

Assignment 7 Solution

[Problem 1.](#)

[Problem 2.](#)

[Problem 3.](#)

[Problem 4.](#)

[Problem 5.](#)

[Code](#)

[3. CQLClient.java](#)

[4. PreparedClient.java](#)

Problem 1.

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 be 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.

```
// A group named cassandra, for the new user cassandra, is created with groupadd.
```

```
[root@quickstart ~]# groupadd cassandra
```

```
// The user cassandra is created with the group cassandra from the previous step.
```

```
[root@quickstart ~]# useradd cassandra -g cassandra
```

```
// The user cassandra's password is changed to cassandra.
```

```
[root@quickstart ~]# passwd cassandra
```

```
Changing password for user cassandra.
```

```
New password:
```

```
BAD PASSWORD: it is based on a dictionary word
```

```
Retype new password:
```

```
passwd: all authentication tokens updated successfully.
```

```
// Permissions on the sudoers file are edited to allow changes.
```

```
[root@quickstart ~]# chmod +w /etc/sudoers
```

```
// The sudoers file is opened, lines are added for cassandra that are similar to
// those for cloudera, and the JAVA_HOME and PATH are set for the system to remember.
```

```
[root@quickstart ~]# visudo -f /etc/sudoers
```

```
Added ...
```

```
cloudera ALL=(ALL) NOPASSWD: ALL
cassandra ALL=(ALL) NOPASSWD: ALL
Defaults env_keep += "LC_TIME LC_ALL LANGUAGE LINGUAS
_XKB_CHARSET XAUTHORITY"
Defaults env_keep += "JAVA_HOME PATH"
```

```
// Permissions are reset on the sudoers file.
```

```
[root@quickstart ~]# chmod -w /etc/sudoers
```

```
// The apache cassandra file, after downloading it, is expanded. The last 15+ lines of output
// are included after the command.
```

```
[cassandra@quickstart ~]$ tar -xzf apache-cassandra-2.1.13-bin.tar.gz
```

```
...
```

```
apache-cassandra-2.1.13/bin/stop-server.ps1
apache-cassandra-2.1.13/tools/bin/cassandra-stress
apache-cassandra-2.1.13/tools/bin/cassandra-stress.bat
apache-cassandra-2.1.13/tools/bin/cassandra-stressd
apache-cassandra-2.1.13/tools/bin/cassandra.in.bat
apache-cassandra-2.1.13/tools/bin/cassandra.in.sh
apache-cassandra-2.1.13/tools/bin/json2sstable
apache-cassandra-2.1.13/tools/bin/json2sstable.bat
apache-cassandra-2.1.13/tools/bin/sstable2json
apache-cassandra-2.1.13/tools/bin/sstable2json.bat
apache-cassandra-2.1.13/tools/bin/sstableexpiredblockers
apache-cassandra-2.1.13/tools/bin/sstableexpiredblockers.bat
apache-cassandra-2.1.13/tools/bin/sstablelevelreset
apache-cassandra-2.1.13/tools/bin/sstablemetadata
apache-cassandra-2.1.13/tools/bin/sstablemetadata.bat
apache-cassandra-2.1.13/tools/bin/sstableofflinerelevel
apache-cassandra-2.1.13/tools/bin/sstable repairedset
apache-cassandra-2.1.13/tools/bin/sstablesplit
apache-cassandra-2.1.13/tools/bin/sstablesplit.bat
apache-cassandra-2.1.13/tools/bin/token-generator
apache-cassandra-2.1.13/tools/bin/token-generator.bat
```

// The file is renamed.

```
[cassandra@quickstart ~]$ mv apache-cassandra-2.1.13 cassandra-2.1.13
```

// At this point, /home/cassandra/cassandra-2.1.13/bin is added to the file ~/.bash_profile.

// Next the bash_profile is sourced.

```
[cassandra@quickstart ~]$ source ~/.bash_profile
```

```
[cassandra@quickstart ~]$ cd ~cassandra/cassandra-2.1.13/bin
```

// Finally, cassandra is started and is kept as a foreground process for close monitoring.

// The last approximately 15 lines of output are included from the command.

```
[cassandra@quickstart bin]$ cassandra -f
```

...

```
INFO 20:27:39 Compacted 4 sstables to
```

```
[/home/cassandra/cassandra-2.1.13/bin/./data/data/system/local-7ad54392bcdd35a684174e047860b377/system-local-ka-13,]. 6,333 bytes to 5,758 (~90% of original) in 231ms = 0.023772MB/s. 4 total partitions merged to 1. Partition merge counts were {4:1, }
```

```
INFO 20:27:40 Netty using native Epoll event loop
```

```
INFO 20:27:40 Using Netty Version: [netty-buffer=netty-buffer-4.0.23.Final.208198c,
```

```
netty-codec=netty-codec-4.0.23.Final.208198c,
```

```
netty-codec-http=netty-codec-http-4.0.23.Final.208198c,
```

```
netty-codec-socks=netty-codec-socks-4.0.23.Final.208198c,
```

```
netty-common=netty-common-4.0.23.Final.208198c,
```

```
netty-handler=netty-handler-4.0.23.Final.208198c,
```

```
netty-transport=netty-transport-4.0.23.Final.208198c,
```

```
netty-transport-rxtx=netty-transport-rxtx-4.0.23.Final.208198c,
```

```
netty-transport-sctp=netty-transport-sctp-4.0.23.Final.208198c,
```

```
netty-transport-udt=netty-transport-udt-4.0.23.Final.208198c]
```

```
INFO 20:27:40 Starting listening for CQL clients on localhost/127.0.0.1:9042...
```

```
INFO 20:27:40 Binding thrift service to localhost/127.0.0.1:9160
```

```
INFO 20:27:40 Listening for thrift clients...
```

// Cassandra's command line interface is accessed with cqlsh.

```
[cassandra@quickstart bin]$ cqlsh
```

```
Connected to Test Cluster at 127.0.0.1:9042.
```

```
[cqlsh 5.0.1 | Cassandra 2.1.13 | CQL spec 3.2.1 | Native protocol v3]
```

```
Use HELP for help.
```

// A new keyspace named mykeyspace is created and set to be used.

```
cqlsh> CREATE KEYSPACE mykeyspace WITH REPLICATION = { 'class' : 'SimpleStrategy',  
'replication_factor' : 1 };
```

```
cqlsh> use mykeyspace;
```

```
// A new table named person is created with the user id as the primary key. It contains the
// information specified from the handout.
cqlsh:mykeyspace> CREATE TABLE person( id uuid PRIMARY KEY, first text, last text, city
text, cell_1 text, cell_2 text, cell_3 text );
```

```
// Data for the user Ryan Ballenger is inserted into the table person.
cqlsh:mykeyspace> INSERT INTO person (id, first, last, city, cell_1, cell_2, cell_3) VALUES
(uuid(), 'Ryan', 'Ballenger', 'Somerville', '6144064942', null, null);
```

```
// Data for the user George Washington is inserted into the table person.
cqlsh:mykeyspace> INSERT INTO person (id, first, last, city, cell_1, cell_2, cell_3) VALUES
(uuid(), 'George', 'Washington', 'DC', '6145556666', null, null);
```

```
// Data for the user Ice Cube is inserted into the table person.
cqlsh:mykeyspace> INSERT INTO person (id, first, last, city, cell_1, cell_2, cell_3) VALUES
(uuid(), 'Ice', 'Cube', 'Compton', '6141112222', null, null);
```

```
// The table person is displayed with the select command that includes all but the last
// two cell phone columns.
```

```
cqlsh:mykeyspace> select first, last, city, cell_1, id from person;
```

first	last	city	cell_1	id
Ryan	Ballenger	Somerville	6144064942	d87b750d-520e-48e5-ade2-4e77fdca0078
George	Washington	DC	6145556666	eb1f4564-ccac-4eac-bf87-89ad5445cc06
Ice	Cube	Compton	6141112222	a260f1c5-1014-48df-9123-7a4a2cff4beb

(3 rows)

Problem 2.

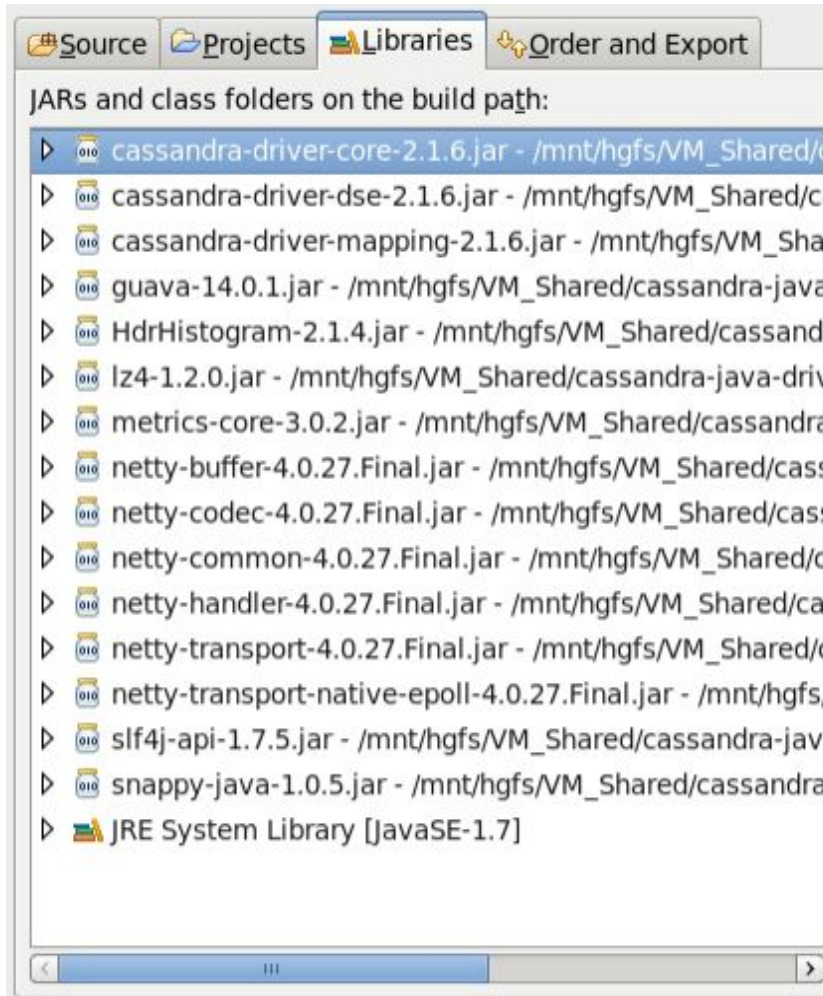
Problem 2) Create an Eclipse project. Move attached class SimpleClient into the project. Place attached log4j.properties file in the src directory of your project. Properly set the Build Path of your project. Make sure that Cassandra is started. Run your SimpleClient class as a Java Application. Capture console output. It should basically say that you are running a single machine Cassandra cluster on the host 127.0.0.1.

```
// A new Eclipse project was created to run SimpleClient with the log4j properties.
```

// File -> New -> Project -> Java Project

// log4j.properties and SimpleClient were copied into the src folder. SimpleClient was
// also placed inside edu/hu/cassandra within the src folder for the package placement.

// Jar files are added to the classpath. The .jar's are from cassandra-java-driver-2.1.6 and
// include the following .jar files:



// Lastly, SimpleClient is run as a java application and the output is below.

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See <http://www.slf4j.org/codes.html#StaticLoggerBinder> for further details.
Connected to cluster: Test Cluster
Datacenter: datacenter1; Host: /127.0.0.1; Rack: rack1

Problem 3.

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 Cassandra 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.

// CQLClient has been modified to produce the person table from earlier. For the first run, the
// debugger is still on to confirm the program and the debug option functions properly. The end
// of the output is pasted below including the connection information and query output.

...

```
DEBUG [cluster1-nio-worker-1](Connection.java:158) - Connection[/127.0.0.1:9042-2,
inFlight=0, closed=false] Connection opened successfully
DEBUG [cluster1-nio-worker-1](SessionManager.java:308) - Added connection pool for
/127.0.0.1:9042
Connected to cluster: Test Cluster
Datacenter: datacenter1; Host: /127.0.0.1; Rack: rack1
DEBUG [cluster1-nio-worker-0](Cluster.java:2074) - Received event EVENT DROPPED TABLE
mykeyspacetwo.person, scheduling delivery
DEBUG [cluster1-nio-worker-0](Cluster.java:2074) - Received event EVENT DROPPED
KEYSPACE mykeyspacetwo, scheduling delivery
DEBUG [cluster1-worker-0](ControlConnection.java:657) - Checking for schema agreement:
versions are [5e6d065a-e6c2-32ae-a2a0-3acb928b8703]
DEBUG [cluster1-nio-worker-0](Cluster.java:2074) - Received event EVENT CREATED
KEYSPACE mykeyspacetwo, scheduling delivery
DEBUG [cluster1-worker-0](ControlConnection.java:288) - [Control connection] Refreshing
schema for mykeyspacetwo
DEBUG [cluster1-worker-0](ControlConnection.java:530) - [Control connection] Refreshing node
list and token map
DEBUG [cluster1-nio-worker-1](Cluster.java:2022) - Refreshing schema for mykeyspacetwo
DEBUG [cluster1-worker-0](ControlConnection.java:657) - Checking for schema agreement:
versions are [d3e26aaf-f694-3ab8-8b6d-41ee54b9c643]
DEBUG [cluster1-worker-0](ControlConnection.java:530) - [Control connection] Refreshing node
list and token map
DEBUG [cluster1-nio-worker-0](Cluster.java:2074) - Received event EVENT CREATED TABLE
mykeyspacetwo.person, scheduling delivery
```

```
DEBUG [cluster1-worker-0](ControlConnection.java:288) - [Control connection] Refreshing
schema for mykeyspacetwo.person (table)
DEBUG [cluster1-nio-worker-1](Cluster.java:2022) - Refreshing schema for
mykeyspacetwo.person
DEBUG [cluster1-worker-0](ControlConnection.java:657) - Checking for schema agreement:
versions are [c974b7bb-c3a5-3d41-b41a-0b93a2e6cd78]
First   | Last   | City   | Cell #1 | Cell #2 | Cell #3 |
```

```
-----
Ice     | Cube   | Compton | 6141112222 | null    | null    |
George  | Washington | DC      | 6145556666 | null    | null    |
Ryan    | Ballenger | Somerville | 6144064942 | null    | null    |
```

```
DEBUG [main](Cluster.java:1399) - Shutting down
DEBUG [main](Connection.java:618) - Connection[/127.0.0.1:9042-1, inFlight=0, closed=true]
closing connection
DEBUG [main](Connection.java:618) - Connection[/127.0.0.1:9042-2, inFlight=0, closed=true]
closing connection
```

```
// As directed in the handout, the log4j.properties file is edited to stop the DEBUG lines.
// The new file is below followed by the output from CQLClient with the new configuration.
```

log4j.properties

```
log4j.rootLogger=OFF
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t](%F:%L) - %m%n
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=example.log
log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=1
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

```
// CQLClient output with the DEBUG lines turned off as described earlier.
```

```
Connected to cluster: Test Cluster
Datacenter: datacenter1; Host: /127.0.0.1; Rack: rack1
First   | Last   | City   | Cell #1 | Cell #2 | Cell #3 |
-----
Ice     | Cube   | Compton | 6141112222 | null    | null    |
George  | Washington | DC      | 6145556666 | null    | null    |
```

Problem 4.

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.

```

// PreparedClient is created with CQLClient as a starting place and then loadData() is modified
// to use prepared statements rather than hard-coding. The output is below and the
// PreparedClient.java code is at the end of this document. A screenshot is used to maintain
// the output spacing.

```

```

<terminated> PreparedClient [Java Application] /usr/java/jdk1.7.0_67-cloudera/bin/java (Mar 25, 2016, 7:17:28 PM)
Connected to cluster: Test Cluster
Datacenter: datacenter1; Host: /127.0.0.1; Rack: rack1

```

First	Last	City	Cell #1	Cell #2	Cell #3	ID
Ryan	Ballenger	Somerville	6144064942	null	null	81010d4e-0b76-497b-a355-52045078cf7a
George	Washington	DC	6145556666	null	null	5d1163a0-104d-4749-ac32-e5583a49f1d0
Ice	Cube	Compton	6141112222	null	null	d31f1e3f-9e4b-45c9-acdf-8aa3b73812f5

Problem 5.

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.

Steps on AWS

1. Amazon EC2 console
2. Click Launch Instance
3. Select Amazon Linux AMI
4. Select t2.micro
5. Click Review and Launch
6. Click Launch
7. Click Create a new key pair and name it MyFirstKey
8. Click Download Key Pair
9. Click Launch Instance

// The key is moved where AWS recommends for Mac users.

```
Ryans-MacBook-Pro:~ Ryan$ cp MyFirstKey.pem ~/.ssh/
```

// This command ensures the private key is not publically viewable.

```
Ryans-MacBook-Pro:~ Ryan$ chmod 400 ~/.ssh/MyFirstKey.pem
```

// A Tomcat 8 distribution is downloaded and the scp command is used to move it
// to the AWS linux instance.

```
Ryans-MacBook-Pro:~ Ryan$ scp -r -i ~/.ssh/MyFirstKey.pem  
/Users/Ryan/Downloads/apache-tomcat-8.0.32 ec2-user@52.207.215.141:/home/ec2-user/  
.DS_Store          100% 12KB 12.0KB/s 00:00  
.DS_Store          100% 12KB 12.0KB/s 00:00  
bootstrap.jar      100% 28KB 27.8KB/s 00:00  
catalina-tasks.xml 100% 1686 1.7KB/s 00:00  
catalina.bat       100% 14KB 13.5KB/s 00:00  
catalina.sh        100% 21KB 20.9KB/s 00:00  
commons-daemon-native.tar.gz 100% 200KB 200.1KB/s 00:00  
commons-daemon.jar 100% 24KB 23.7KB/s 00:00  
configtest.bat     100% 2040 2.0KB/s 00:00  
configtest.sh      100% 1922 1.9KB/s 00:00  
daemon.sh          100% 7888 7.7KB/s 00:00  
digest.bat         100% 2091 2.0KB/s 00:00  
digest.sh          100% 1965 1.9KB/s 00:00  
setclasspath.bat   100% 3430 3.4KB/s 00:00  
setclasspath.sh    100% 3547 3.5KB/s 00:00  
shutdown.bat       100% 2020 2.0KB/s 00:00  
shutdown.sh        100% 1902 1.9KB/s 00:00
```

...

// The following SSH command connects to the Linux instance.

Ryans-MacBook-Pro:~ Ryan\$ ssh -i ~/.ssh/MyFirstKey.pem ec2-user@52.207.215.141

// Permissions are adjusted to make the files in the Apache folder executable.

[ec2-user@ip-172-31-50-40 ~]\$ chmod a+x apache-tomcat-8.0.32/bin/*

// The startup.sh file is executed to launch Tomcat and the output is listed below.

[ec2-user@ip-172-31-50-40 ~]\$./apache-tomcat-8.0.32/bin/startup.sh

Using CATALINA_BASE: /home/ec2-user/apache-tomcat-8.0.32

Using CATALINA_HOME: /home/ec2-user/apache-tomcat-8.0.32

Using CATALINA_TMPDIR: /home/ec2-user/apache-tomcat-8.0.32/temp

Using JRE_HOME: /usr/lib/jvm/jre

Using CLASSPATH:

/home/ec2-user/apache-tomcat-8.0.32/bin/bootstrap.jar:/home/ec2-user/apache-tomcat-8.0.32/bin/tomcat-juli.jar

Tomcat started.

// Also security had to be adjusted on AWS. **Security Groups**->Click on the right instance ->

// **Add Rule** -> **Port** should be set to **8080** and **Source** should be **Anywhere**

// Lastly, <http://ec2-52-207-215-141.compute-1.amazonaws.com:8080/> is visited and

// the Tomcat page states that it was successfully installed on the Linux instance.


← → ↻ ec2-52-207-215-141.compute-1.amazonaws.com:8080 ☆ ☰

Home Documentation Configuration Examples Wiki Mailing Lists Find Help

Apache Tomcat/8.0.32

The Apache Software Foundation
http://www.apache.org/

If you're seeing this, you've successfully installed Tomcat. Congratulations!

 Recommended Reading:
[Security Considerations HOW-TO](#)
[Manager Application HOW-TO](#)
[Clustering/Session Replication HOW-TO](#)

Server Status
Manager App
Host Manager

Developer Quick Start

Tomcat Setup Realms & AAA Examples Servlet Specifications
First Web Application JDBC DataSources Tomcat Versions

Managing Tomcat

For security, access to the `manager.webapp` is restricted. Users are defined in:

```
$CATALINA_HOME/conf/tomcat-users.xml
```

In Tomcat 8.0 access to the manager application is split between different users.
[Read more...](#)

[Release Notes](#)
[Changelog](#)
[Migration Guide](#)
[Security Notices](#)

Documentation

[Tomcat 8.0 Documentation](#)
[Tomcat 8.0 Configuration](#)
[Tomcat Wiki](#)

Find additional important configuration information in:

```
$CATALINA_HOME/RUNNING.txt
```

Developers may be interested in:

[Tomcat 8.0 Bug Database](#)
[Tomcat 8.0 JavaDocs](#)
[Tomcat 8.0 SVN Repository](#)

Getting Help

FAQ and Mailing Lists

The following mailing lists are available:

[tomcat-announce](#)
Important announcements, releases, security vulnerability notifications. (Low volume).

[tomcat-users](#)
User support and discussion

[taglibs-user](#)
User support and discussion for [Apache Taglibs](#)

[tomcat-dev](#)
Development mailing list, including commit messages

Code

3. CQLClient.java

```
package edu.hu.cassandra;

import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.Host;
import com.datastax.driver.core.Metadata;
import com.datastax.driver.core.Session;
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Row;

public class CQLClient {
    private Cluster cluster;
    private Session session;

    public void connect(String node) {
        cluster = Cluster.builder()
            .addContactPoint(node).build();
        session = cluster.connect("myKeySpace");
        Metadata metadata = cluster.getMetadata();
        System.out.printf("Connected to cluster: %s\n",
            metadata.getClusterName());
        for ( Host host : metadata.getAllHosts() ) {
            System.out.printf("Datacenter: %s; Host: %s; Rack: %s\n",
                host.getDatacenter(), host.getAddress(), host.getRack());
        }
    }

    public void createSchema() {
        session.execute("DROP KEYSPACE IF EXISTS myKeySpaceTwo ;");

        session.execute("CREATE KEYSPACE myKeySpaceTwo WITH replication " +
            "= {'class':'SimpleStrategy', 'replication_factor':1};");

        session.execute(
            "CREATE TABLE myKeySpaceTwo.person (" +
            "id uuid PRIMARY KEY," +
```

```

        "first text,"+
        "last text,"+
        "city text,"+
        "cell_1 text,"+
        "cell_2 text,"+
        "cell_3 text);");
    }

    public void loadData() {
        session.execute(
            "INSERT INTO myKeySpaceTwo.person (id, first, last, city, cell_1, cell_2,
cell_3)" +
            "VALUES (uuid(), 'Ryan', 'Ballenger', 'Somerville', '6144064942',
null, null);");

        session.execute(
            "INSERT INTO myKeySpaceTwo.person (id, first, last, city, cell_1, cell_2,
cell_3)" +
            "VALUES (uuid(), 'George', 'Washington', 'DC', '6145556666', null,
null);");

        session.execute(
            "INSERT INTO myKeySpaceTwo.person (id, first, last, city, cell_1, cell_2,
cell_3)" +
            "VALUES (uuid(), 'Ice', 'Cube', 'Compton', '6141112222', null,
null);");
    }

    public void querySchema(){
        ResultSet results = session.execute("SELECT * FROM
myKeySpaceTwo.person;"); //+
        //"WHERE id = 2cc9ccb7-6221-4ccb-8387-f22b6a1b354d;");
        System.out.println(String.format("%-10s | %-10s | %-10s | %-10s | %-10s |
%-10s | \n%s", "First", "Last", "City", "Cell #1", "Cell #2", "Cell #3",
"-----"));
        for (Row row : results) {
            System.out.println(String.format("%-10s | %-10s | %-10s | %-10s |
%-10s | %-10s | ", row.getString("first"),
row.getString("last"), row.getString("city"),
row.getString("cell_1"),row.getString("cell_2"),row.getString("cell_3")));
        }
        System.out.println();
    }

```

```

    }

    public void close() {
        cluster.close(); // .shutdown();
    }

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

```

4. **PreparedClient.java**

```

package edu.hu.cassandra;

import java.util.*;

import com.datastax.driver.core.BoundStatement;
import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.Host;
import com.datastax.driver.core.Metadata;
import com.datastax.driver.core.PreparedStatement;
import com.datastax.driver.core.Session;
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Row;

import com.datastax.driver.core.Statement;

    public class PreparedClient {
        private Cluster cluster;
        private Session session;

        public void connect(String node) {

```

```

cluster = Cluster.builder()
    .addContactPoint(node).build();
session = cluster.connect("mykeyspace");
Metadata metadata = cluster.getMetadata();
System.out.printf("Connected to cluster: %s\n",
    metadata.getClusterName());
for ( Host host : metadata.getAllHosts() ) {
    System.out.printf("Datacenter: %s; Host: %s; Rack: %s\n",
        host.getDatacenter(), host.getAddress(), host.getRack());
}
}

public void createSchema() {
    session.execute("DROP KEYSPACE IF EXISTS myKeySpaceTwo ;");

    session.execute("CREATE KEYSPACE myKeySpaceTwo WITH replication " +
        "= {'class':'SimpleStrategy', 'replication_factor':1};");

    session.execute(
        "CREATE TABLE myKeySpaceTwo.person (" +
        "id uuid PRIMARY KEY," +
        "first text," +
        "last text," +
        "city text," +
        "cell_1 text," +
        "cell_2 text," +
        "cell_3 text);");
}

public void loadData() {

    PreparedStatement statement = session.prepare(
        "INSERT INTO MyKeySpaceTwo.person (id, first, last, city, cell_1,
cell_2, cell_3) " +
        "VALUES (?, ?, ?, ?, ?, ?, ?);");

    BoundStatement boundStatement = new BoundStatement(statement);

    session.execute(boundStatement.bind(
        UUID.randomUUID(),
        "Ryan",

```

```

        "Ballenger",
        "Somerville",
        "6144064942",
        null,
        null
    ));

    session.execute(boundStatement.bind(
        UUID.randomUUID(),
        "George",
        "Washington",
        "DC",
        "6145556666",
        null,
        null
    ));

    session.execute(boundStatement.bind(
        UUID.randomUUID(),
        "Ice",
        "Cube",
        "Compton",
        "6141112222",
        null,
        null
    ));

}

public void querySchema(){
    ResultSet results = session.execute("SELECT * FROM
myKeySpaceTwo.person;"); //+
    //"WHERE id = 2cc9ccb7-6221-4ccb-8387-f22b6a1b354d;");
    System.out.println(String.format("%-10s | %-10s | %-10s | %-10s | %-10s |
%-10s | %-10s \n%s", "First", "Last", "City", "Cell #1", "Cell #2", "Cell #3", "ID",
    "-----"));
    for (Row row : results) {
        System.out.println(String.format("%-10s | %-10s | %-10s | %-10s |
%-10s | %-10s | %-10s |", row.getString("first"),
            row.getString("last"), row.getString("city"),
row.getString("cell_1"),row.getString("cell_2"),row.getString("cell_3"), row.getUUID("id")));
    }
}

```



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