## HU Extension Assignment 07 E63 Big Data Analytics

### Handed out: 03/12/2016 Due by 11:30PM EST, 03/25/2016

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used as separate files. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. Please, submit to the class drop box. For issues and comments visit the class Discussion Board. The following problems are formulated in Java, however you can solve the following problems using any language of your choice that is supported by Cassandra Client API-s. You are not obliged to use Eclipse. You are welcome to use any IDE of your choice.

**Problem 1)** Install Cassandra server on your Cloudera VM. Use one of the methods described in notes.Use Cassandra SQL Client, cqlsh, to create and populate table person. Let every person by described by his or her first and last name, and city where he or she lives. Let every person possess up to three cell phones. Populate your table with three individuals using cqlsh client. Demonstrate that you can select the content of your table person.

CREATE KEYSPACE mykeyspace WITH REPLICATION = { 'class' :'SimpleStrategy', 'replication\_factor' : 1 };

create table person (user\_id int PRIMARY KEY, fname text, lname text , city text, cell\_phone1 text, cell\_phone2 text, cell\_phone3 text);

insert into person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (1, 'Rohan', 'Pulekar', 'Waltham', '6174591008', '6178555244', NULL);

insert into person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (2, 'Vinita', 'Chaudhari', 'Waltham', '8432759393', '6178555244', '9476665544');

insert into person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (3, 'Gauri', 'Pulekar', 'Worcester', '3728484938', NULL, NULL);

select \* from person;

**Problem 2)** Create an Eclipse project. Move attached class SimpleClient into the project. Place attached log4j.properties file in the src directory of your project. Properly set the Build Path of your 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.

**Problem 3)** Write a simple Java client starting from the attached Java class CQLClient to your Java project. As you can see this class performs basic CQL operations on your Cassandara database. It opens a session to Cassandra cluster, creates a keyspace, creates new table, inserts and queries some rows in that table. Modify that class so that it creates, populates and queries table person introduced in Problem 1. You might want to run this problem in a Cassandra keyspace different from the one created in Problem 1. Modify your log4j.properties to stop DEBUG lines from being printed out. Capture all the steps, working code and resulting console outputs. Submit modified log4j.properties file, as well.

CREATE KEYSPACE assignment7\_problem3 WITH REPLICATION = { 'class' :'SimpleStrategy', 'replication\_factor' : 1 };

**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. 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 mykeyspace.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 keyspaces if they overlap with ones in this problem. Otherwise, you might get an error telling you that a keyspace (tables) already exist.

Submit the working code and all console outputs.

CREATE KEYSPACE prepared\_client WITH REPLICATION = { 'class' :'SimpleStrategy', 'replication\_factor' : 1 };

**Problem 5)** Instantiate a micro Amazon Linux instance in AWS Cloud. Download a Tomcat 8 distribution to your local machine and then transfer the file to the AWS instance using an scp command. Install Tomcat on your remote instance. Verify that port 8080 is set properly in Tomcat’s server.xml configuration file. Start Tomcat on remote machine. Demonstrate that you can use your browser to open the Welcome page of the remote Tomcat.