

Problem 1) Create a Debian Linux Virtual machine by downloading OS ISO file from <http://www.debian.org>. Please use amd64 architecture. Install Oracle's Java Development Kit 1.7_xx. Please do not download Java8. Cassandra might or might not work with Java8. Download Cassandra 2.0.6 from <http://cassandra.apache.org> and install. Verify that you can start Cassandra and can successfully open `cqlsh` prompt. Create your own keyspace, create a table with three "column families" and insert two rows into that table. Demonstrate that you can query inserted rows.

Problem 2) Install a relatively recent Eclipse in your Debian VM. Download Cassandra Java driver 2.0 from <http://www.datastax.com>. Downloaded file has some 6MB and should be named like: `cassandra-java-driver-2.0.0.tar.gz`. Expand the archive. You will see two `cassandra-driver-...jar` files. Also, in the subdirectory `lib`, you will see several additional `jar` files. You will add all of those to the Build Path of your Eclipse project. Next, create a Java Project in your Eclipse. Move attached class `SimpleClient` into the project. Place attached `log4j.properties` file in the `src` directory of your project. Add above mentioned jars to the Build Path of the project. Make sure that Cassandra is started. Run your `SimpleClient` class as a Java Application. Capture console output. It should basically say that you are running a single machine Cassandra cluster on the host 127.0.0.1. Modify your `log4j.properties` to stop so many DEBUG lines from being printed out. Capture all the steps, working code and resulting console outputs. Submit modified `log4j.properties` file, as well.

Problem 3) Add attached class `CQLClient` to your Java project. As you can see this class performs basic CQL operations on your Cassandra database. It opens a session to Cassandra cluster, creates a new keyspace, creates new table, inserts and queries some rows in that table. Change the name of the keyspace, table name and the column names. In the insert statements, replace values appropriate for your table and columns. Run your `CQLClient` class as a Java Application. Capture Eclipse console output. Submit working code and console output. Next open your `cqlsh` prompt, switch to (use) keystore your Java program created and demonstrate that you can select the values your Java program inserted into Cassandra. Add Debian Linux and Cassandra to your Resume. Descriptions of classes `SimpleClient` and `CQLClient` are added to the lecture notes and modified lecture notes are uploaded to the class site.

Problem 4) Placing hard-coded values inside your CQL (SQL) statements, as we did in the previous problem, is considered a bad programming practice. For all kind of reasons, including application security, code reuse and application performance, you want to be able to write generic CQL (SQL) statements which have placeholders for values and then assign concrete values at the moment when you want to perform database operations. In the class `CQLClient` we executed such hard coded (CQL) SQL statements using method `execute()` on the `Session` object. As suggested, a better way is to create objects of `PreparedStatement` type. Those objects will contain CQL statements and bind values

(place-holders). Prepared statements will only need to be parsed once by Cassandra cluster. We will bind values to the variables and execute the bound statements when we want to read or write data from or to Cassandra's tables.

In your project, create a new class called `PerparedClient` by copying the content of `CQLClient`. Next, modify `loadData()` method. Add code to your client for:

- creating a prepared statement
- creating a bound statement from the prepared statement and binding values to its variables
- executing the bound statement to insert data

Add code to prepare an INSERT statement. You get a prepared statement by calling the `prepare` method on your session.

```
PreparedStatement statement = getSession().prepare(
    "INSERT INTO simplex.songs " +
    "(id, title, album, artist) " +
    "VALUES (?, ?, ?, ?);");
```

Add code to bind values to the prepared statement's variables and then execute the statement. You create a bound statement by calling its constructor and passing in the prepared statement. Use the `bind` method to bind values and execute the bound statement on your session.

```
BoundStatement boundStatement = new
BoundStatement(statement);
getSession().execute(boundStatement.bind(
    UUID.fromString("756716f7-2e54-4715-9f00-
91dcbea6cf50"),
    "La Petite Tonkinoise",
    "Bye Bye Blackbird",
    "Joséphine Baker" ) );
```

Note that you cannot pass in string representations of UUIDs or sets as you did in the previous `loadData()` method.

Add code to create a new bound statement for inserting data into the `simplex.playlists` table.

```
statement = getSession().prepare(
    "INSERT INTO simplex.playlists " +
    "(id, song_id, title, album, artist) " +
    "VALUES (?, ?, ?, ?, ?);");
boundStatement = new BoundStatement(statement);
getSession().execute(boundStatement.bind(
    UUID.fromString("2cc9ccb7-6221-4ccb-8387-f22b6a1b354d"),
    UUID.fromString("756716f7-2e54-4715-9f00-91dcbea6cf50"),
    "La Petite Tonkinoise",
    "Bye Bye Blackbird",
```

```
"Joséphine Baker") );
```

Review the `main()` method of your class.

```
public static void main(String[] args) {  
    PreparedClient client = new PreparedClient();  
    client.connect("127.0.0.1");  
    client.createSchema();  
    client.loadData();  
    client.querySchema();  
    client.close();  
}
```

Of course, in the above, replace the keyspace name, table names and column names with names you used in your version of `CQLClient` class. Before running this new class go to the `cqlsh` prompt and drop your existing tables and the existing keyspace. Otherwise, you will get an error telling you that a keyspace (tables) with existing name(s) already exist.

Submit the working code and all console outputs.

As usual, please capture all the steps of your implementation, with comments indicating what is it you are accomplishing with every step, in an MS Word document. **PLEASE capture code as text files and insert into the Word document. ALWAYS provide separate copies of entire scripts or Java classes. We might want to run your code and we do not have time to retype the code by reading it from your screenshots.**

Please place all files you want to submit in a folder named: HW10. Compress that folder into an archive named E63_LastNameFirstNameHW10. ZIP. Upload the archive to the course drop box on the class web site. Please send comments and questions to cscie63@fas.harvard.edu