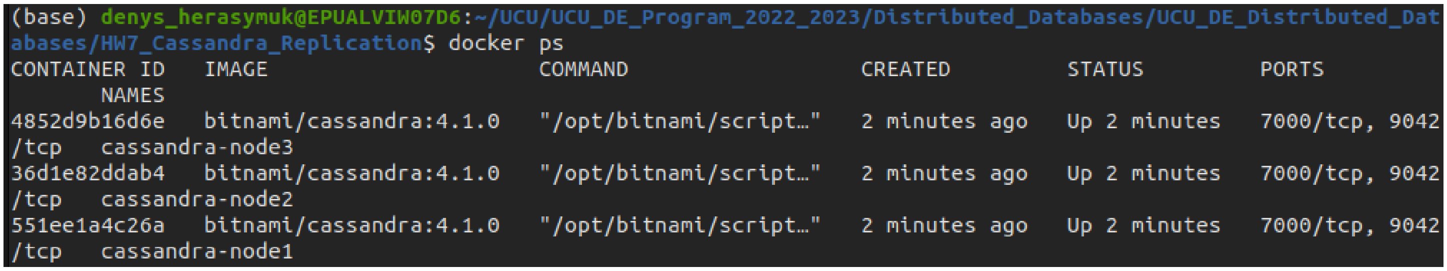
Homework 7. Replication in Cassandra

1) Configure 3 node cluster

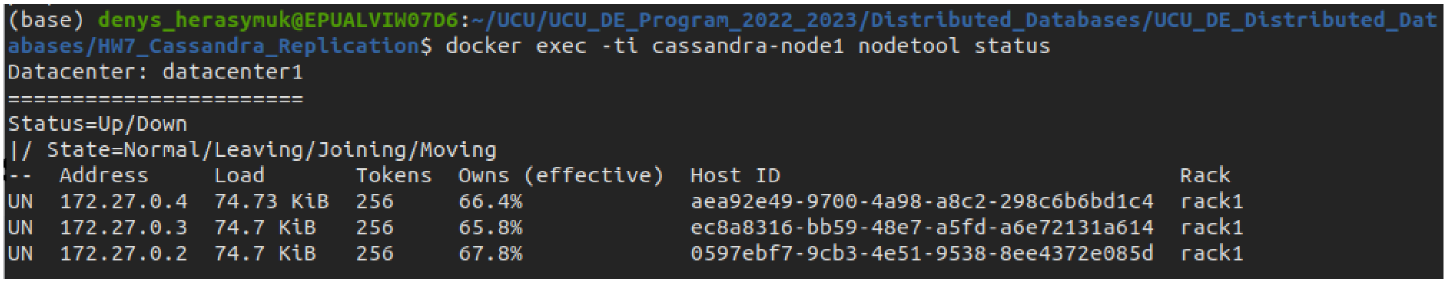
Commands were taken from this tutorial. As a docker image, I used bitnami/cassandra.

I used a docker-compose located in the repo to create a network, run docker containers, and expose ports.



```
(base) denys_herasymuk@EPUALVIW07D6:~/UCU/UCU_DE_Program_2022_2023/Distributed_Databases/UCU_DE_Distributed_Dat
abases/HW7_Cassandra_Replication$ docker ps
CONTAINER ID   IMAGE                      COMMAND                   CREATED          STATUS          PORTS
       NAMES
4852d9b16d6e   bitnami/cassandra:4.1.0    "/opt/bitnami/script…"   2 minutes ago    Up 2 minutes    7000/tcp, 9042
/tcp   cassandra-node3
36d1e82ddab4   bitnami/cassandra:4.1.0    "/opt/bitnami/script…"   2 minutes ago    Up 2 minutes    7000/tcp, 9042
/tcp   cassandra-node2
551ee1a4c26a   bitnami/cassandra:4.1.0    "/opt/bitnami/script…"   2 minutes ago    Up 2 minutes    7000/tcp, 9042
/tcp   cassandra-node1
```

2) Validate configuration



```
(base) denys_herasymuk@EPUALVIW07D6:~/UCU/UCU_DE_Program_2022_2023/Distributed_Databases/UCU_DE_Distributed_Dat
abases/HW7_Cassandra_Replication$ docker exec -ti cassandra-node1 nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns (effective)  Host ID                               Rack
UN  172.27.0.4  74.73 KiB  256     66.4%             aea92e49-9700-4a98-a8c2-298c6b6bd1c4  rack1
UN  172.27.0.3  74.7 KiB   256     65.8%             ec8a8316-bb59-48e7-a5fd-a6e72131a614  rack1
UN  172.27.0.2  74.7 KiB   256     67.8%             0597ebf7-9cb3-4e51-9538-8ee4372e085d  rack1
```

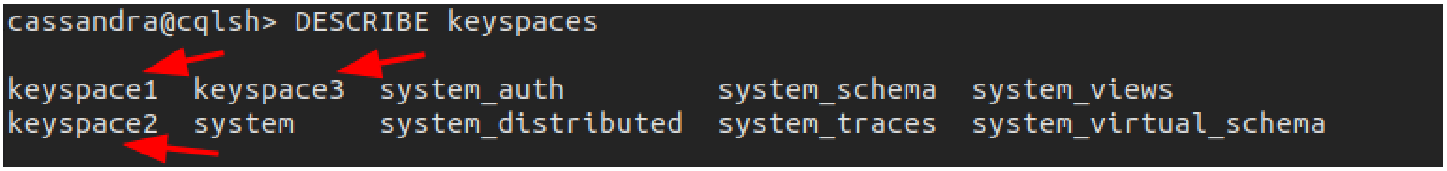
3) Create 3 keyspaces with replication factor 1, 2, 3 using *cqlsh*

Connect to the node 1 and run cqlsh.

```
$ docker exec -it cassandra-node1 bash
I have no name!@5749f4941419:/$ cqlsh
```

Create three keyspaces with a replication factor 1, 2, 3, respectively.

```
cassandra@cqlsh> CREATE  KEYSPACE keyspace1 WITH REPLICATION = {'class' :
'NetworkTopologyStrategy', 'replication_factor' : 1};

cassandra@cqlsh> CREATE  KEYSPACE keyspace2 WITH REPLICATION = {'class' :
'NetworkTopologyStrategy', 'replication_factor' : 2};

cassandra@cqlsh> CREATE  KEYSPACE keyspace3 WITH REPLICATION = {'class' :
'NetworkTopologyStrategy', 'replication_factor' : 3};
```



```
cassandra@cqlsh> DESCRIBE keyspaces

keyspace1  keyspace3  system_auth         system_schema  system_views
keyspace2  system     system_distributed  system_traces  system_virtual_schema
```

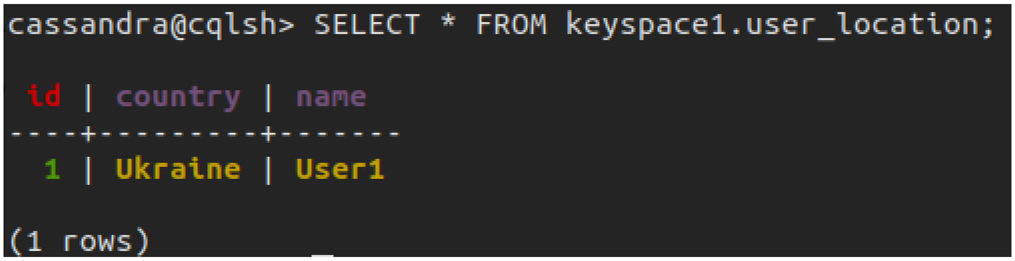
4) Create a table in each of the keyspaces

```
CREATE TABLE keyspace1.user_location (
    id int primary key,
    name text,
    country text
);

CREATE TABLE keyspace2.user_location (
    id int primary key,
    name text,
    country text
);

CREATE TABLE keyspace3.user_location (
    id int primary key,
    name text,
    country text
);
```

5) Write and read to / and from different nodes

Write to and read from keyspace 1 on the node 1.

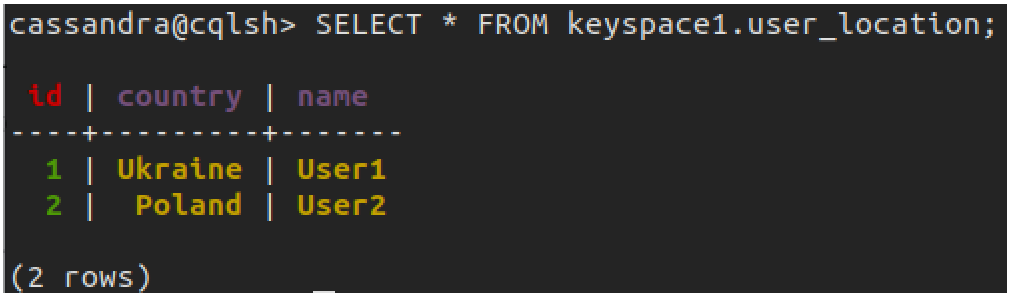
```
INSERT INTO keyspace1.user_location (id, name, country) VALUES (1, 'User1', 'Ukraine');

SELECT * FROM keyspace1.user_location;
```



```
cassandra@cqlsh> SELECT * FROM keyspace1.user_location;

 id | country | name
----+---------+-------
  1 | Ukraine | User1

(1 rows)
```

Write to and read from keyspace 1 on the node 2.

```
$ docker exec -it cassandra-node2 bash

cqlsh> INSERT INTO keyspace1.user_location (id, name, country) VALUES (2, 'User2',
'Poland');

cqlsh> SELECT * FROM keyspace1.user_location;
```



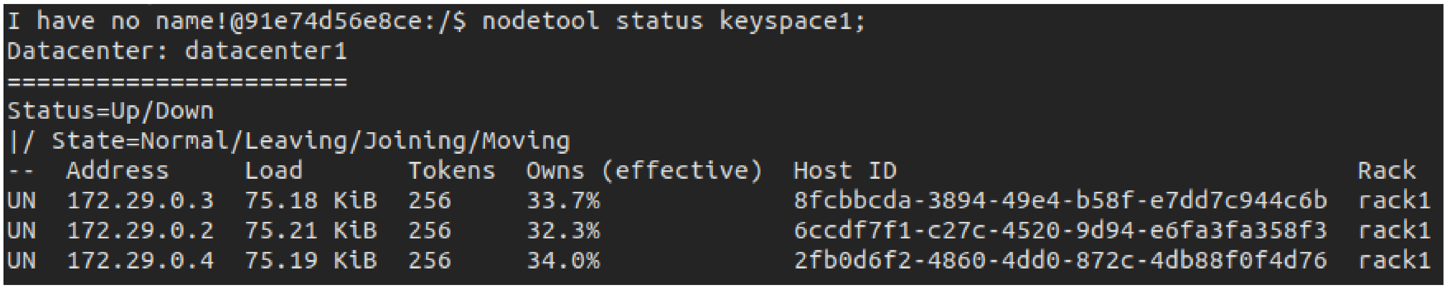
```
cassandra@cqlsh> SELECT * FROM keyspace1.user_location;

 id | country | name
----+---------+-------
  1 | Ukraine | User1
  2 |  Poland | User2

(2 rows)
```

6) Insert the data into the created tables and look at their distribution across the cluster nodes

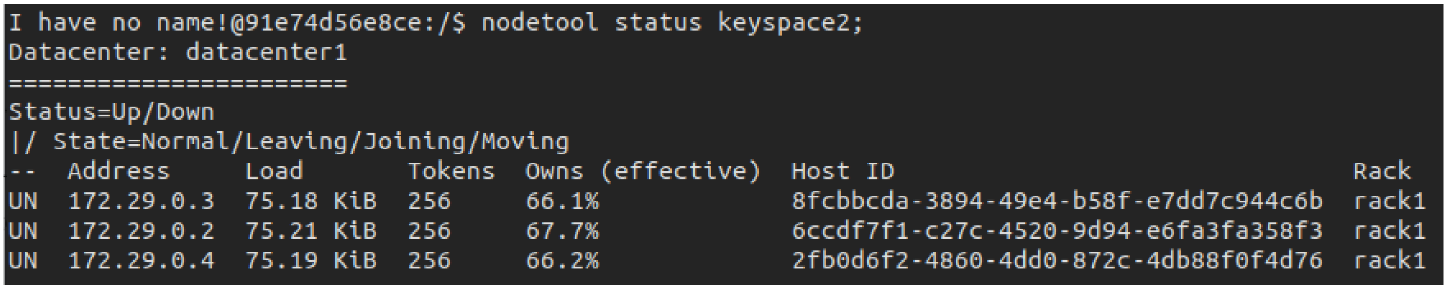
Execute the following commands for each of the keyspaces.

```
INSERT INTO keyspace1.user_location (id, name, country) VALUES (3, 'User3', 'Ukraine');
INSERT INTO keyspace1.user_location (id, name, country) VALUES (4, 'User4', 'Poland');
…
INSERT INTO keyspace1.user_location (id, name, country) VALUES (9, 'User9', 'Ukraine');
INSERT INTO keyspace1.user_location (id, name, country) VALUES (10, 'User10',
'Poland');
```

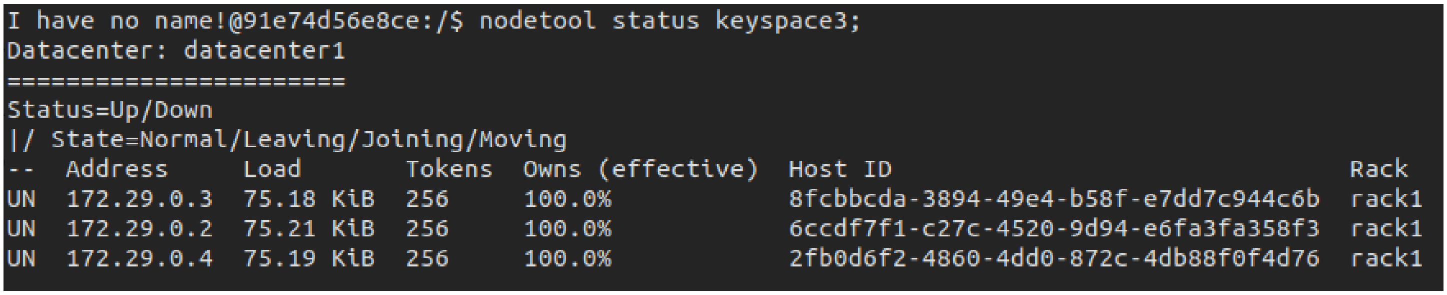
Look at the data distribution.



```
I have no name!@91e74d56e8ce:/$ nodetool status keyspace1;
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns (effective)  Host ID                               Rack
UN  172.29.0.3  75.18 KiB  256     33.7%             8fcbbcda-3894-49e4-b58f-e7dd7c944c6b  rack1
UN  172.29.0.2  75.21 KiB  256     32.3%             6ccdf7f1-c27c-4520-9d94-e6fa3fa358f3  rack1
UN  172.29.0.4  75.19 KiB  256     34.0%             2fb0d6f2-4860-4dd0-872c-4db88f0f4d76  rack1
```

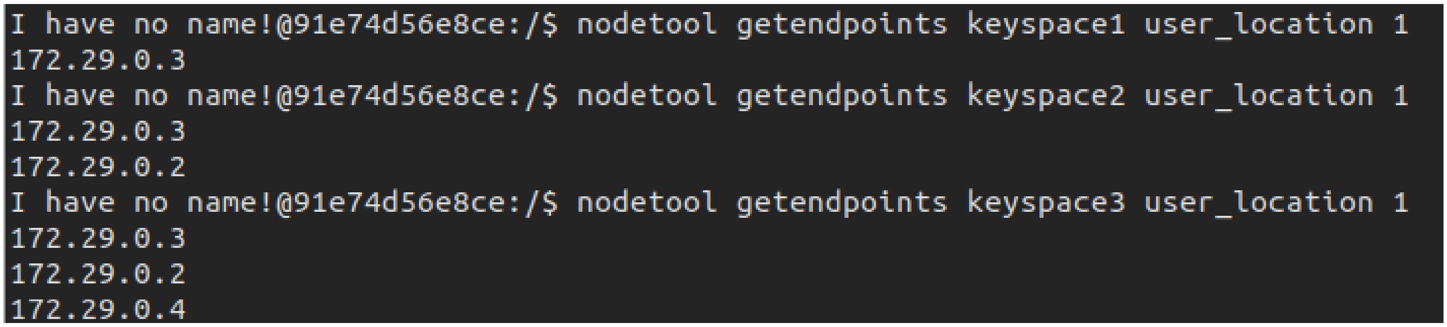


```
I have no name!@91e74d56e8ce:/$ nodetool status keyspace2;
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns (effective)  Host ID                               Rack
UN  172.29.0.3  75.18 KiB  256     66.1%             8fcbbcda-3894-49e4-b58f-e7dd7c944c6b  rack1
UN  172.29.0.2  75.21 KiB  256     67.7%             6ccdf7f1-c27c-4520-9d94-e6fa3fa358f3  rack1
UN  172.29.0.4  75.19 KiB  256     66.2%             2fb0d6f2-4860-4dd0-872c-4db88f0f4d76  rack1
```



```
I have no name!@91e74d56e8ce:/$ nodetool status keyspace3;
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns (effective)  Host ID                               Rack
UN  172.29.0.3  75.18 KiB  256     100.0%            8fcbbcda-3894-49e4-b58f-e7dd7c944c6b  rack1
UN  172.29.0.2  75.21 KiB  256     100.0%            6ccdf7f1-c27c-4520-9d94-e6fa3fa358f3  rack1
UN  172.29.0.4  75.19 KiB  256     100.0%            2fb0d6f2-4860-4dd0-872c-4db88f0f4d76  rack1
```

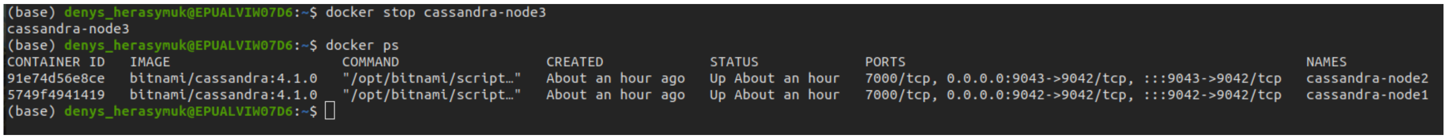
7) For any record from each of the keyspaces, display the nodes on which the data is stored



```
I have no name!@91e74d56e8ce:/$ nodetool getendpoints keyspace1 user_location 1
172.29.0.3
I have no name!@91e74d56e8ce:/$ nodetool getendpoints keyspace2 user_location 1
172.29.0.3
172.29.0.2
I have no name!@91e74d56e8ce:/$ nodetool getendpoints keyspace3 user_location 1
172.29.0.3
172.29.0.2
172.29.0.4
```

8) Disable one of the nodes. For each of the keyspaces, determine with which levels of *consistency* we can read and write, and which of them provide *strong consistency*

Disable node 3.



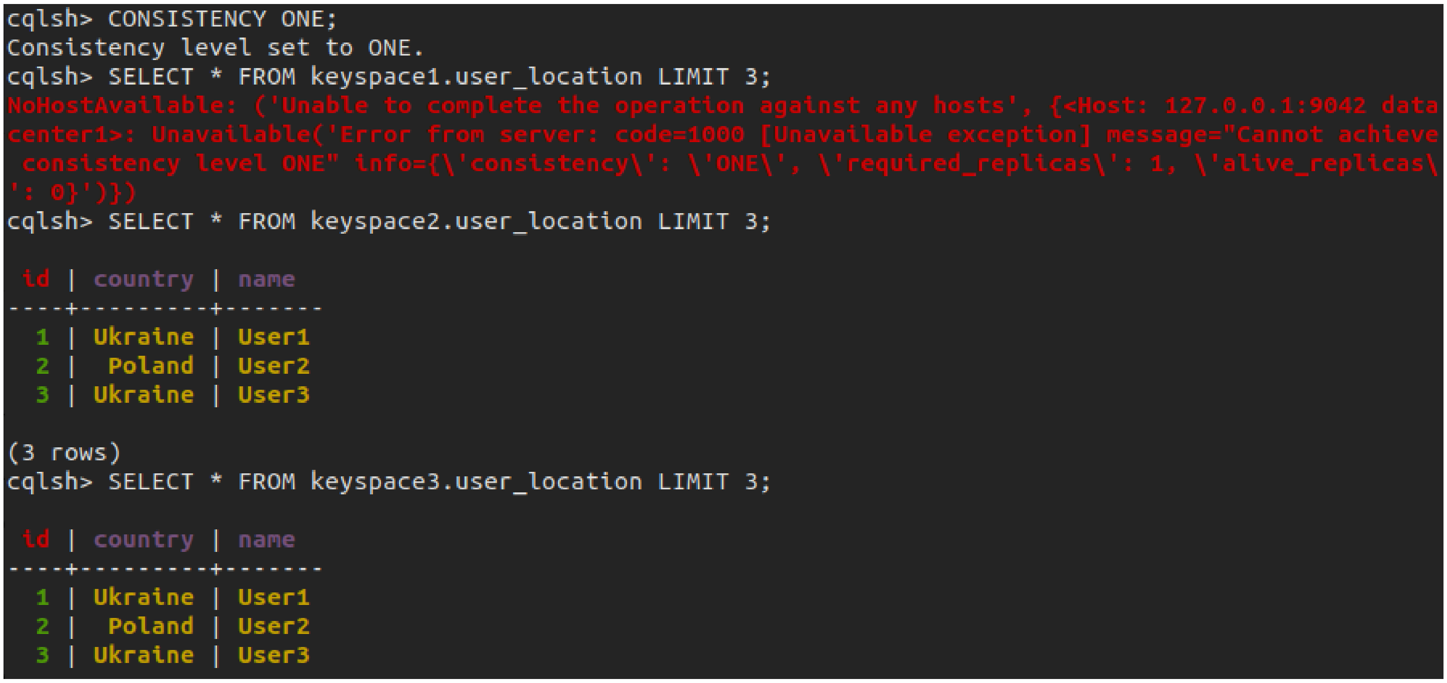
```
(base) denys_herasymuk@EPUALVIW07D6:~$ docker stop cassandra-node3
cassandra-node3
(base) denys_herasymuk@EPUALVIW07D6:~$ docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED             STATUS             PORTS                                                  NAMES
91e74d56e8ce   bitnami/cassandra:4.1.0   "/opt/bitnami/script…"   About an hour ago   Up About an hour   7000/tcp, 0.0.0.0:9043->9042/tcp, :::9043->9042/tcp   cassandra-node2
5749f4941419   bitnami/cassandra:4.1.0   "/opt/bitnami/script…"   About an hour ago   Up About an hour   7000/tcp, 0.0.0.0:9042->9042/tcp, :::9042->9042/tcp   cassandra-node1
(base) denys_herasymuk@EPUALVIW07D6:~$ 
```

Consistency levels in Cassandra
https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlConfigConsistency.html

Execute the following commands to understand the allowed read consistency levels for each keyspace.

```
CONSISTENCY <level>;

SELECT * FROM keyspace1.user_location LIMIT 3;
SELECT * FROM keyspace2.user_location LIMIT 3;
SELECT * FROM keyspace3.user_location LIMIT 3;
```

Example of the output.



```
cqlsh> CONSISTENCY ONE;
Consistency level set to ONE.
cqlsh> SELECT * FROM keyspace1.user_location LIMIT 3;
NoHostAvailable: ('Unable to complete the operation against any hosts', {<Host: 127.0.0.1:9042 data
center1>: Unavailable('Error from server: code=1000 [Unavailable exception] message="Cannot achieve
 consistency level ONE" info={\'consistency\': \'ONE\', \'required_replicas\': 1, \'alive_replicas\
': 0}')})
cqlsh> SELECT * FROM keyspace2.user_location LIMIT 3;

 id | country | name
----+---------+-------
  1 | Ukraine | User1
  2 |  Poland | User2
  3 | Ukraine | User3

(3 rows)
cqlsh> SELECT * FROM keyspace3.user_location LIMIT 3;

 id | country | name
----+---------+-------
  1 | Ukraine | User1
  2 |  Poland | User2
  3 | Ukraine | User3
```

Read consistency summary

- **Keyspace 1**: None of the consistency levels works.
- **Keyspace 2**: Works only with ONE.
- **Keyspace 3**: Works with ONE, TWO, SERIAL, QUORUM (but not THREE or ALL).

Execute the following commands to understand the allowed write consistency levels for each keyspace.

```
CONSISTENCY <level>;

INSERT INTO keyspace1.user_location (id, name, country) VALUES (11, 'User11',
'Ukraine');
INSERT INTO keyspace2.user_location (id, name, country) VALUES (11, 'User11',
'Ukraine');
INSERT INTO keyspace3.user_location (id, name, country) VALUES (11, 'User11',
'Ukraine');
```

Write consistency summary

- **Keyspace 1**: Works with ANY, ONE.
- **Keyspace 2**: Works with ANY, ONE.
- **Keyspace 3**: Works with ANY, ONE, TWO, QUORUM (but not THREE or ALL).

Strong consistency summary

Strong consistency can be achieved if W + R > RF, where R – read CL replica count, W – write CL replica count, RF – replication factor (src).

- **Keyspace 1**: Since there is only one replica, read and write operations are strongly consistent..
- **Keyspace 2**: No strongly consistent combinations.
- **Keyspace 3**: The following combinations are strongly consistent (R + W): TWO + TWO, TWO + QUORUM, QUORUM + QUORUM, SERIAL + TWO, SERIAL + QUORUM.

9-11) Create network partition and value conflict

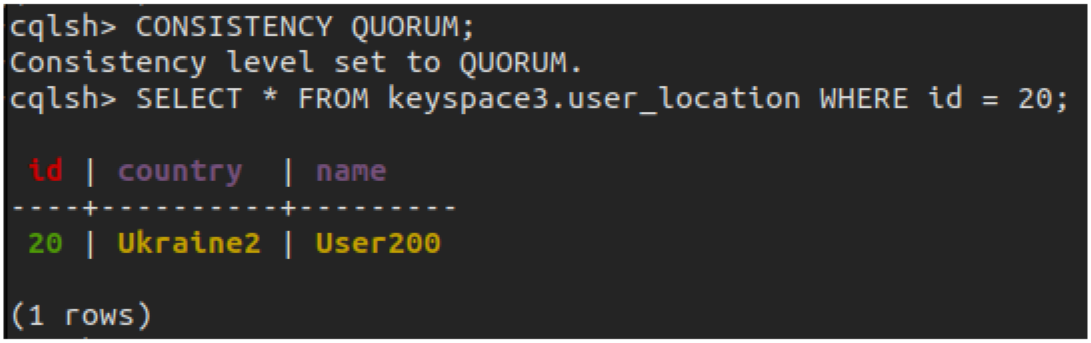
Write different values on each of the nodes.

```
# ====================== Write one record only on node 1 ======================
docker stop cassandra-node2
docker stop cassandra-node3

cassandra-node1: INSERT INTO keyspace3.user_location (id, name, country) VALUES (20,
'User20', 'Ukraine');


# ====================== Write one record only on node 3 ======================
docker start cassandra-node3
docker stop cassandra-node2

cassandra-node3: INSERT INTO keyspace3.user_location (id, name, country) VALUES (20,
'User2000', 'Ukraine3');


# ====================== Write one record only on node 2 ======================
docker start cassandra-node2
docker stop cassandra-node1

cassandra-node2: INSERT INTO keyspace3.user_location (id, name, country) VALUES (20,
'User200', 'Ukraine2');


docker start cassandra-node1
docker start cassandra-node2
```

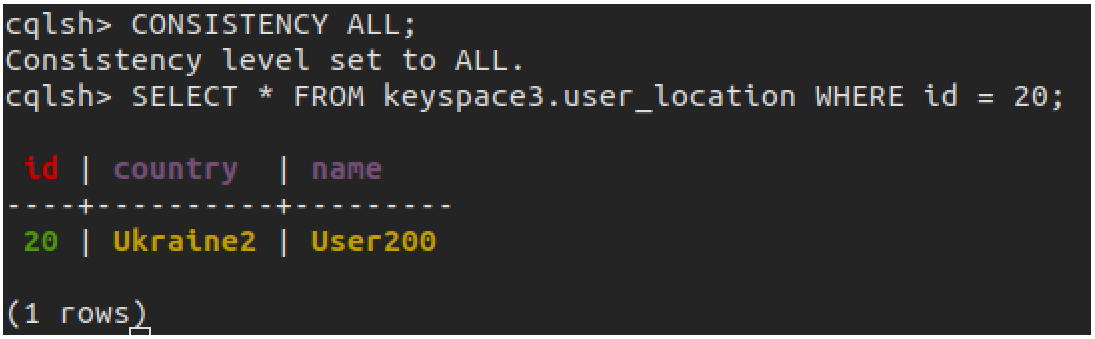
First, reading with **consistency level ONE** returned id=20, name=User20, country=Ukraine. However, in several seconds it returned id=20, name=User200, country=Ukraine2.

Read with **consistency level QUORUM/ALL** returns id=20, name=User200, country=Ukraine2.

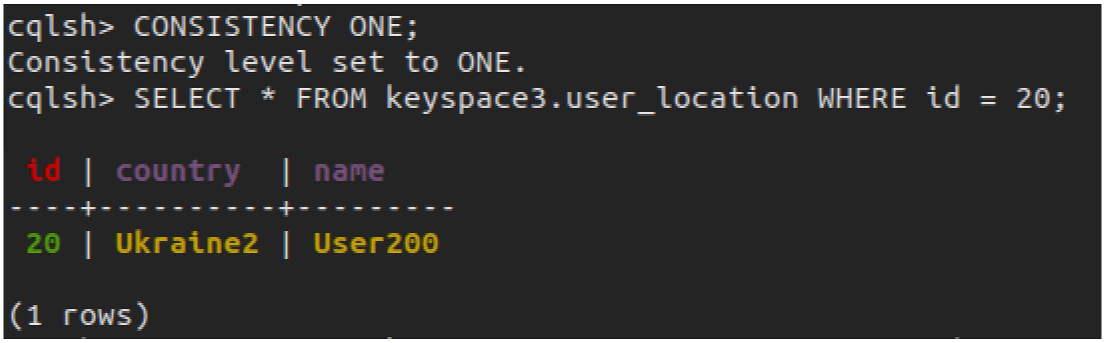
This means that Read Repair occurred, and the nodes synchronized and used the last written value for the id.



```
cqlsh> CONSISTENCY QUORUM;
Consistency level set to QUORUM.
cqlsh> SELECT * FROM keyspace3.user_location WHERE id = 20;

 id | country  | name
----+----------+---------
 20 | Ukraine2 | User200

(1 rows)
```



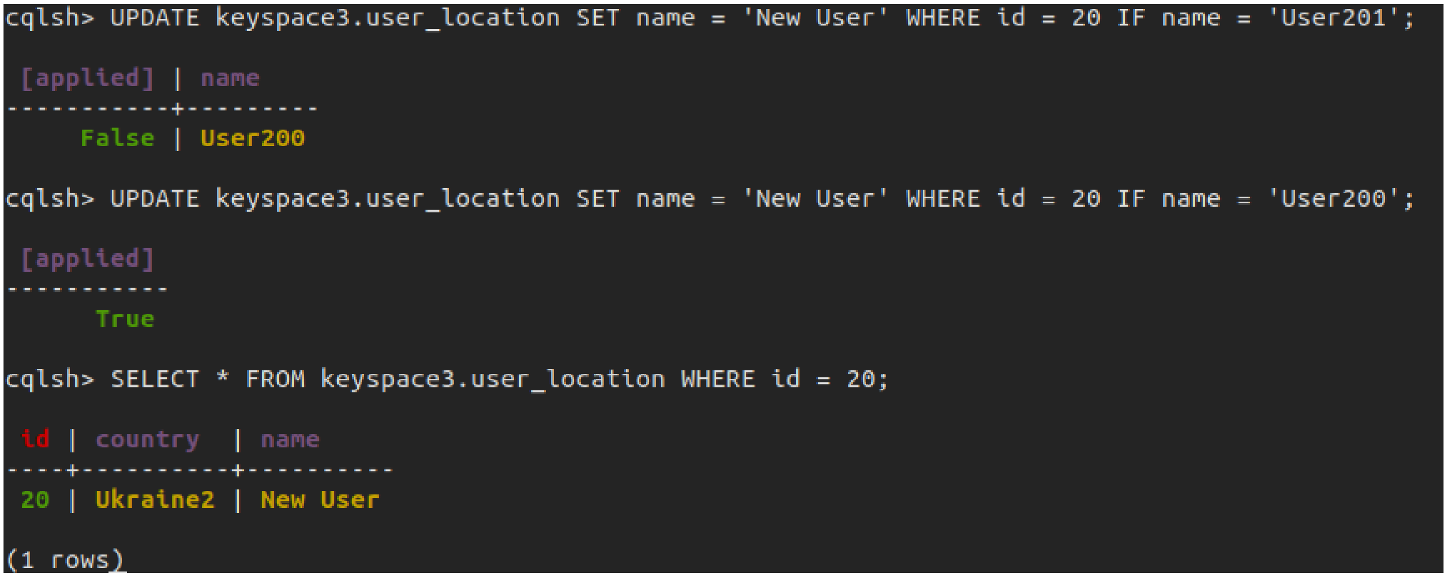
```
cqlsh> CONSISTENCY ALL;
Consistency level set to ALL.
cqlsh> SELECT * FROM keyspace3.user_location WHERE id = 20;

 id | country  | name
----+----------+---------
 20 | Ukraine2 | User200

(1 rows)
```



```
cqlsh> CONSISTENCY ONE;
Consistency level set to ONE.
cqlsh> SELECT * FROM keyspace3.user_location WHERE id = 20;

 id | country  | name
----+----------+---------
 20 | Ukraine2 | User200

(1 rows)
```

12) Check the behavior of lightweight transactions for previous points in a non-partitioned and partitioned cluster

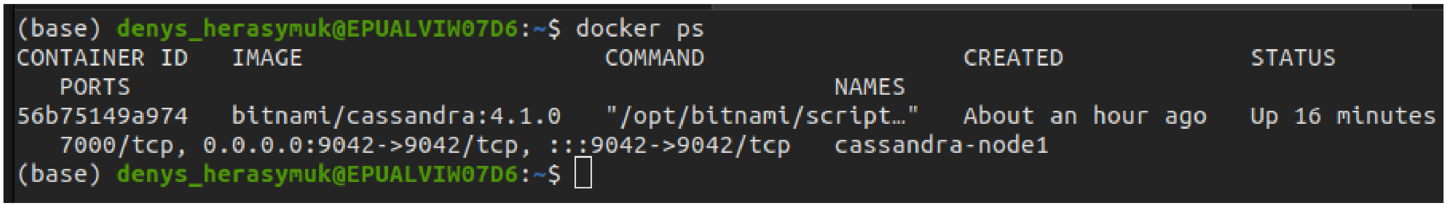
1. Non-partitioned cluster case



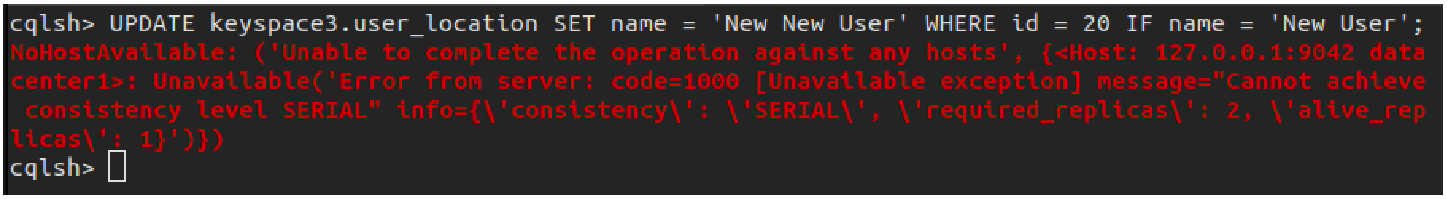
```
cqlsh> UPDATE keyspace3.user_location SET name = 'New User' WHERE id = 20 IF name = 'User201';

 [applied] | name
-----------+---------
     False | User200

cqlsh> UPDATE keyspace3.user_location SET name = 'New User' WHERE id = 20 IF name = 'User200';

 [applied]
-----------
      True

cqlsh> SELECT * FROM keyspace3.user_location WHERE id = 20;

 id | country  | name
----+----------+----------
 20 | Ukraine2 | New User

(1 rows)
```

2. Partitioned cluster case

Disable node 2 and 3.



```
(base) denys_herasymuk@EPUALVIW07D6:~$ docker ps
CONTAINER ID   IMAGE                        COMMAND                  CREATED              STATUS
   PORTS                                                 NAMES
56b75149a974   bitnami/cassandra:4.1.0      "/opt/bitnami/script…"   About an hour ago    Up 16 minutes
   7000/tcp, 0.0.0.0:9042->9042/tcp, :::9042->9042/tcp   cassandra-node1
(base) denys_herasymuk@EPUALVIW07D6:~$
```



```
cqlsh> UPDATE keyspace3.user_location SET name = 'New New User' WHERE id = 20 IF name = 'New User';
NoHostAvailable: ('Unable to complete the operation against any hosts', {<Host: 127.0.0.1:9042 data
center1>: Unavailable('Error from server: code=1000 [Unavailable exception] message="Cannot achieve
 consistency level SERIAL" info={\'consistency\': \'SERIAL\', \'required_replicas\': 2, \'alive_rep
licas\': 1}')})
cqlsh>
```

Since lightweight transactions have SERIAL consistency level (similar to QUORUM), the update query above requires at least 2 available nodes.