDBC.zip` from the links on the ANGEL website and but **don't** extract it.
2. Using the 'Import' Function of Eclipse, import the `JDBC.zip` archive into your workspace:
    a. Choose File --> Import
    b. Choose "Existing Projects into Workspace" and click Next
    c. Choose "Select archive file:" and find the `JDBC.zip` file that you downloaded.
3. Notice that the JAR file sqljdbc.jar is a registered library in the build path. This JAR file provides the necessary interfaces for JDBC to operation. A backup of this file has been provided in the 'lib' folder should you need it. (If you get errors about the jar file not being found, ask for help.)
4. This project should compile properly once it has been imported. Run the program.

   a. **Question 1:** What message is printed on the console when your run the program?
5. Before proceeding with the lab, take a few minutes to review the code contained in both Connection.java and JDBC.java
   a. Examine how the server and database names are selected
   b. Review each of the methods called by MAIN in JDBC.java
   c. **Question 2:** In your own words, give a one-sentence description of what each of these classes—Connection.java and JDBC.java—is for.

Task 2: Connect to the Server

Your next task is to establish a JDBC connection with the database server. The JDBC.java file contains an implementation and very little needs to be changed to make this possible.

1. In the main method, uncomment the code that prompts the user for their connection information. Run the program and experiment with entering different information in the dialog box.
   a. **Question 3:** What are the two different ways in which the wasCancelled field of ConnectionInfo can be set to true?
2. Back in the main method, uncomment the code to test the connection to the server. Edit the program so that you can successfully connect to our database server, `dyknow.cs.rose-hulman.edu`. You should study the testConnection method to figure out how to do this.
   a. **Question 4:** What did you have to change to get this connection to work?

Task 3: Send a Simple Query

You are now able to connect to and communicate with SQL Server 2005. You will now issue a simple query via the JDBC interface.

1. Uncomment the code in the `main` method that calls the `testQuery` method. Edit the program so that the query will run successfully.
   a. **Result Set 1:** Copy and paste the result of this query into your answers document.
2. Change the sample query so that it returns results for any specific country.that is in the table.
   a. Question 5: What did you change to do this?
3. Change the sample query to have the WHERE clause use a parameter, i.e. a question mark.
   a. Question 6: What did you have to do to get this to work?

Task 4: Call a Stored Procedure

In Lab 7 you learned about Stored Procedures. In this section of the Lab Assignment, you will issue a call to a Stored Procedure.

1. Uncomment the rest of the `main` method.

2. Use a New Query in SQL Management Studio to add the following stored procedure to your copy of the Northwind database:

```
CREATE PROC uspCustomersByCountry
        @country VARCHAR(25) = 'Germany',
        @count INT OUTPUT
AS

-- Shows how an OUTPUT parameter can be used to return a result
SET @count = (SELECT COUNT(*) FROM Customers WHERE Country = @country)

-- The following three "naked" queries show various things in the result sets.
SELECT CustomerID, CompanyName, ContactName FROM Customers WHERE Country =
@country
DELETE Customers WHERE 1=0
SELECT * FROM Orders o
        WHERE EXISTS(SELECT * FROM Customers c
                WHERE Country = @country AND o.CustomerID = c.CustomerID)

RETURN 0
GO
```

3. Edit this stored procedure to include error-handling code that checks the value of `@@ERROR` after each of the "naked" queries in this stored procedure. (See `SprocExamples.txt` on Angel for example code that checks `@@ERROR`.)

    a. **SQL File 1:** Save the edited SQL Query as Task4.sql

4. Back in Java, edit the program so that the sproc will run successfully. Then run the program.

    a. **Result Set 2:** Copy and paste all the results of the stored procedure into your answers document. Edit the results to indicate which line of the stored procedure is responsible for returning the particular piece of the results. You may need to comment out parts of the stored procedure and re-run the program to figure this out.

Task 5: Investigate JDBC Objects

The provided code reads data returned in tables from a query as Strings. JDBC lets you read some types of data directly. For example, you can read integer data as ints so that you don't have to convert a String to an integer. In this task, you'll change the displayRows method to also print the type of each piece of data in a result table.

In the provided code, the `processRows` method uses a `while` loop to find out what different sorts of data were returned by a query. This is like in Query Analyzer where a single script might return multiple tables, plus some messages. When the loop detects an update count that is less than zero, that indicates that a table of data—a `ResultSet`—is the next thing available. The method then calls `displayRows` to print out the table. That method iterates over the rows and columns of the table, calling `getString(i)` on the result set to get a `String` version of the data in column `i` of the current row.

1. Find the line inside the inner-most loop in `displayRows` that calls `rs.getString(i)`. After that line, insert the following code:

```
Object columnObject = rs.getObject(i);
if (columnObject != null) {
      System.out.print(":" + columnObject.getClass());
}else {
```

```
            System.out.print(":null");
        }
```

This code gets a Java object from column `i`, using whatever type is considered best for data from that column. The call to `getClass()` then returns and prints the type of that object.

2. Run your program again. Find the results that are drawn from the Orders table of Northwind.
    a. **Result Set 3:** Make a list of each column name and the Java type that is used to represent that column.
3. Look up `java.sql.ResultSet` in the Java API reference.
    a. **Question 7:** Besides getString and getObject, list the names of 5 different get methods that you can use to read data from columns of a result set.

The methods you listed for Question 7 may be used directly if you already know the type for a column in a particular result set.

Task 6: Call Another Stored Procedure

For this task you will create a new method that calls one of the procedures that you wrote for the Stored Procedures lab.

1. Start by duplicating the `executeStoredProcedure` method, giving the copy a new name – Remember to edit the main method to include a call to your new method.
2. Edit your new method to call your "update" stored procedure for Order Details. If your stored procedure has a space in its name, you'll have to put quotes around the procedure name when calling it. Test and debug until your stored procedure is executing correctly. Recall that to put a quotation mark inside a String in Java you must escape the quotation mark with a backslash. For example:

```
String query = "{ ? = call \"Some Procedure\"(?,?) }";
```

    1. **Result Set 4:** Copy and paste the results of running your Java program into your answers document.

Turn-in Instructions

Create a .rar or .zip archive containing:

- Edited JDBC.java file
- Text file or PDF with answers to the 7 Questions and copies of the 4 Result Sets
- Task4.sql Query
- a **who.txt** file with you and your partners name, if you worked with someone else

Upload this single archive file to the Lab dropbox on Angel. Submissions that are not compressed into a single archive will receive zero credit. After submitting this archive to ANGEL, please complete the feedback survey!

Revision History

Jan. 18, 2007:	Fixed bugs identified in class, Curt Clifton.
Jan. 17, 2007:	Changed instructions for importing into Eclipse, Curt Clifton.
Jan. 15, 2007:	Updated Lab and Source for use with SQL 2005, Bryan Musial.
Jan. 21, 2006:	Added tip about calling sproc with spaces in its name, Curt Clifton.
Jan. 19, 2006:	Fixed typos. Added borders to questions so they are obvious in print.
Jan. 18, 2006:	JCreator instructions enhanced by Steve Chenoweth
Jan. 18, 2006:	Lab instructions written by Curt Clifton
Dec. 2005:	Installation instructions written and sample code identified by Will Mathies.