Exercise 11

Further Cassandra

Prior Knowledge
Unix Command Line Shell
Cassandra exercise

Learning Objectives
Better understand Cassandra's CQL shell and CQL
Understand limitations of Cassandra compared with SQL
Understand JSON support and non-traditional data-types

Software Requirements
(see separate document for installation of these)

- Apache Cassandra

1. Make sure Cassandra is running
   a. In a Terminal window (Crtl-Alt-T) type:

      ```
      service cassandra status
      ```

   b. You should see

      ```
      * Cassandra is running
      ```

   c. If not, try

      ```
      sudo service cassandra start
      ```

      and then check the status again.

2. Now you can start the Cassandra Shell:
   Type:
   `cqlsh`

   You should see:

```
/home/oxclo/.local/bin/cqlsh:470: DeprecationWarning: Legacy execution
      parameters will be removed in 4.0. Consider using execution
      profiles.
/home/oxclo/.local/bin/cqlsh:500: DeprecationWarning: Setting the
      consistency level at the session level will be removed in 4.0.
      Consider using execution profiles and setting the desired
      consitency level to the EXEC_PROFILE_DEFAULT profile.
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 4.0 | CQL spec 3.4.5 | Native protocol v4]
Use HELP for help.
cqlsh>
```


3. First, let's try some queries on the data.

```
use wind;
```

4. Try:

```
select * from winddata where time = '2015-01-01' and
stationid = 'SF36';
```

You should see:

```
 stationid | time                      | direction | temp  | velocity
-----------+---------------------------+-----------+-------+----------
      SF36 | 2015-01-01 00:00:00+0000 |     116.9 | 11.33 |    2.727
```

5. Now try

```
select * from winddata where time <= '2015-01-02'
and stationid = 'SF36' limit 20;
```

All normal:

```
 stationid | time                      | direction | temp  | velocity
-----------+---------------------------+-----------+-------+----------
      SF36 | 2015-01-01 00:00:00+0000 |     116.9 | 11.33 |    2.727
      SF36 | 2015-01-01 00:05:00+0000 |     108.5 | 11.25 |    1.814
      SF36 | 2015-01-01 00:10:00+0000 |     113.7 |  11.2 |    2.621
      SF36 | 2015-01-01 00:15:00+0000 |     117.8 | 11.11 |    3.678
      SF36 | 2015-01-01 00:20:00+0000 |     117.3 | 11.07 |    2.842
      SF36 | 2015-01-01 00:25:00+0000 |     117.3 | 11.07 |    2.629
      SF36 | 2015-01-01 00:30:00+0000 |     117.3 | 11.09 |    2.235
      SF36 | 2015-01-01 00:35:00+0000 |     117.2 | 11.09 |    2.043
      SF36 | 2015-01-01 00:40:00+0000 |     117.2 | 11.05 |    1.635
      SF36 | 2015-01-01 00:45:00+0000 |     117.3 | 10.93 |    2.224
      SF36 | 2015-01-01 00:50:00+0000 |     112.5 | 10.86 |    1.822
      SF36 | 2015-01-01 00:55:00+0000 |     108.7 |  10.8 |    0.866
      SF36 | 2015-01-01 01:00:00+0000 |     108.7 | 10.67 |    1.068
      SF36 | 2015-01-01 01:05:00+0000 |     108.6 | 10.54 |    1.393
      SF36 | 2015-01-01 01:10:00+0000 |     108.7 | 10.44 |    1.468
      SF36 | 2015-01-01 01:15:00+0000 |     108.9 | 10.37 |    1.859
      SF36 | 2015-01-01 01:20:00+0000 |     108.6 | 10.29 |     1.67
      SF36 | 2015-01-01 01:25:00+0000 |     108.6 | 10.25 |    1.241
      SF36 | 2015-01-01 01:30:00+0000 |     108.5 | 10.21 |    0.675
      SF36 | 2015-01-01 01:35:00+0000 |     108.4 | 10.26 |    0.623

(20 rows)
```


6. Now another:

```
select * from winddata where time <= '2015-01-01 01:00:00' and
stationid in ('SF37', 'SF36');
```

stationid	time	direction	temp	velocity
SF36	2015-01-01 00:00:00+0000	116.9	11.33	2.727
SF36	2015-01-01 00:05:00+0000	108.5	11.25	1.814
SF36	2015-01-01 00:10:00+0000	113.7	11.2	2.621
SF36	2015-01-01 00:15:00+0000	117.8	11.11	3.678
SF36	2015-01-01 00:20:00+0000	117.3	11.07	2.842
SF36	2015-01-01 00:25:00+0000	117.3	11.07	2.629
SF36	2015-01-01 00:30:00+0000	117.3	11.09	2.235
SF36	2015-01-01 00:35:00+0000	117.2	11.09	2.043
SF36	2015-01-01 00:40:00+0000	117.2	11.05	1.635
SF36	2015-01-01 00:45:00+0000	117.3	10.93	2.224
SF36	2015-01-01 00:50:00+0000	112.5	10.86	1.822
SF36	2015-01-01 00:55:00+0000	108.7	10.8	0.866
SF36	2015-01-01 01:00:00+0000	108.7	10.67	1.068
SF37	2015-01-01 00:00:00+0000	252.3	11.11	3.774
SF37	2015-01-01 00:05:00+0000	273.89999	10.75	2.69
SF37	2015-01-01 00:10:00+0000	299.79999	11.1	1.747
SF37	2015-01-01 00:15:00+0000	303.5	11.65	1.534
SF37	2015-01-01 00:20:00+0000	282.79999	10.27	2.269
SF37	2015-01-01 00:25:00+0000	281.70001	9.72	2.141
SF37	2015-01-01 00:30:00+0000	292.70001	9.78	1.054
SF37	2015-01-01 00:35:00+0000	280.39999	9.53	2.36
SF37	2015-01-01 00:40:00+0000	280.29999	9.3	2.155
SF37	2015-01-01 00:45:00+0000	266.10001	9.37	3.1
SF37	2015-01-01 00:50:00+0000	272	9.46	2.703
SF37	2015-01-01 00:55:00+0000	265.39999	9.54	3.026
SF37	2015-01-01 01:00:00+0000	291.60001	9.7	1.508

(26 rows)

7. So we can query normally can we? Let's try something else:

```
select * from winddata where time <= '2015-01-01 01:00:00';
```

Uh oh!

```
InvalidRequest: code=2200 [Invalid query] message="Cannot
execute this query as it might involve data filtering and
thus may have unpredictable performance. If you want to
execute this query despite the performance
unpredictability, use ALLOW FILTERING"
```

Basically, Cassandra will not do unbounded time queries, unless you force it to!

8. Try again, but this time explicitly enabling this query.

```
select * from winddata where time <= '2015-01-01
01:00:00' allow filtering;
```


9. Now let's try another query:

```
select * from winddata where time <= '2015-01-01
01:00:00' and temp < 10 ;
```

Again this fails. Unlike a normal SQL database, you cannot do arbitrary queries on Cassandra. You must limit your queries to those that can be done based on the primary key. There are ways of creating secondary indices, but these basically create a whole new table under the covers to allow efficient searching.

10. We have now come across some limitations of Cassandra. Let's look at the extra stuff you can do.

First let's try some JSON support. Try the following:

```
CREATE KEYSPACE jsontest WITH REPLICATION = { 'class' :
'SimpleStrategy', 'replication_factor' : 1 };

use jsontest;
create table users  (id text primary key, name text, age int ,
job text);
insert into users (id, name, age, job) values ('1', 'Paul', 46,
'Student') ;
select json * from users;
```

You should see:

```
[json]
------------------------------------------------------------
 {"id": "1", "age": 46, "job": "Student", "name": "Paul"}

(1 rows)
```

11. Now let's insert data using JSON.
Notice how we can use either JSON or not and they interoperate

```
insert into users json ' {"id": "2", "age": 43,
"job": "Teacher", "name": "Henry"} ';

select * from users;

id | age | job     | name
----+-----+---------+-------
  2 |  43 | Teacher | Henry
  1 |  46 | Student |  Paul

(2 rows)
```


12. Of course, JSON supports complex types including lists, maps, sets and other data. Luckily Cassandra does too. Try out the map type with the following commands:

```
create table demomap ( id int primary key, mapdata
map<text,text>);
insert into demomap json
'{"id":1, "mapdata":{ "key1": "value1","key2":"value2"}}';

select * from demomap;

select json * from demomap;
```

13. Now let's try out the **set** type.

```
create table demoset (id int primary key, myset set<text>);

-- insert as json
insert into demoset json ' { "id":1, "myset":["a","b","c"]}';

-- insert in traditional sql style
insert into demoset (id, myset) values (2, {'hello','paul'});

select * from demoset;
select json * from demoset;
```

14. CQL also supports a list type. See if you can figure it out. If not, there is an example over the page.

15. List example:

```
create table demolist (id int primary key, list list<text>);

insert into demolist (id, list) values (1,['a1','b2','c3']);

select * from demolist;

id | list
----+--------------------
  1 | ['a1', 'b2', 'c3']

(1 rows)

update demolist set list = ['z1'] + list where id = 1;
select * from demolist;

-- what do you expect here?
```

16. That's all for now!