Exercise 4.5 – Maps

This exercise is a continuation of the earlier exercise and is designed to familiarize users with Map queries.

In this exercise, you will:

- Understand how to perform writes with Maps
- Learn to execute updates to existing Maps
- Be able to run reads on Maps
- Learn how to use Maps in both the IDE and in CQL

Part 1: The writeMap() function in IDE

This section of the exercise will examine the `writeMap()` function. A map relates one item to another with a key-value pair. For each key, only one value may exist, and duplicates cannot be stored. Both the key and the value are designated with a data type.

Before you get started, run the following CQL statement:

```
CREATE TABLE usermap (userid int primary key, things map<text, text>);
```

Step 1: Create a new session, map strings to new hash maps.

```
CassandraUserDAO userDAO = new CassandraUserDAO();
DseSession session = CassandraSession.getSession();
Map<String, String> map0 = new HashMap<>();
Map<String, String> map1 = new HashMap<>();
Map<String, String> map2 = new HashMap<>();
Map<String, String> map3 = new HashMap<>();
```

Step 2: Assign item(s) to each separate map.

```
map0.put("1", "end table");
map1.put("1", "table");
map2.put("1", "table"); map2.put("2", "chair");
map3.put("1", "table"); map3.put("2", "chair"); map3.put("3", "lid");
```

Step 3: Create a new batch statement and run a prepared statement to insert data into the *usermap* table.

```
BatchStatement batch = new BatchStatement();
PreparedStatement ps = session.prepare("INSERT INTO usermap (userid,
   things) VALUES (:userid, :things)");
```

Step 4: Bind the statements to maps.

```
BoundStatement bs0 = ps.bind(0, map0);
BoundStatement bs1 = ps.bind(1, map1);
BoundStatement bs2 = ps.bind(2, map2);
BoundStatement bs3 = ps.bind(3, map3);
```

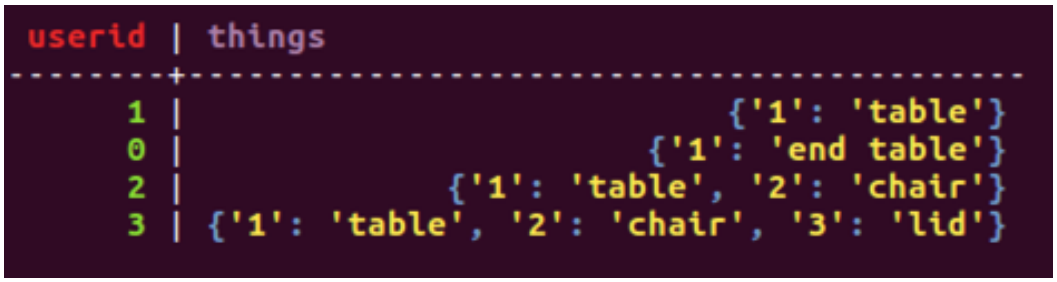
Step 5: Batch the statements.

```
batch.add(bs0);
batch.add(bs1);
batch.add(bs2);
batch.add(bs3);
```

Step 6: Execute the resulting set.

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

1. Confirm the maps have been added to the *userset* table.



Part 3: The updateMap() Function in IDE

This next portion of this exercise examines the `updateMap()` function. You will write code that will update the map data you inserted previously.

Step 1: Create a new session and map or set variables.

```
CassandraUserDAO userDAO = new CassandraUserDAO();
DseSession session = CassandraSession.getSession();

Map<String, String> map1 = new HashMap<>();
Set<String> Set2 = new HashSet<>();
Map<String, String> map3 = new HashMap<>();
Map<String, String> map4 = new HashMap<>();
Map<String, String> map6 = new HashMap<>();
Integer x = 0;
```

Step 2: Add items to the maps/sets.

```
map1.put("2", "chair");
Set2.add("1");
map3.put("1", "box");
map4.put("1", "sofa"); map4.put("2", "lamp");
map6.put("3", "lava lamp");
```

Step 4: Create a new `BatchStatement` and then add additional `PreparedStatements` that will be used by the `QueryBuilder` statements we'll be writing to update the data in the table.

```
BatchStatement batch = new BatchStatement();
PreparedStatement ps1 = session.prepare("UPDATE usermap SET things =
   things + :things WHERE userid = :userid");
PreparedStatement ps2 = session.prepare("UPDATE usermap SET things =
   things - :idx WHERE userid = :userid");
```

Step 5: Next, add items to the map using `BuiltStatements`. Note that set2 is used to pass a single value to the QueryBuilder statement for the BuiltStatement built2. This is because the QueryBuilder statement is going to remove the map instead of adding to a map (which would require we provide a map instead of a set).

```
// add an item to the map
BuiltStatement built1 =
   QueryBuilder.update("usermap").with(QueryBuilder.put("things", "2",
   "chair"))
        .where(QueryBuilder.eq("userid",1));

// remove an item from the map
BuiltStatement built2 =
   QueryBuilder.update("usermap").with(QueryBuilder.removeAll("things",
   Set2))
```

```
        .where(QueryBuilder.eq("userid",2));

// replace a map
BuiltStatement built3 =
   QueryBuilder.update("usermap").with(QueryBuilder.set("things",map3))
        .where(QueryBuilder.eq("userid",3));

// add a new row to the table
Insert insert4 = QueryBuilder.insertInto("usermap")
        .value("userid", 4)
        .value("things", map4);

// update an item in a map
x = 1;
BuiltStatement built5 =
   QueryBuilder.update("usermap").with(QueryBuilder.set("things['" + x
   + "']", "lava lamp"))
        .where(QueryBuilder.eq("userid", 1));

// delete a row from table
BuiltStatement built6 = QueryBuilder.delete().from("usermap")
        .where((QueryBuilder.eq("userid",0)));
```

Step 6: Add a new batch list.

```
batch.add(built1);
batch.add(built2);
batch.add(built3);
batch.add(insert4);
batch.add(built5);
batch.add(built6);
```

Step 7: Execute the resulting set by right-clicking on the `updateMap()` function in IDE and running *Debug 'updateMap()'*.

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

1. Confirm the final results are the same as that shown in the previous exercise. Pull up the CQLSH terminal and view the contents of the *usermap* table.

```
 userid | things
--------+-----------------------------------
      1 | {'1': 'lava lamp', '2': 'chair'}
      2 |                    {'2': 'chair'}
      4 |        {'1': 'sofa', '2': 'lamp'}
      3 |                     {'1': 'box'}
```

Part 3: The readMap() Function in IDE

This final segment of the exercise will perform a `readMap()` function. This will help confirm that the code executed correctly by displaying the result to the IDE console.

1. Locate the `readMap()` function in the *UserDAOCassandraTest.java* file. Copy and paste the following code directly into the TODO block of this function. This code will dimension the map you will read from as well as fetch the *usermap* table's data from the database:

```
CassandraUserDAO userDAO = new CassandraUserDAO();
DseSession session = CassandraSession.getSession();
Map<String, String> userstuff = new HashMap<>();
        Integer userid;
       ResultSet result = session.execute("SELECT * FROM usermap");
```

2. Next, create a for-next loop. We will use this to iterate through each row in the result set.

```
        for (Row row : result) {

        }
```

3. Next, add this code inside the for-next loop. It fetches the *userid* and its associated map from each row.

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

4. Next add this code just below the code you just added. It will create an iterator and a while loop. Inside that loop, the iterator fetches a key-value pair and prints it to the console along with its associated *userid*.

```
            Iterator it = userstuff.entrySet().iterator();
            while (it.hasNext()) {
                Map.Entry pair = (Map.Entry) it.next();
                System.out.println(userid + " : "
                        + pair.getKey() + " = "
                        + pair.getValue());
                it.remove();
            }
```

5. Now review your code. It should look as follows:

```
        //TODO: Your code starts here
        CassandraUserDAO userDAO = new CassandraUserDAO();
        DseSession session = CassandraSession.getSession();
        Map<String, String> userstuff = new HashMap<>();
        Integer userid;
     ResultSet result = session.execute("SELECT * FROM usermap");

        for (Row row : result) {
            userid = row.get("userid", Integer.class);
            userstuff = row.getMap("things", String.class,
   String.class);

            Iterator it = userstuff.entrySet().iterator();
            while (it.hasNext()) {
                Map.Entry pair = (Map.Entry) it.next();
                System.out.println(userid + " : "
                        + pair.getKey() + " = "
                        + pair.getValue());
                it.remove();
            }
        }
        //TODO: Your code ends here
```

6. Now run the function. It should show the following output in the console:

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

Process finished with exit code 0
```

Optional Step: CQLSH Statements

The purpose of this step is to familiarize you with the CQL syntax required to do the inserts, updates, and reads we just executed via code. Truncate the *usermap* table and execute the following statements:

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

Confirm the values were entered correctly by listing the *usermap* table.

```
[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> CREATE TABLE usermap (userid int PRIMARY KEY, things map<
text, text>);
cqlsh:killrvideo_test> INSERT INTO usermap (userid, things) VALUES (1, {'1':'tab
le'});
cqlsh:killrvideo_test> INSERT INTO usermap (userid, things) VALUES (0, {'1':'end
 table'});
cqlsh:killrvideo_test> INSERT INTO usermap (userid, things) VALUES (2, {'1':'tab
le', '2':'chair'});
cqlsh:killrvideo_test> INSERT INTO usermap (userid, things) VALUES (3, {'1':'tab
le', '2':'chair','3': 'lid'});
cqlsh:killrvideo_test> SELECT * FROM usermap;

 userid | things
--------+----------------------------------------
      1 |                          {'1': 'table'}
      0 |                      {'1': 'end table'}
      2 |             {'1': 'table', '2': 'chair'}
      3 | {'1': 'table', '2': 'chair', '3': 'lid'}

(4 rows)
cqlsh:killrvideo_test>
```

Execute the following update statements:

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

Confirm the values have been updated by listing the contents of the *usermap* table. The results should appear similar to the following:

```
cqlsh:killrvideo_test> SELECT * FROM usermap;

 userid | things
--------+-----------------------------------
      1 | {'1': 'lava lamp', '2': 'chair'}
      2 |                    {'2': 'chair'}
      4 |       {'1': 'sofa', '2': 'lamp'}
      3 |                      {'1': 'box'}

(4 rows)
cqlsh:killrvideo_test>
```