### CSCI 470/680E -- Complete Assignment 6 -- Fall 2014

**Description**

This project will implement a cooperating client-server database system. You will write both the Server and the Client. Base your Server program on the Server program (<http://www.seasite.niu.edu/cs580java/javaServer.html> ) supplied on the course Web site.

The Server program must be stored and run on hopper.cs.niu.edu. Your instructor will let you know which port you will use.

Write your own client (You can start from <http://www.seasite.niu.edu/cs580java/javaClient.html>).  The client is a command line program. You need to design a simple text-based menu. It needs to be clear enough for the user to select one of the JDBC commands (e.g. 1. ADD; 2. SEARCH; 3. DELETE etc. See details of the commands below.), and pass necessary information to the program. Use prompt to remind user about the required input. Use validation to ensure the correct data types.

Theoretically, the Client should be able to run from any local client computer. However, due to the private subnet and the tightening of the firewall settings implemented this semester, there are some restrictions: 1) When testing your assignment on campus, your client can run from any other computer. 2) If you want to test your assignment off campus, then put both the client and the server program on hopper, and run both of them on hopper (using ssh to open different windows).

As a requirement for grading, you will demonstrate this system. In addition to the designated host and server port number, you are also assigned a database, and a demo time slot. All are posted in the Blackboard Assessment section with this assignment.

Information on JDBC is the document on the course website: <http://www.seasite.niu.edu/cs580java/jdbc.html> .

The application server program should connect to the mysql database.  Each programming team will have its own database to manipulate. Each database contains a single table whose name is **cust**, so this is the table name you should specify in all of your SQL queries (e.g., select \* from cust where name = 'nameString'). The cust table has 4 columns. These are:

Column Number         Column Name         Data Type  
        1                         name                         char(20)  
        2                         ssn                            char(10)  
        3                         address                     char(40)  
        4                         code                          int

The ssn column has a unique primary index on it and thus all values in this column must be unique.  All database names (table name, column names) are lowercase and must be coded as such in SQL statements.

These databases/tables might not be empty so your first transaction type to implement might be GETALL (see below).

Your client-server pair should perform the following operations on the database.

* **ADD** a new customer. The user will need to select this command, and then enter 4 pieces of information on the client application based on the 4 columns of the **cust** table. Once all the information are provided by the user, the client sends them to the server.

The server program should insert this new information into the database and inform the client of its success or failure. The client application should display a message indicating the result of the transaction. (Note that attempting to insert a record with a duplicate ssn will raise an SQLException.)

* **SEARCH** for a customer by SSN. The user will need to select this command, and then enter an ssn. Once all the information are provided, the client sends them to the server.

The server should search for the record with the customer's ssn and return the information back to the client for display in the console. If the customer is not found, have the server program notify the client of this and the client can display a "Not Found" message.

* **DELETE** a customer. The user will need to select this command, and then enter an ssn. Once all the information are provided, the client sends them to the server.

The server should attempt to delete the record with that customer's ssn and should inform the client of its success or failure. The client applet should display a message in console indicating the result of the transaction.

* **UPDATE** a customer **address**. The user will need to select this command, and then enter an ssn and the new address. Once all the information are provided, the client sends them to the server.

The server program should change the address for this ssn (if the ssn is found) and report to the client on update success or failure. The client should display a message in console indicating the result of the transaction.  The server should **not** do this by reading the existing record to ensure that the ssn exists in the database.  Rather it will prepare and execute a command and report on whether it succeeded of failed.  -- This means that the server must use a ***PreparedStatement****.*  Only **one** database transaction is required, rather than two.

* **GETALL** - when the user selects this command, the client sends the choice to the server.

The program should retrieve and display all the database records returned by the server in the console, one record per line.

**Development and Testing Details**

Again, see JDBC Notes at <http://www.seasite.niu.edu/cs580java/jdbc.html> for specific Driver loader notes for mysql.

Development and testing of client-server apps is somewhat awkward. To test a new version of the server, you will have to

* store the newly compiled version on unix/linux host (e.g. hopper.cs.niu.edu)
* *kill* the currently running server program
* start the new one (it is recommended to run the server in the background)

(See the section "Development and Testing" in the JDBC Notes referenced above.)

Testing a new version of the client is simpler - just re-compile and re-run.

**Application Protocol:**

You can use Strings to transfer messages back and forth between the client and server, including the type of command and the related information and the feedback from server.

**You are strongly encouraged to use** [**Object Serialization**](http://www.seasite.niu.edu/cs580java/Object_Serialization.html)(e.g. serialized object of a class that holds the customer information, <http://www.seasite.niu.edu/cs580java/Object_Serialization.html> )to send messages back and forth between the client and the server.  To minimize communications, include a transaction field in the objects sent from the client to the server to indicate the type of transactions (e.g. add, update, or list etc.), as opposed to sending a message indicating the transaction type followed by a separate message with the transaction details.  You will need to devise a way for the server to send multiple records from a *GetAll* request and for the client to retrieve those records. Even if you end up choosing to use Strings to send messages back and forth, you are still required to study the material for object serialization.

 The only access you have to the database is through your Java Server program. The database you connect to contains one table. The table is named **cust**, so this is the table you should specify in your SQL queries. This table is probably **not** initially empty.

When mysql was installed on *courses.cs.niu.edu*, certain default installation options were put into effect. One of these is that the mysql database server (the program that receives requests from the Java driver and sends back the results) is listening on port 3306. We left this option in effect, so this is the port your Java application server program should connect to; e.g.,

String url = "jdbc:mysql://courses:3306/JavaCustXX";

where NN = your assigned database number.

You have to make sure that your CLASSPATH environment variable is set in order to successfully run the server.

1.) to use the classes from Java 1.6  
2.) to use the classes from the the driver classes .jar file  
3.) to use the classes from your directory where your java application resides

So, edit the .bashrc file that is on your root directory, to include the following ***single*** line:

export CLASSPATH=.:/usr/share/java/\*

**Notes**:

1. The initial . before the colon is for the current directory (or working directory) where your server is located.

2. If you have trouble with your .bashrc file, you can just type in the export statement on the command line before you begin working in a new terminal session on turing.

**Be very careful** that you type this correctly.

**Submission**

The Client will can be run from any local client computer on campus. As a requirement for grading, you will demonstrate this system. You will need to bring a laptop with the client on it or a USB drive with all the required files to run the client in your instructor's office (or other given location) for the demonstration.

You will need to demonstrate the system in your instructor’s office (or other location given by your instructor). **You should start the server** before you come to demo the system.

**You should have submitted your source code** in Blackboard by the due date (which will be earlier than your demo time)**.** You can come earlier than the assigned demo time slot, but not later. No late submission is allowed for this assignment.