CSCI 470/680E Assignment 5

This project will implement both the client and the server of a cooperating client-server database system. Skeleton code for the server and client programs is available on Blackboard. The GUI design for your client application may be quite simple, but it should be clear, nicely organized, and should not require excessive scrolling, etc.

The Server program should be stored and run on `turing.cs.niu.edu` or `hopper.cs.niu.edu`. The skeleton client program code posted on Blackboard assumes that the server is running on `turing`. If your Unix account is located on `hopper`, you will need to make sure you change the hostname in the Client OR log in to your account through `turing` when you want to run the Server.

See the list posted on Blackboard for the name of your database and the port number to use with your server.

The server program should connect to the `mysql` database management system. Each student will have their own database to manipulate. Each database contains a single table whose name is `customer`, so this is the table name you should specify in all of your SQL queries (e.g., `SELECT * FROM customer`). The `customer` table has 4 columns. These are:

Column Number	Column Name	Data Type	Notes
1	`name`	VARCHAR(20)	NOT NULL
2	`ssn`	CHAR(11)	NOT NULL, PRIMARY KEY
3	`address`	VARCHAR(40)	NOT NULL
4	`zipCode`	CHAR(5)	NOT NULL

The `ssn` column is the primary key 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.

All of the database tables are initially empty so your first transaction type to implement might be ADD (see below).

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

ADD a new customer. The user has entered information into four `JTextField`s in the client and has pressed an “add” button. The client should validate the input entered into the text fields:

- The `name` should not be empty and should not be greater than 20 characters in length.
- The `ssn` should not be empty, should be exactly 11 characters in length, and be composed solely of digits and the hyphen character in the format *`999-99-9999`*.
- The address should not be empty and should not be greater than 40 characters in length.
- The `zipCode` should not be empty, should be exactly 5 characters in length, and should only contain numeric digits.

If one or more fields are invalid, display an error message (in a `JLabel`, for example) and do not send an add request to the server.

If the transaction data is valid, an add request containing that data should be sent to the server. The server program should attempt to insert this new information into the database and then 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 throw an `SQLException`.)

DELETE a customer. The user has provided an SSN in the `JTextField` and has pressed a "delete" button. The client should verify that the SSN is valid (see above), and if so, transmit a delete request containing the SSN to the server.

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

UPDATE a customer address. The user has entered an SSN and the new address and pressed an "update" button. The client should verify that the SSN and the address are valid (see above), and transmit an update request containing that info to the server. The server program should change the address for that SSN (if the ssn is found) and report to the client on update success or failure. The client should display a message 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 should prepare and execute a command and report on whether it succeeded of failed. (This means that a `PreparedStatement` needs to be used and only one database transaction is required, rather than two.)

GETALL the records. When the user presses this button, the program should retrieve and display all of the database records in a `JTextArea`. This `JTextArea` should be wide enough to display all of the data for one record and must support vertical scrolling (using the `JScrollPane` class). The output records need not be neatly formatted – just dump the raw information, one record per line, with some character used to separate the fields. The number of records retrieved should be also be displayed (perhaps in a `JLabel`).

Development and Testing Details

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

- kill the currently running server program.
- store the newly compiled version on our Unix system.
- start the new one (running in the background).

Application Protocol

You are strongly encouraged to use Object Serialization to send messages back and forth between the client and the server (the skeleton code posted on Blackboard assumes that you will do so). To minimize communications, include a transaction field in the objects sent from the client to the server to indicate GETALL, ADD, UPDATE, or DELETE (as opposed to sending a message indicating the transaction type followed by a separate message with the transaction details).

Likewise, you may also want to include a field to allow a success or error message to be sent back to the client from the server.

You will need to devise a way for the server to send multiple records from a GETALL request and for the application to retrieve that number of records. Even if you end up choosing to use `String`s to send messages back and forth, you are still required to study the material on object serialization for the final exam.

Programming Notes

See the **JDBC Notes** posted on Blackboard for more details.