Exercise 3.2 – Simple Statements

This exercise is designed to familiarize you with the basic code required to execute Simple Statements that read data from and write data to DSE.

In this exercise, you will complete the `addUser()` and `getUser()` functions of the `CassandraUserDAO` class object.

This exercise uses three (3) files in the *Session 2* exercise project:

- *CassandraUserDAO.java:* The file that defines the class through which the Killrvideo application reads and writes data to DSE
- *UserDAO.java:* This file defines a class that first simulates data being received from a webpage and then kicks off the process of writing that data to DSE. It does this by instantiating and initializing a `CassandraUserDAO` instance. It then fetches new user data from *TestData.java* and passes it to the `addUser()` function of the newly initialized `CassandraUserDAO` instance which in turn writes that data to DSE.
- *TestData.java:* A file that packages up new user data for several test users.

All steps will have most of the required code already written out. Your task is to place the additional code where it is needed and understand how to create and test the session. Here are the outlined steps.

STEPS:

Step 1: Examine the files: *TestData.java*, *CassandraUserDAO.java*, and *UserDAO.java*
Step 2: Complete the `addUser()` function
Step 3: Add two users and verify
Step 4: Add lightweight transaction to prevent adding the same user twice
Step 5: Complete the `getUser()` function

Step 1: Examine TestData.java, CassandraUserDAO.java and UserDAO.java

The purpose of this step is for you to become familiar with the running code, so you will understand *what* is happening and *why* it's happening.

1. Note the following CQL code for the *user* table and the *USER* java class:

```
CREATE TABLE user (
    email text PRIMARY KEY,
    addresses map<text,frozen<address>>,
    phone_numbers map<text, decimal>,
    joined timestamp,
    fname text,
    lname text,
    password blob,
    salt blob,
    user_id uuid
);

public class User {
    private String email;
    private String firstName;
    private String lastName;
    private Date joined;
    private ByteBuffer password;
    private ByteBuffer salt;
    private UUID userId;
    private Map<String, BigDecimal> phoneNumbers;
    private Map<String, Address> addresses;

    ...
```

2. If you have not already done so, launch the IDE (Che) and open the *session2 killrvideo* project.
3. Open and review the following three files:
    a. *~/session2/src/test/java/com/datastax/training/killrvideo/testutilities/TestData.java*
    b. *~/session2/src/main/java/com/datastax/training/killrvideo/model/dao/cassandra/CassandraUserDAO.java.*
    c. *~/session2/src/test/java/com/datastax/training/killrvideo/dao/cassandra/UserDAOTest.java*
4. Examine the file *TestData.java*. This file defines public class `TestData`, which has two public members of java class type `USER: TEST_USER1` and `TEST_USER2`. Both public members are initialized when an instance of `TestData` is instantiated. Take some time to examine the file so you are certain you understand how it functions.

5. Next, examine the file *CassandraUserDAO.java*. This file defines the class `CassandraUserDAO`, which is the class through which the Killrvideo application reads and writes data to DSE.

    a. Note that this file has a `getUser()` method and an `addUser()` method, both of which are incomplete. The `getUser()` method is missing some getter methods, and `addUser()` is missing the `SimpleStatement` syntax required to write a user to DSE.

6. Finally, examine the file *UserDAOTest.java*. This file defines `UserDAOTest`, which creates an instance of `CassandraUserDAO`, fetches a public user member from `TestData` and passes it to the `addUser()` method of `CassandraUserDAO`.

Step 2: Complete addUser() function

The purpose of this step is to add the syntax for the `SimpleStatement` that will complete this function. It requires instantiating a new `SimpleStatement` and supplying the CQL and the parameter values needed. The getters of the `User` object that is being passed into the `CassandraUserDAO` object will be used to provide those parameter values.

1. Within the IDE, navigate to the following file: *~/session2/src/main/java/com/datastax/training/killrvideo/model/dao/cassandra/CassandraUserDAO.java*
2. In the *CassandraUserDAO* file, locate the `addUser()` method.
3. Locate the section of method shown that appears as the following:

```
// TODO: You fill in this code


// TODO: Your added code ends here
```

4. Add the following simple statement syntax:

```
SimpleStatement statement = new SimpleStatement( , );
```

5. Continue to build onto this simple statement. The two parameter values required to initialize the `SimpleStatement()` object are the CQL statement and the parameter values. Start by adding the CQL Statement to the existing statement as follows:

```
SimpleStatement statement = new SimpleStatement(
    "INSERT INTO USER (email, joined, user_id, fname, lname, password,
salt, phone_numbers, addresses)" + "VALUES (?,?,?,?,?,?,?,?,?) IF NOT
EXISTS",
);
```

6. Note that the question marks indicate parameter values. These will need to be included after the comma that comes after the CQL Statement. Add `getter` statements following the previous inclusions:

```
SimpleStatement statement = new SimpleStatement(
    "INSERT INTO USER (email, joined, user_id, fname, lname, password,
salt, phone_numbers, addresses) " + "VALUES (?,?,?,?,?,?,?,?,?) IF NOT
EXISTS",
        newUser.getEmail(),
        newUser.getJoined(),
        newUser.getUserId(),
        newUser.getFirstName(),
        newUser.getLastName(),
        newUser.getPassword(),
        newUser.getSalt(),
        newUser.getPhoneNumbers(),
        newUser.getAddresses()
);
```

7. The CQL Statement is now complete and is ready to be executed. Insert the `session.execute()` statement after the `SimpleStatement` as shown below:

```
// TODO: You fill in this code

SimpleStatement statement = new SimpleStatement(
    "INSERT INTO USER (email, joined, user_id, fname, lname, password,
salt, phone_numbers, addresses) " + "VALUES (?,?,?,?,?,?,?,?,?) IF NOT
EXISTS",
        newUser.getEmail(),
        newUser.getJoined(),
        newUser.getUserId(),
        newUser.getFirstName(),
        newUser.getLastName(),
        newUser.getPassword(),
        newUser.getSalt(),
        newUser.getPhoneNumbers(),
        newUser.getAddresses()

);
    ResultSet result = session.execute(statement);

// TODO: Your added code ends here
```

8. The newly updated code is now ready. Do not run any code just yet; additional statements are still required.

Step 3: Add two users and verify

The purpose of this step is to ensure your code works and that you can successfully insert two (2) users into the *User* table.

1. Open the following file: *~/session2/src/test/java/com/datastax/training/killrvideo/dao/cassandra/UserDAOTest.java*
2. Find the following function:

```
public void testAddUser()
```

3. Verify that the code appears similar to what is shown below. In this portion of the code the following three (3) things should occur:
   a. A new `CassandraUserDAO` object and `User` object are instantiated.
   b. The `User` object is initialized with the `TEST_USER1` member of the `TestData` object.
   c. The `User` object is passed to the function we just completed so the `TEST_USER1` data gets submitted.

This is how the code should currently look in the file:

```
@Test
public void testAddUser() throws UserAlreadyExistsException {
    CassandraUserDAO userDAO = new CassandraUserDAO();
    User userToAdd = TestData.TEST_USER1;
    userDAO.addUser(userToAdd);

}
```

4. Now, examine the following file in the Che IDE: *~/session2/src/test/java/com/datastax/training/killrvideo/testutilities/TestData.java*.
5. Navigate to the section preceding `Test_User2`. Here we will insert the `TEST_USER1` data. The data to be inserted is shown as follows.

```
    TEST_USER1.setEmail("joeschmo@blah.com");
    TEST_USER1.setFirstName("Joseph");
    TEST_USER1.setLastName("Schmo");
    TEST_USER1.setJoined(new Date());

    TEST_USER1.setPassword(ByteBuffer.wrap("fake".getBytes()));
    TEST_USER1.setSalt(ByteBuffer.wrap("bake".getBytes()));
    TEST_USER1.setUserId(UUID.randomUUID());
    Map<String, BigDecimal> user1phones = new HashMap<>();
    user1phones.put("Home", new BigDecimal(123456789));
    user1phones.put("Mobile", new BigDecimal(2125551212));
    TEST_USER1.setPhoneNumbers(user1phones);
    TEST_USER1.setAddresses(new HashMap<>());
```

6. The file will auto save.
7. Run the code for the file *UserDAOTest.java*.
8. The code will now run and compile. If successful, you should see the following:
9. Open the Che terminal window and type `cqlsh <NodeIP>`
10. Once in the CQL shell, type `USE killrvideo_test;`
11. Type `SELECT * FROM user;`
12. There should be a single row of data as shown below.
13. Return to `testAddUser()` in the *UserDAOTest.java* file.
14. Change `TEST_USER1` to `TEST_USER2`.

```
User userToAdd = TestData.TEST_USER2;
```

15. File will auto save.
16. Re-run the file by going to `Run > Run 'UserDAOTest'`.
17. Switch to the terminal window and execute the following command within `cqlsh`:

```
SELECT * FROM user;
```

18. Confirm that there are now two rows of data for two separate users.

Step 4: Add lightweight transaction to prevent adding the same user twice

The purpose of this step is to ensure that a user, that has already been added, is not inadvertently overwritten.

You will now add a lightweight transaction that will check the `wasApplied` Boolean property of the `ResultSet` object. If it is false, it means the insert did not take place and that the original row in the table was not overwritten. The applications `UserAlreadyExistsException` will be raised and the error text indicated in the throw new statement will be returned to the calling application.

1. Navigate to the `addUser()` function of the `CassandraUserDAO` class.
2. Add the following code just below the `session.execute()` statement:

```
if (!result.wasApplied()) {
    throw new UserAlreadyExistsException(
        "Could not save user with the specified id. A duplicate already
exists")};
```

3. The code block should now appear as shown below:

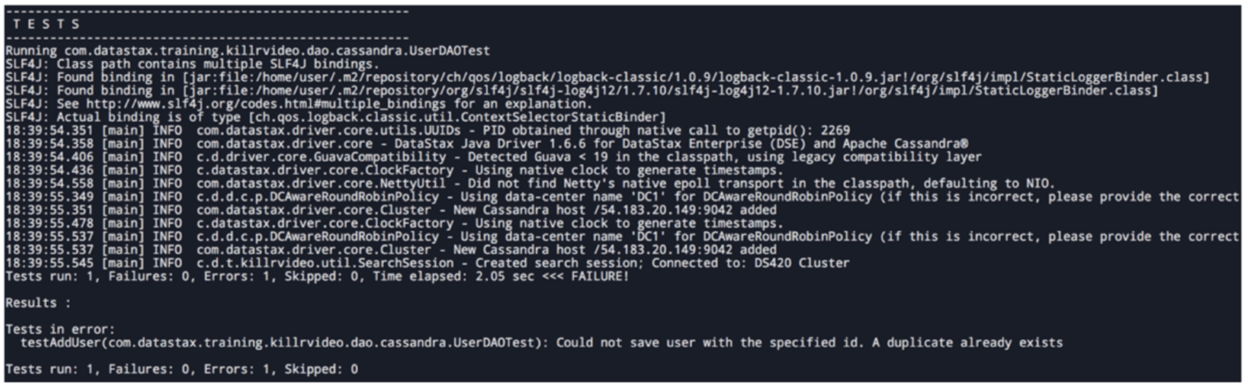
```
// TODO: You fill in this code

        SimpleStatement statement = new SimpleStatement(
                "INSERT INTO USER (email, joined, user_id, fname,
                lname, password, salt, phone_numbers, addresses) "
                        + "VALUES (?,?,?,?,?,?,?,?,?) IF NOT EXISTS",
                         newUser.getEmail(),
                         newUser.getJoined(),
                         newUser.getUserId(),
                         newUser.getFirstName(),
                         newUser.getLastName(),
                         newUser.getPassword(),
                         newUser.getSalt(),
                         newUser.getPhoneNumbers(),
                         newUser.getAddresses()
        );
        ResultSet result = session.execute(statement);

        if (!result.wasApplied()) {
            throw new UserAlreadyExistsException(
                "Could not save user with the specified id. A duplicate
already exists");
        }
```

4. Save the changes to the file.

5. Re-run the file *UserDAOTest.java* by going to `Run > Run 'UserDAOTest'`.
6. You should see an error as shown below.



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.datastax.training.killrvideo.dao.cassandra.UserDAOTest
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/user/.m2/repository/ch/qos/logback/logback-classic/1.0.9/logback-classic-1.0.9.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/user/.m2/repository/org/slf4j/slf4j-log4j12/1.7.10/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [ch.qos.logback.classic.util.ContextSelectorStaticBinder]
18:39:54.351 [main] INFO  com.datastax.driver.core.utils.UUIDs - PID obtained through native call to getpid(): 2269
18:39:54.358 [main] INFO  com.datastax.driver.core - DataStax Java Driver 1.6.6 for DataStax Enterprise (DSE) and Apache Cassandra®
18:39:54.406 [main] INFO  c.d.driver.core.GuavaCompatibility - Detected Guava < 19 in the classpath, using legacy compatibility layer
18:39:54.436 [main] INFO  c.datastax.driver.core.ClockFactory - Using native clock to generate timestamps.
18:39:54.558 [main] INFO  com.datastax.driver.core.NettyUtil - Did not find Netty's native epoll transport in the classpath, defaulting to NIO.
18:39:55.349 [main] INFO  c.d.d.c.p.DCAwareRoundRobinPolicy - Using data-center name 'DC1' for DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct
18:39:55.351 [main] INFO  com.datastax.driver.core.Cluster - New Cassandra host /54.183.20.149:9042 added
18:39:55.478 [main] INFO  c.datastax.driver.core.ClockFactory - Using native clock to generate timestamps.
18:39:55.537 [main] INFO  c.d.d.c.p.DCAwareRoundRobinPolicy - Using data-center name 'DC1' for DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct
18:39:55.537 [main] INFO  com.datastax.driver.core.Cluster - New Cassandra host /54.183.20.149:9042 added
18:39:55.545 [main] INFO  c.d.t.killrvideo.util.SearchSession - Created search session; Connected to: DS420 Cluster
Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 2.05 sec <<< FAILURE!

Results :

Tests in error:
  testAddUser(com.datastax.training.killrvideo.dao.cassandra.UserDAOTest): Could not save user with the specified id. A duplicate already exists

Tests run: 1, Failures: 0, Errors: 1, Skipped: 0
```

Step 5: Complete getUser() function

The purpose of this step is to write a Simple Statement and add the appropriate setters to the `getUser()` method.

1. Locate the `getUser()` method within the *CassandraUserDAO.java* file. Note that the `session`, `user`, and `row` objects, as well as the `return` statement, have been created for you.
2. Add the code below to instantiate a `SimpleStatement` object that executes a `SELECT` statement.

```
ResultSet resultSet = session.execute("SELECT * FROM user WHERE email =
'" + email + "'");
```

3. Note that the parameter variable email that was passed into `getUser()` is concatenated with the CQL statement as part of the syntax. That is because the primary key of the CQL statement is the field email. Add a light transaction immediately following that checks if the statement returns any data:

```
    if (resultSet.isExhausted()) {
        return null;
    }
```

4. The function will terminate and return null if no data was returned. Add the code that fetches the row in the event that data was returned.

```
    row = resultSet.one();
```

5. Next, add the setters that will update the appropriate properties on the `User` object that will be returned to calling function:

```
    newUser.setEmail(row.getString("email"));
    newUser.setFirstName(row.getString("fname"));
    newUser.setLastName(row.getString("lname"));
    newUser.setJoined(row.get("joined", Date.class));
    newUser.setPassword(row.getBytes("password"));
    newUser.setSalt(row.getBytes("salt"));
    newUser.setUserId(row.getUUID("user_id"));
    newUser.setPhoneNumbers(row.getMap("phone_numbers", String.class,
BigDecimal.class));
```

6. The contents of the function should now appear as follows.

```
public User getUser(String email) {
    DseSession session = getCassandraSession();

    User newUser = new User();
    Row row = null;

    // TODO: You fill in this code

    ResultSet resultSet = session.execute("SELECT * FROM user WHERE
email = '" + email + "'");

    if (resultSet.isExhausted()) {
        return null;
    }

    row = resultSet.one();
    newUser.setEmail(row.getString("email"));
    newUser.setFirstName(row.getString("fname"));
    newUser.setLastName(row.getString("lname"));
    newUser.setJoined(row.get("joined", Date.class));
    newUser.setPassword(row.getBytes("password"));
    newUser.setSalt(row.getBytes("salt"));
    newUser.setUserId(row.getUUID("user_id"));
    newUser.setPhoneNumbers(row.getMap("phone_numbers", String.class,
BigDecimal.class));
    // TODO: Your added code ends here

    return newUser;
}
```

7. Within Che, click on the *PLAY* icon on the navigation bar at the top of the IDE.
8. Select `Recompile 'CassandraUserDAO.java'`.
9. If the compilation is successful, your task is complete.
10. If the compilation does not succeed, review your work to identify the issue.

Now write your own code in the *UserDAOTest.java* file that calls the `getUser()` function. See below for a potential answer.

1. Add the following line at the end of the function:

```
User fetchUser = userDAO.getUser("joeschmo@blah.com");
```

2. Place a breakpoint on the closing bracket of the function and debug the function.
3. If the code runs without errors, you should be able to then view the object in the watch pane. Check the next page for a potential code solution:

```
    @Test
    public void testGetUser() {
        CassandraUserDAO userDAO = new CassandraUserDAO();
        User fetchUser = new User();
        fetchUser = userDAO.getUser("joeschmo@blah.com");

    }
```