Exercise 4.4 – Sets

This exercise is a continuation of the previous exercise and is designed to familiarize users with Set queries.

In this exercise, you will:

- Learn to execute writes using Sets
- Run updates on Sets
- Be able to perform reads on Sets
- Understand how to use Sets using CQL

Part 1: The writeSet() Function in IDE

This next exercise will explore writing `Sets` to the database. Locate the `writeSet()` function in the *UserDAOCassandraTest.java* file. Copy and paste the following portion of code into that function:

Before you get started, run the following CQL statement to create the table you will be using:

```
CREATE TABLE userset (userid int PRIMARY KEY, things set<text>);
```

Step 1: Instantiate a new session and then create a list strings and have them equal hashed `Sets`.

```
CassandraUserDAO userDAO = new CassandraUserDAO();
DseSession session = CassandraSession.getSession();
Set<String> s0 = new HashSet<>();
Set<String> s1 = new HashSet<>();
Set<String> s2 = new HashSet<>();
Set<String> s3 = new HashSet<>();
```

Step 2: Assign item of furniture to each set. Note there can be multiple items per `Set`. Add additional `Sets` for an added challenge.

```
s0.add("end table");
s1.add("table");
s2.add("table"); s2.add("chair");
s3.add("table"); s3.add("chair"); s3.add("lid");
```

Step 3: Create a new `BatchStatement`. This is a single line statement.

```
BatchStatement batch = new BatchStatement();
```

Step 4: Using the `INSERT` command, add the items into the *userset* table.

```
Insert is0 = QueryBuilder.insertInto("userset").value("userid",
   0).value("things", s0);
Insert is1 = QueryBuilder.insertInto("userset").value("userid",
   1).value("things", s1);
Insert is2 = QueryBuilder.insertInto("userset").value("userid",
   2).value("things", s2);
Insert is3 = QueryBuilder.insertInto("userset").value("userid",
   3).value("things", s3);
```

Step 5: Add each item into a batch set.

```
batch.add(is0);
batch.add(is1);
batch.add(is2);
batch.add(is3);
```

Step 6: Execute the function.

```
ResultSet result = session.execute(batch);
```

1. Run the command within the IDE by right-clicking on the icon to the left of the function and select *Debug 'writeSet()'*.

2. Did the command execute correctly? What happened? How do you fix this?

3. After correcting the issue, re-run the IDE `writeSet()` function. Confirm the items were added to the correct table by performing the following command:

Part 2: The updateSet() Function in IDE

This next segment of `Sets` is to update them in place. Locate the `updateSet()` function in the *UserDAOCassandraTest.java* file. Copy and paste the following segments of code into this function:

Step 1: Create a set of strings and assign them to hashed sets.

```
CassandraUserDAO userDAO = new CassandraUserDAO();
DseSession session = CassandraSession.getSession();
Set<String> s2 = new HashSet<>();
Set<String> s3 = new HashSet<>();
Set<String> s4 = new HashSet<>();
```

Step 2: Instantiate items to each set.

```
s2.add("table");
s3.add("box");
s4.add("sofa");
s4.add("lamp");
```

Step 3: Create a new `BatchStatement`.

```
BatchStatement batch = new BatchStatement();
```

Step 4: Update each set by either adding, removing, or changing existing items in place.

```
// add an item to a set
BuiltStatement bs1 =
   QueryBuilder.update("userset").with(QueryBuilder.append("things",
   "chair")).where(QueryBuilder.eq("userid",1));

// remove an item from a set
BuiltStatement bs2 =
   QueryBuilder.update("userset").with(QueryBuilder.removeAll("things",
   s2)).where(QueryBuilder.eq("userid",2));

// replace set
BuiltStatement bs3 =
   QueryBuilder.update("userset").with(QueryBuilder.set("things",s3)).w
   here(QueryBuilder.eq("userid",3));

// add a new row to the table
Insert is4 = QueryBuilder.insertInto("userset").value("userid",
   4).value("things", s4);

// update an item in a set
BuiltStatement bs5 =
   QueryBuilder.update("userset").with(QueryBuilder.remove("things",
```

```
   "table")).and(QueryBuilder.add("things","lava
   lamp")).where((QueryBuilder.eq("userid",1)));

// delete a row from the table
BuiltStatement bs6 =
   QueryBuilder.delete().from("userset").where((QueryBuilder.eq("userid
   ",0)));
```

Step 5: Add each string to a new batch set.

```
batch.add(bs1);
batch.add(bs2);
batch.add(bs3);
batch.add(is4);
batch.add(bs5);
batch.add(bs6);
```

Step 6: Execute the result. Check the outcome and confirm everything worked as intended.

```
ResultSet result = session.execute(batch);
```

1. Using the CQLSH terminal, query the *userset* table within the *killrvideo_test* keyspace and confirm items were updated in place.

2. Confirm the output is similar to the following screenshot:

Part 3: The readSet() Function in IDE

The final segment of `Sets` is to run a `read()` function to confirm the code executed correctly. The result should be displayed within the IDE console.

1. Locate the `readSet()` function in the *UserDAOCassandraTest.java* file. Copy and paste the following code directly into the TODO block of this function:

```
CassandraUserDAO userDAO = new CassandraUserDAO();
DseSession session = CassandraSession.getSession();
Set<String> userstuff = new HashSet<>();
Integer userid;
```

2. Now enter the code that will read from the table.

```
        ResultSet result = session.execute("SELECT * FROM userset");
```

3. Write the code that will extract the values in the set of each row. Enter the for-next loop below:

```
        for (Row row : result) {

            }
```

4. Within the for-next loop, add the following two lines. These lines will extract the *userid* and the set from the row:

```
            userid = row.get("userid", Integer.class);
            userstuff = row.getSet("things", String.class);
```

5. Add this for-next loop just below the last two lines you entered. It will iterate through each item in a row's set and print it out along with the *userid*.

```
            for (String stuff: userstuff)
            {
                System.out.println(userid + " : " + stuff);
            }
```

6. Your code should now look as shown below:

```
        Set<String> userstuff = new HashSet<>();
        Integer userid;
        ResultSet result = session.execute("SELECT * FROM userset");

        for (Row row : result) {
            userid = row.get("userid", Integer.class);
            userstuff = row.getSet("things", String.class);
            for (String stuff: userstuff)
            {
                System.out.println(userid + " : " + stuff);
            }
```

7. Run the function. It should show the following output in the console:

```
1 : chair
1 : lava lamp
2 : chair
4 : lamp
4 : sofa
3 : box

Process finished with exit code 0
```

OPTIONAL STEP: CQL

The purpose of the following step is for you to familiarize yourself with the CQL syntax required to execute the writes, updates and reads we executed.

1. Truncate the table *userset*.

2. Run the following statements. Notice the syntax is the same as for lists except that curly braces are used instead of square brackets.

```
INSERT INTO userset (userid, things) VALUES (1, {'table'});
INSERT INTO userset (userid, things) VALUES (0, {'end table'});
INSERT INTO userset (userid, things) VALUES (2, {'table','chair'});
INSERT INTO userset (userid, things) VALUES (3,
   {'table','chair','lid'});
```

3. Run a *select* and compare the results to what you inserted.

4. Now run the statements below to execute the updates.

```
UPDATE userset SET things = things + {'chair'} WHERE userid = 1;
UPDATE userset SET things = things - {'table'} WHERE userid = 2;
UPDATE userset SET things = {'box'} WHERE userid = 3;
INSERT INTO userset (userid, things) VALUES (4, {'sofa','lamp'});
UPDATE userset SET things = things - {'table'}, things = things +
   {'lava lamp'} WHERE userid = 1;
DELETE FROM userset WHERE userid = 0;
```

5. Confirm the items have now been assigned the proper *userid* and are in the correct order by running a *select* statement:

```
SELECT * FROM userset;
```