

## Session 4

---

<b>Setup .....</b>	<b>1</b>
<b>OVERVIEW .....</b>	<b>2</b>
<b>EXERCISE #1: Add a Users Table and DAO Code .....</b>	<b>3</b>
<b>EXERCISE #2: Implement the addUser Method .....</b>	<b>3</b>
<b>EXERCISE #3: Add a Set to the User Table to Store Playlists .....</b>	<b>3</b>
<b>EXERCISE #4: Insert Code to Add and Delete Playlists.....</b>	<b>4</b>
<b>EXERCISE #5: Running Queries with DevCenter .....</b>	<b>4</b>

### Setup

You can continue using the database from the previous session, but if you would like to start fresh, we've provided the script "playlist.cql" in the scripts directory.

Here are the directions to start from scratch: Wipe out everything you've done by dropping the keyspace. You can't be "using" the keyspace that you're dropping, so either use the system keyspace:

```
use system;
```

or restart cqlsh.

To drop the keyspace:

```
drop keyspace playlist;
```

To recreate everything, there is a script in the "scripts" directory of the project. You can run the script on Unix machines with the SOURCE command. For example, if you run it from the scripts directory:

```
source 'playlist.cql';
```

Double-check the paths in playlist.cql to the .csv files in the COPY commands, and ensure that the copy command will read in the csv files.

Load the application in Eclipse as you have done in the previous sessions, and run it accordingly.

## OVERVIEW

In this session we will be adding user accounts, and the ability for those users to add playlists. When you click “My playlists”, the system navigates to the PlaylistsServlet class. If your current session has logged on, then we query for the playlists for that user. If there is no user logged in, you are taken to the login screen. The screen is very simplistic. You may type a userid and password, and either create a new user or log the current user in.

 <p>The screenshot shows the 'Login Page' with a dark blue background. At the top, there are links for 'HOME' and 'PLAYLIST LOGIN'. Below the links are two input fields: 'Enter your email here' and 'Password'. At the bottom, there are two buttons: 'Login' and 'I Don't Have an Account'.</p> <p>Login Page</p>	 <p>The screenshot shows the 'Playlist Page' with a dark blue background. At the top, there are links for 'HOME' and 'LOGOUT'. Below the links, it says 'Playlists for john@doe.com'. There is a form to add a new playlist with the label 'New Playlist Name', an input field, and an 'Add' button. Below the form, there is a list of playlists with a minus sign '-' next to each item: 'AFTER 8', 'CALIFORNIA DREAMING', and 'WORKOUT MIX'.</p> <p>Playlist Page</p>
--	---

Once logged in, you are taken to the playlist page which shows a list of playlists for the given user. To add a playlist, simply type the name of the new playlist in the box, and click the “Add button”. To delete a playlist, click the ‘-’ next to it.

The User registration and verification is handled through the UserDao class and it has the following key methods – the accessors and constructors are not shown. Examine the code in the UserDao.java file.

Column Name	Type
<b>static</b> addUser(String, String): UserDao	Given a username and password, add a new user and return a UserDao object
<b>public</b> deleteUser(): void	Delete this user
<b>public</b> validateLogin (String, String): UserDao	If the user exists and the username is valid, return a UserDao, otherwise throw an exception
<b>public static</b> getUser (String username): UserDao	Return the UserDao for the given user name.

## EXERCISE #1: Add a Users Table and DAO Code

1. Add a simple users table called “users” to the database with the following columns below. Because we do lookups on this table by username, username should be the Primary Key.

Column Name	Type
username	text
password	text

## EXERCISE #2: Implement the addUser Method

2. It’s critical that the application only inserts each user once into the database, and the perfect way to accomplish this is by using a lightweight transaction. To do this we need to implement a couple of steps:
  - a. Insert a new user record into the user table. The fields to insert are username and password. Use the IF NOT EXISTS CLAUSE to ensure that the user is only inserted if it doesn’t already exist
  - b. The insert statement returns a ResultSet object which contains only 1 row. Retrieve the “[applied]” column.
  - c. If the change was not applied, throw a UserExistsException

## EXERCISE #3: Add a Set to the User Table to Store Playlists

3. Each user has a list of playlists, and playlist names must be unique for a given user. Let’s take advantage of collections in Cassandra to Model this. Use CQL shell to add a column called “playlist\_names” to the users table. Its type is a Set of Strings.

Column Name	Type
playlist_names	Set of <character data>

## EXERCISE #4: Insert Code to Add and Delete Playlists

4. The PlaylistDAO object in this session has 2 methods to implement: createPlaylist and deletePlaylist
5. Add the missing code to the createPlaylist Method. Because we're updating a Set object, bind variables are unsupported for the set manipulation itself. This means that we need to put the values directly in our CQL statement. As we have seen in a previous exercise, single quotes in a literal text value can be problematic.

We can, however, replace all single-quotes in the playlist name with a *pair* of single-quotes, so the playlist can be passed as a literal on the CQL statement. It's accomplished by this line in the code using the replace() method of the String class:

```
fixed_playlist_name = playlist_name.replace("'", "'");
```

You may, however, use a bind variable for any predicates on the where clause.

6. Add the missing code in the deletePlaylist Method as well. The code needs to remove the playlist from the set. Again, you must pass a literal for the set manipulation clauses in the CQL statement, but you may use bind variables for any "where" predicates.
7. Now that you should be able to register users and add and delete playlists, test out this functionality in the application.

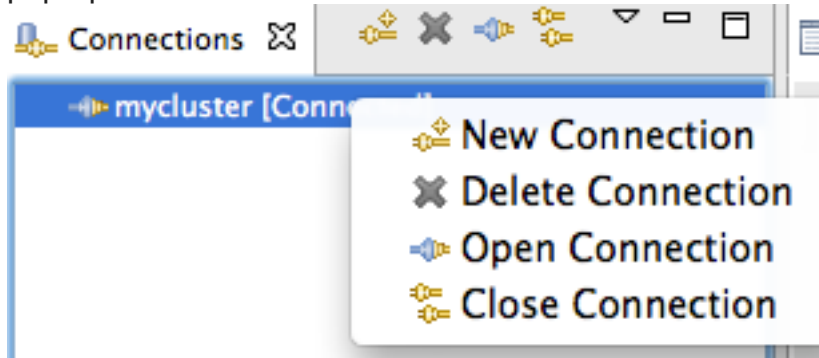
## EXERCISE #5: Running Queries with DevCenter

Now that you are familiar with CQL shell for running queries, lets run some queries with DevCenter.

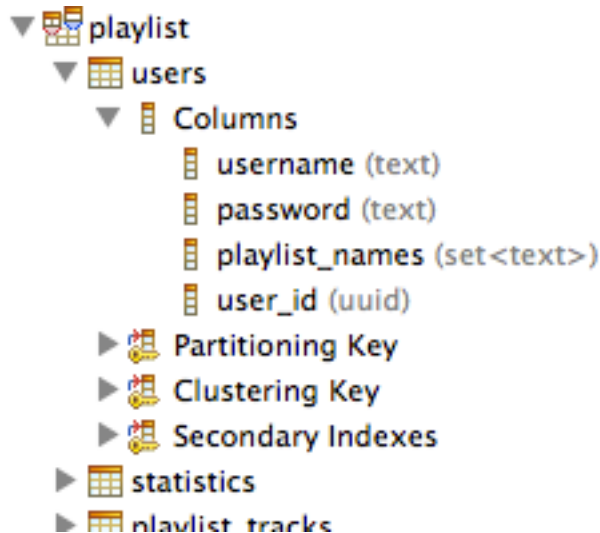
8. Download and install the DevCenter tool from <http://www.datastax.com/download/clientdrivers>
9. Start the DevCenter application
10. Create a connection to your Cassandra node running on localhost. To do this, click the connect icon at the top of the connections box.



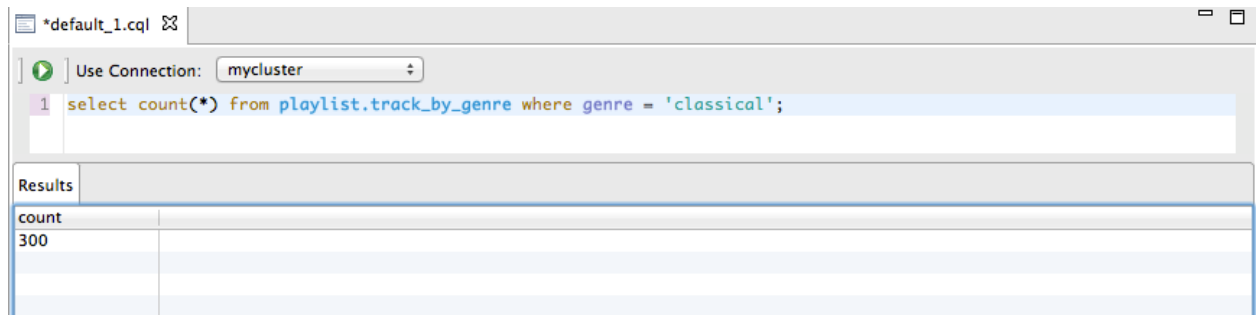
11. In the dialog, fill in a name you want to give this connection, and add the node "localhost" to the list of nodes, and click OK.
12. The application should automatically connect to your cluster. To connect or disconnect, simply click the plug icon, or right click the connection, and use the pop-up menu



13. Upon connecting, you should see your schema in the rightmost box. Choose the new connection in the "Use Connection" box at the top of the window. Explore the schema.



14. Now try some queries against your tables. This is an example of a count(\*) query. Notice that by default, DevCenter sets the limit to 300 by default, so although we have more than 300 classical tunes, DevCenter reports 300.



The screenshot shows a CQL client window titled '\*default\_1.cql'. It features a 'Use Connection:' dropdown menu set to 'mycluster'. Below the menu, a single CQL query is entered: `select count(*) from playlist.track_by_genre where genre = 'classical';`. The query is highlighted with a light blue background. Below the query editor, a 'Results' tab is active, displaying a table with two columns: 'count' and an empty cell. The value '300' is shown in the 'count' column.

count	
300	