Session 6

OVERVIEW

In this 6th and final session, we'll build out our final portion of the application, the statistics views, and in addition, use advanced features of Cassandra to optimize some existing pieces of the application.

EXERCISE #1: Using Counters to Collect Statistics

We've saved statistics for last as it's a component that touches nearly every other component in the system. The StatisticsDAO has 2 properties and 3 methods:

```
private final String counter_name;
private final long counter_value;

public static void increment_counter(String counter_name)
public static void decrement_counter(String counter_name)
public static List<StatisticsDAO> getStatistics()
```

To collect statistics, we simply place lines like this throughout the code:

```
 StatisticsDAO.increment_counter("failed login attempts");
```

This means that it's easy to define a new counter. The method getStatistics() will automatically return all of the collected statistics.

1. Add a statistics collection table called "statistics" to the database. Ensure to select the correct primary key.

Column Name	Type
counter_name	text
counter_value	counter

2. Implement the following methods in the StatisticsDAO class: increment_counter, and decrement_counter. These are static methods which will bump the counter_value column for the given counter name.

3. As you navigate the application, visit the "Statistics" page link off of the home page, and view your application statistics.

EXERCISE #2: Verify Logins by Reading at Quorum

4. In an earlier session we implemented code to read in user information. The consistency of user information is often more critical than other entities. You wouldn't want to cancel a user's account and then still be able to log in because they have read from a node that hasn't "heard" about the cancellation. By using Cassandra's tunable consistency, we can read the user data with a higher consistency level. By default, each request to Cassandra requires a response from only 1 replica of your data. On a query-by-query basis, you can choose to require a response from additional nodes, and then take the most recent version of a value.

5. We've added a new method to accomplish this in the UserDAO class:

```
UserDAO getUserWithQuorum(String username)
```

It's currently similar to the getUser() method. We've provided the CQL statement. Don't forget to decorate this statement and set then you can set its consistency level before you execute it.

EXERCISE #3: Using Paging to Handle Large Result Sets

When Fetching large result sets, Cassandra caches the entire result set and sends it to the application in a single block. By using the paging feature of the Java driver, the driver will automatically retrieve the result set in reasonable sized chunks.

6. Modify the listSongsByGenre method of the TracksDAO Class to retrieve the results 200 rows at a time. You need only add 1 line to the method.

EXERCISE #4: Improving Query Performance With a Row Cache

The Row cache can improve query performance on certain tables. Try the row cache on some of the tables in the music browser.

7. Using the nodetool info command, examine the "row cache" statistics, and notice that the row cache size is 0.

8. Set the row cache size by updating the cassandra.yaml file. In tarball installs, it can be found in the <tarball location>/conf directory. In package installs, you can find it in /etc/cassandra/cassandra.yaml. You may need to edit it with root permissions. For windows installs, find it in C:\Program Files\DataStax Community\apache-cassandra\conf. Increase the row cache size to 50M. Here is an example of the line you need to edit (after the edit):

```
# Maximum size of the row cache in memory.
# NOTE: if you reduce the size, you may not get
# you hottest keys loaded on startup.
#
# Default value is 0, to disable row caching.
row_cache_size_in_mb: 50
```

9. Enable the row cache on music browser tables by issuing an ALTER TABLE statement to enable the row cache.

10. Restart Cassandra

11. Issue several queries by clicking on the same records in the music browser. For example, if you enabled the row cache on the track_by_artist table, click on the same artist several times

12. Re-run the "nodetool info" command, and you should see your cache performance:

```
Rack                   : rack1
Exceptions             : 0
Key Cache              : size 97324 (bytes), capacity 104857600
(bytes), 67 hits, 82 requests, 0.817 recent hit rate, 14400
save period in seconds
Row Cache              : size 20253 (bytes), capacity 52428800
(bytes), 1 hits, 2 requests, 0.500 recent hit rate, 0 save
period in seconds
```

EXERCISE #5: (BONUS) Use a Static Variable for the Prepared Statement

13. One of the key benefits of using a PreparedStatement object is that we can prepare a statement once, and reuse it every time. You have already added a prepared statement to the method TrackDAO.listSongsByArtist() in **Error! Reference source not found.**

14. Use a Static class variable to store the prepared statement. Ensure that you only ever prepare the statement once. Look at the way we used a static variable in the CassandaData class for some clues as to how to do this. The PreparedStatement class is also thread-safe, which means several concurrent web requests can share the same prepared statement object.