Exercise 3.1 – Prepared Statements

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

In this exercise, you will complete the constructor that kicks off the insertion of a video as well as the `addVideo()` method that actually writes the video to DSE.

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

- *CassandraVideoDAO.java:* The file that defines the class responsible for reading and writing data to the video table.
- *PreparedStatementVideoDAOTest.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 `CassandraVideoDAO` instance. It then fetches new user data from AbstractVideoTest.java and passes it to the `addVideo()` function of the newly initialized `CassandraVideoDAO` instance which in turn writes that data to DSE.
- *AbstractVideoTest.java:* This file packages up a test video for this exercise.

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 a Prepared Statement. Here is a brief summary of the steps you will need to perform:

Step 1: Examine the files: *AbstractVideoTest.java*, *CassandraVideoDAO.java*, and *PreparedStatementVideoDAOTest.java*
Step 2: Complete the `CassandraVideoDAO` constructor
Step 3: Complete the `addVideo()` method
Step 4: Add a video and confirm it exists

Step 1: Examine AbstractVideoTest.java, CassandraVideoDAO.java and PreparedStatementVideoDAOTest.java

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

1. Note the CQL code for the *videos* table below:

```
// KillrVideo tables

// Entity table that will store many videos for a unique user
CREATE TABLE videos (
   video_id TIMEUUID,
   user_id UUID,
   title TEXT,
   description TEXT,
   type TEXT,
   url TEXT,
   release_date TIMESTAMP,
   release_year INT,
   avg_rating FLOAT,
   mpaa_rating TEXT,
   tags SET<TEXT>,
   preview_thumbnail BLOB,
   genres SET<TEXT>,
   PRIMARY KEY ((video_id))
);
```

2. If you have not already done so, launch the IDE (Theia) and open the *session2 killrvideo* project.

3. Open the following three files

   ~/session2/src/test/java/com/datastax/training/killrvideo/testutilities/AbstractVideoTest.java

   ~/session2/src/main/java/com/datastax/training/killrvideo/model/dao/cassandra/CassandraVideoDAO.java.

   ~/session2/src/test/java/com/datastax/training/killrvideo/dao/cassandra/PreparedStatementVideoDAOTest.java

4. First, examine the file *AbstractVideoTest.java*. This file defines public class `AbstractVideoTest` which offers a method called `createVideo()`. When called, this function returns a single video ("Pirates of the Caribbean") with values such as title, description, average rating, etc. This is the video we will insert into the video table.

5. Next, examine the file *CassandraVideoDAO.java*. This file defines `CassandraVideoDAO`, which is the class through which the Killrvideo application reads and writes video data to DSE. Note that it has an constructor called `CassandraVideoDAO` and an `addVideo()` method, both of which are incomplete. This constructor is missing the Prepared Statement syntax, and `addVideo()` is missing the syntax required to package up the data required for the prepared statement.

6. Finally, examine the file *PreparedStatementVideoDAOTest.java*. This file defines the `PreparedStatementVideoDAOTest` class and is responsible for creating an instance of `CassandraVideoDAO` and calling its `addVideo()` method. Note that its method `testAddVideo()` is already complete.

Step 2: Complete CassandraVideoDAO Constructor

The purpose of this step is to complete the code that will ensure the Prepared Statement is compiled and ready to go before the `addVideo()` function is called.

1. Switch to the *session2* project in the Theia IDE, if not already open.

2. In the IDE editor, navigate to the following file: *~/session2/src/main/java/com/datastax/training/killrvideo/model/dao/cassandra/CassandraVideoDAO.java*

3. Create a backup copy of this file. In a terminal window and within this working directory, run the following command:

```
cd
/home/ubuntu/session2/src/main/java/com/datastax/training/killrvideo/mo
del/dao/cassandra

cp CassandraVideoDAO.java CassandraVideoDAO.java.dist
```

4. Find the constructor `CassandraVideoDAO()` and view its current contents:

```
// Used to set the bucket value for the latest_videos table
private int currentDate;

public CassandraVideoDAO() {
    super();
    DseSession session = getCassandraSession();

// TODO: Insert your prepared statement here


// TODO: Your code ends here

selectByGenre = session.prepare("SELECT * FROM top_videos WHERE genre =
? AND release_year >= ? AND release_year < ? LIMIT ? ALLOW FILTERING");

        Date date = new Date();
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);
        currentDate = cal.get(Calendar.YEAR)*10000 +
       (cal.get(Calendar.MONTH)+1)*100 +
       cal.get(Calendar.DAY_OF_MONTH);
```

This method creates a session variable and sets the value of a private variable called “current data”. The `TODO` section is where you will place the prepared statement for inserting a new video. There is a prepared statement for an alternate method already present. The statement you insert will appear similar.

7. Uncomment the variable called `insertStatement` of type `PreparedStatement`. The variable is located just below the opening line of the class:

```
public class CassandraVideoDAO extends AbstractMapperDAO<Video> implements VideoDAO {

    private final PreparedStatement insertStatement;
    // private final PreparedStatement deleteStatement;
    // private final PreparedStatement deleteVideosByTagStatement;
    // private final PreparedStatement selectByTag;
    // private final PreparedStatement updateAvgRatingStatement;
    // private final PreparedStatement addTagStatement;
    // private final PreparedStatement removeTagStatement;
    // private final PreparedStatement updateVideoStatement;
    private final PreparedStatement selectByGenre;
```

   The `insertStatement` is a variable that is accessible to all of the methods in this class, including the constructor and the `addVideo()` method. It will be used in the next step.

5. Now, let's add the prepared statement. Insert the following code between the comments, replacing the existing lines of code:

```
// TODO: Insert your prepared statement here
    insertStatement = session.prepare(
        "INSERT INTO VIDEOS (video_id, user_id, title, type, " +
        "release_date, description, mpaa_rating, genres, tags, " +
        "preview_thumbnail, url, avg_rating) " +
        "VALUES (:video_id, :user_id, :title, :type, :release_date, " +
        ":description, :mpaa_rating, :genres, :tags, " +
        ":preview_thumbnail, :url, :avg_rating)"
    );
//TODO: Your code ends here
```

As you can see, a CQL statement is submitted to the `session.prepare()` method. Note that *release_year* is not included. This keeps unnecessary tombstones from getting created.

When an instance of `CassandraVideoDAO` is instantiated and initialized, this statement will be prepared ahead of time. That means that when the `addVideo()` method is called, the compilation stage will have already taken place. Therefore, if the user adds many videos, compilation will be skipped each time, thus making those transactions faster and reducing the workload on the database.

Also note that the CQL statement contains placeholders for values that will be submitted later when the `addVideo()` method is called, such as *video_id, user_id, title, type, release_date*, etc.

Step 3: Complete the addVideo() method

1. Scroll down and locate the mostly empty `addVideo()` function. Note the `newVideo` object that is passed in as a parameter value. It is an object that contains all the values needed to populate a row in the *Videos* table; we will need to add the code to pass those values to the prepared statement.

```
@Override
public void addVideo(Video newVideo) throws VideoAlreadyExistsException
{
    DseSession session = getCassandraSession();

//TODO: Insert your code here

//TODO: Your added code ends here

}
```

2. Add the code to make the method work. First, add the code below immediately following the first `TODO` block:

```
BoundStatement insertToVideos = insertStatement.bind();
```

   This code creates a *bound* statement and binds it to the *prepared* statement created in the previous step. As you will recall, the *prepared* statement has placeholders for parameter values in the CQL code. The *bound* statement will be used to provide those values at runtime.

3. Next, add the first parameter value. Insert the following line below the bound statement:

```
if (newVideo.hasVideoId()) {
    insertToVideos.setUUID("video_id", newVideo.getVideoId());
}
```

   This line sets the UUID value of the *bound* statement based on the `newVideo` object's *video_id* property. First, it checks to see if the value is on the `newVideo` object. If so, it uses the setter method on the `insertToVideos` object to set the *video_id* so it can be passed to the prepared statement.

4. Add the following code for the remaining parameters' values:

```
if (newVideo.hasUserId()) {
    insertToVideos.setUUID("user_id", newVideo.getUserId());
}

if (newVideo.hasTitle()) {
    insertToVideos.setString("title", newVideo.getTitle());
}

if (newVideo.hasType()) {
    insertToVideos.setString("type", newVideo.getType());
}

if (newVideo.hasReleaseDate()) {
    insertToVideos.setTimestamp("release_date",
newVideo.getReleaseDate());
}

if (newVideo.hasReleaseYear()) {
    insertToVideos.setInt("release_year", newVideo.getReleaseYear());
}

if (newVideo.hasDescription()) {
    insertToVideos.setString("description", newVideo.getDescription());
}

if (newVideo.hasMpaaRating()) {
    insertToVideos.setString("mpaa_rating", newVideo.getMpaaRating());
}

if (newVideo.hasGenres()) {
    insertToVideos.setSet("genres", newVideo.getGenres());
}

if (newVideo.hasTags()) {
    insertToVideos.setSet("tags", newVideo.getTags());
}

if (newVideo.hasPreviewThumbnail()) {
    insertToVideos.setBytes("preview_thumbnail",
newVideo.getPreviewThumbnail());
}

if (newVideo.hasUrl()) {
    insertToVideos.setString("url", newVideo.getUrl());
}

    insertToVideos.setFloat("avg_rating", newVideo.getAvgRating());
```

5. Add the code that will execute the Prepared Statement. Insert the following lines just below the previously added code:

```
// Execute the statement
ResultSet result = session.execute(insertToVideos);
```

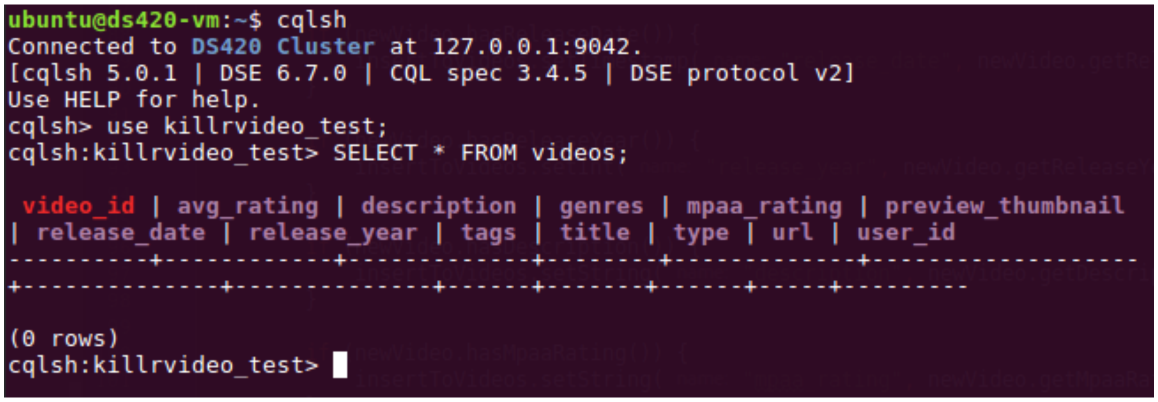
When done, the custom code in the file *CassandraVideoDAO.java* should appear as follows:

```
@Override
public void addVideo(Video newVideo) throws VideoAlreadyExistsException
{
    DseSession session = getCassandraSession();

    // Bucket value extracted from videoId, used for the latest_videos
table
    Calendar cal = Calendar.getInstance();
    cal.setTime(new Date(UUIDs.unixTimestamp(newVideo.getVideoId())));
    int bucket = cal.get(Calendar.YEAR)*10000 +
(cal.get(Calendar.MONTH)+1)*100 + cal.get(Calendar.DAY_OF_MONTH);

    // TODO: Insert your code here

    BoundStatement insertToVideos = insertStatement.bind();

    if (newVideo.hasVideoId()) {
        insertToVideos.setUUID("video_id", newVideo.getVideoId());
    }

    if (newVideo.hasUserId()) {
        insertToVideos.setUUID("user_id", newVideo.getUserId());
    }

    if (newVideo.hasTitle()) {
        insertToVideos.setString("title", newVideo.getTitle());
    }

    if (newVideo.hasType()) {
        insertToVideos.setString("type", newVideo.getType());
    }

    if (newVideo.hasReleaseDate()) {
        insertToVideos.setTimestamp("release_date",
newVideo.getReleaseDate());
    }

    if (newVideo.hasReleaseYear()) {
        insertToVideos.setInt("release_year",
newVideo.getReleaseYear());
    }
    if (newVideo.hasDescription()) {
        insertToVideos.setString("description",
newVideo.getDescription());
    }

    if (newVideo.hasMpaaRating()) {
        insertToVideos.setString("mpaa_rating",
newVideo.getMpaaRating());
    }

    if (newVideo.hasGenres()) {
        insertToVideos.setSet("genres", newVideo.getGenres());
    }

    if (newVideo.hasTags()) {
        insertToVideos.setSet("tags", newVideo.getTags());
    }

    if (newVideo.hasPreviewThumbnail()) {
        insertToVideos.setBytes("preview_thumbnail",
newVideo.getPreviewThumbnail());
    }

    if (newVideo.hasUrl()) {
        insertToVideos.setString("url", newVideo.getUrl());
    }

    insertToVideos.setFloat("avg_rating", newVideo.getAvgRating());

    // Execute the statement
    ResultSet result = session.execute(insertToVideos);

    // TODO: Your added code ends here

}
```

Step 4: Add a video and verify

We are now ready to run the code and insert a video. This step will first check that the *videos* table is empty. Then, run code and check the table again to confirm a row was written.

1. Return to the CQL shell and truncate the table: `TRUNCATE videos`. Then use `SELECT * FROM videos;` to verify that the videos table is indeed empty.



```
ubuntu@ds420-vm:~$ cqlsh
Connected to DS420 Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | DSE 6.7.0 | CQL spec 3.4.5 | DSE protocol v2]
Use HELP for help.
cqlsh> use killrvideo_test;
cqlsh:killrvideo_test> SELECT * FROM videos;

 video_id | avg_rating | description | genres | mpaa_rating | preview_thumbnail
| release_date | release_year | tags | title | type | url | user_id
----------+------------+-------------+--------+-------------+-------------------
+--------------+--------------+------+-------+------+-----+---------

(0 rows)
cqlsh:killrvideo_test>
```

2. Navigate to the following file: *~/session2/src/test/java/com/datastax/training/killrvideo/dao/cassandra/PreparedStatementVideoDAOTest.java*

3. Open the file. Note there are only a few lines of code:

```
Video originalVideo = createVideo();

CassandraVideoDAO videoDAO = new CassandraVideoDAO();
videoDAO.addVideo(originalVideo);
```

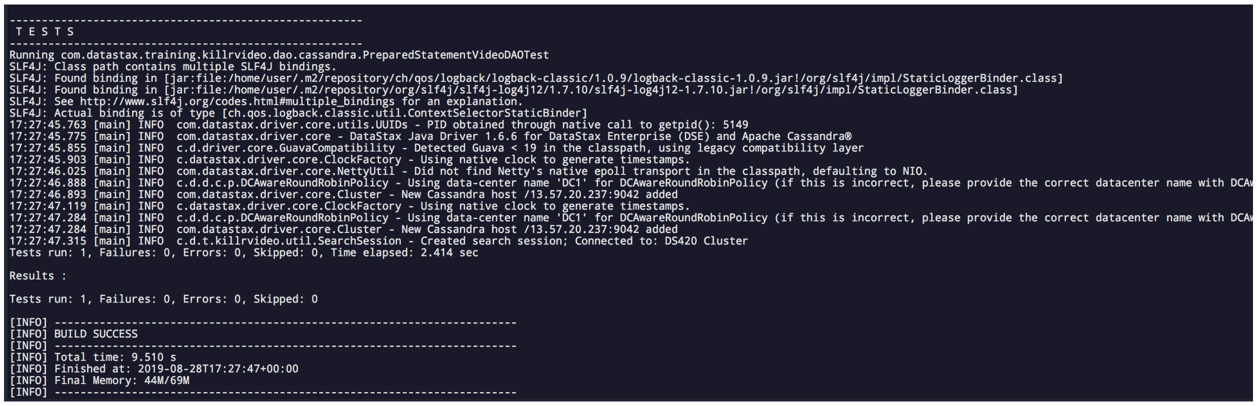
   a. First, `createVideo()` is called which populates an object called `originalVideo` with the “Pirates of the Caribbean” data we saw in the file *AbstractVideoTest.java*.

   b. Next, it instantiates a new `CassandraVideoDAO` object called `videoDAO`. This step creates and pre-compiles the prepared statement.

   c. Finally, it passes the `originalVideo` object with the Pirates of the Caribbean data to the `addVideo()` method just completed. The *prepared* statement is then executed with that data and it will be written to the table.

4. At the navigation bar at the top, select *Run>>’PreparedStatementVideoDAOTest’*.

5. Confirm the code compiled successfully:



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.datastax.training.killrvideo.dao.cassandra.PreparedStatementVideoDAOTest
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]
17:27:45.763 [main] INFO  com.datastax.driver.core.utils.UUIDs - PID obtained through native call to getpid(): 5149
17:27:45.775 [main] INFO  com.datastax.driver.core - DataStax Java Driver 1.6.6 for DataStax Enterprise (DSE) and Apache Cassandra®
17:27:45.855 [main] INFO  c.d.driver.core.GuavaCompatibility - Detected Guava < 19 in the classpath, using legacy compatibility layer
17:27:45.903 [main] INFO  c.datastax.driver.core.ClockFactory - Using native clock to generate timestamps.
17:27:46.025 [main] INFO  com.datastax.driver.core.NettyUtil - Did not find Netty's native epoll transport in the classpath, defaulting to NIO.
17:27:46.888 [main] INFO  c.d.d.c.p.DCAwareRoundRobinPolicy - Using data-center name 'DC1' for DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct datacenter name with DCAw
17:27:46.893 [main] INFO  com.datastax.driver.core.Cluster - New Cassandra host /13.57.20.237:9042 added
17:27:47.119 [main] INFO  c.datastax.driver.core.ClockFactory - Using native clock to generate timestamps.
17:27:47.284 [main] INFO  c.d.d.c.p.DCAwareRoundRobinPolicy - Using data-center name 'DC1' for DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct datacenter name with DCAw
17:27:47.284 [main] INFO  com.datastax.driver.core.Cluster - New Cassandra host /13.57.20.237:9042 added
17:27:47.315 [main] INFO  c.d.t.killrvideo.util.SearchSession - Created search session; Connected to: DS420 Cluster
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.414 sec

Results :

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

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.510 s
[INFO] Finished at: 2019-08-28T17:27:47+00:00
[INFO] Final Memory: 44M/69M
[INFO] ------------------------------------------------------------------------
```

6. Return to the CQL shell and execute the CQL statement again. There should now be a row in the videos table.