Last Name: Phuengmak First Name: Phichayapha Student ID: 84467840

1.

a) CQL Query: DESCRIBE Hoofers

b) Result:

```
token@cqlsh> DESCRIBE Hoofers

CREATE KEYSPACE hoofers WITH replication = {'class': 'NetworkTopologyStrategy', 'us-east1': '3'}  AND durable_writes = true;

CREATE TABLE hoofers.boats (
    bid int PRIMARY KEY,
    bname text,
    color text
) WITH additional_write_policy = '99PERCENTILE'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.UnifiedCompactionStrategy'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99PERCENTILE';
```

c) Answers: The Hoofer keyspace maintains 3 copies of the data. It resides in a cloud region us-east1. The write quorum size is (3 / 2) + 1 = 2 and the read quorum is 2.

2.

a) CQL CREATE Statements:

```
CREATE TABLE IF NOT EXISTS InstructorEducation (
        instructor_id INT,
        education_id INT,
        degree TEXT,
        major TEXT,
        school TEXT,
        graduation_year INT,
        PRIMARY KEY (instructor_id, education_id) );

CREATE TABLE IF NOT EXISTS Meetings (
        meeting_id INT PRIMARY KEY,
        meeting_name TEXT,
        passcode TEXT,
        start_at TIMESTAMP,
        duration INT,
        mute_participants BOOLEAN,
        course_id INT,
        instructor_id INT,
        recurr_id INT );

CREATE TABLE IF NOT EXISTS Recordings (
        recording_id INT PRIMARY KEY,
        start_time TIMESTAMP,
        end_time TIMESTAMP,
        meeting_id INT );

CREATE TABLE IF NOT EXISTS Posts (
        post_id INT PRIMARY KEY,
        post_type TEXT,
        body TEXT,
        created_at TIMESTAMP,
        user_id INT,
        meeting_id INT,
        replied_to_post_id INT,
        topics TEXT );
```

3.

a) PostgreSQL COPY commands:

```
COPY InstructorEducation TO 'E:\CS122D\HW2_instructor_education.csv' WITH (FORMAT CSV, HEADER);
COPY Meetings TO 'E:\CS122D\HW2_meetings.csv' WITH (FORMAT CSV, HEADER);
COPY Recordings TO 'E:\CS122D\HW2_recordings.csv' WITH (FORMAT CSV, HEADER);
COPY Posts TO 'E:\CS122D\HW2_posts.csv' WITH (FORMAT CSV, HEADER);
```

4.

a) First CQL Query:

```
SELECT post_id, user_id, topics, replied_to_post_id
FROM posts
WHERE post_type = 'question'
LIMIT 10;
```

b) Result:

```
token@cqlsh:swoosh> SELECT post_id, user_id, topics, replied_to_post_id
         ... FROM posts
         ... WHERE post_type = 'question';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filt
ering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability
, use ALLOW FILTERING"
```

c) Modified CQL Query:

```
SELECT post_id, user_id, topics, replied_to_post_id
FROM posts
WHERE post_type = 'question'
LIMIT 10
ALLOW FILTERING;
```

b) Result:

post_id	user_id	topics	replied_to_post_id
23	302	midterm1, finalproject, module3	null
660	183	null	null
893	104	project1, homework5, lectures	null
878	44	midterm2, homework6, logistics	null
128	358	homework3, project3, homework2	127
251	266	homework3, project2, module2	null
744	399	null	424
849	376	project1, homework2, finalexam	665
919	246	null	657
117	337	module2	null

(10 rows)

5.

a) CQL Create Statement:

```
CREATE TABLE IF NOT EXISTS Posts_q5 (
        post_id INT,
        post_type TEXT,
        body TEXT,
        created_at TIMESTAMP,
        user_id INT,
        meeting_id INT,
        replied_to_post_id INT,
        topics TEXT,
        PRIMARY KEY ((post_type), post_id) );
```

b) CQL Query:

```
SELECT post_id, user_id, topics, replied_to_post_id
FROM posts_q5
WHERE post_type = 'question'
LIMIT 10;
```

c) Result:

post_id	user_id	topics	replied_to_post_id
1	375	null	null
4	52	null	2
7	375	project4, homework3	1
9	52	module1, logistics, homework2	7
12	0	project2, homework2, midterm1	7
13	97	module4	4
14	482	finalexam, logistics, project2	null
15	42	homework6, homework5, midterm2	null
18	83	null	15
23	302	midterm1, finalproject, module3	null

(10 rows)

d) Explanation:

Changing the partitioning key results in Cassandra not discouraging the query because now that the data is partitioned by post_type, it does not need to look into every node for post_type = ‘question’. It can simply search the node where ‘question’ is partitioned into.

The primary key needs to include post_id because post_type by itself is not unique. There are many posts in which its post_type is ‘question’. Post_id, however, is unique so including it into the primary key makes the key unique.

6.

a) CQL Query:

```
SELECT topics, replied_to_post_id
FROM posts
WHERE post_type = 'question'
ORDER BY created_at DESC
LIMIT 10;
```

b) CQL CREATE Statement:

```
CREATE TABLE IF NOT EXISTS Posts_q6 (
        post_id INT,
        post_type TEXT,
        body TEXT,
        created_at TIMESTAMP,
        user_id INT,
        meeting_id INT,
        replied_to_post_id INT,
        topics TEXT,
        PRIMARY KEY ((post_type), created_at, post_id))
        WITH CLUSTERING ORDER BY (created_at DESC, post_id ASC);

SELECT topics, replied_to_post_id
FROM posts_q6
WHERE post_type = 'question'
LIMIT 10;
```

c) Results:

```
 topics                        | replied_to_post_id
-------------------------------+--------------------
                          null |                363
           homework4, project1 |                703
                          null |                186
                          null |               null
                          null |                 67
                          null |                241
                          null |               null
  module4, project3, homework2 |                465
                          null |                293
 midterm2, midterm1, logistics |                 34

(10 rows)
```

d) Explanation:

Adding a new clustering key changes Cassandra's mind because data in each partition is now ordered by created_at and post_id. By choosing created_at DESC, created_at in each partition is in descending order.

7.

a) CQL Create Statement:

```
/*
dsbulk load ignores columns that exists in the .csv file but doesn't exist in the table
*/

CREATE TABLE IF NOT EXISTS Posts_q7a (
        post_id INT,
        post_type TEXT,
        user_id INT,
        meeting_id INT,
        PRIMARY KEY ((user_id), meeting_id))
        WITH CLUSTERING ORDER BY (meeting_id DESC);
```

b) CQL Create Statement:

```
CREATE TABLE IF NOT EXISTS EnrolledIn_q7b (
        user_id INT,
        course_id INT,
        PRIMARY KEY ((user_id), course_id) )
        WITH CLUSTERING ORDER BY (course_id ASC);
```

c) CQL Create Statement:

```
CREATE TABLE IF NOT EXISTS InstructorEducation_q7c (
        instructor_id INT,
        education_id INT,
        major TEXT,
        school TEXT,
        graduation_year INT,
        PRIMARY KEY ((instructor_id), graduation_year, education_id) )
        WITH CLUSTERING ORDER BY (graduation_year ASC, education_id ASC);
```

d) CQL Create Statement:

```
CREATE TABLE IF NOT EXISTS Meetings_q7d (
        meeting_id INT,
        start_at TIMESTAMP,
        duration INT,
        instructor_id INT,
        PRIMARY KEY ((instructor_id), start_at, meeting_id) )
        WITH CLUSTERING ORDER BY (start_at ASC, meeting_id ASC);
```

8.

a)

- CQL Query:

```
SELECT post_id, post_type
FROM Posts_q7a
WHERE user_id = 291;
```

- Result:

```
 post_id | post_type
---------+-----------
      28 |  question
     648 |  question
     215 |  question
     503 |      note
     119 |  question
      24 |      note

(6 rows)
```

b)

- CQL Query:

```
SELECT COUNT (course_id)
FROM EnrolledIn_q7b
WHERE user_id = 13;
```

- Result:

```
 system.count(course_id)
-------------------------
                       4

(1 rows)
```

c)

- CQL Query:

```
SELECT school, major, graduation_year
FROM InstructorEducation_q7c
WHERE instructor_id = 39;
```

- Result:

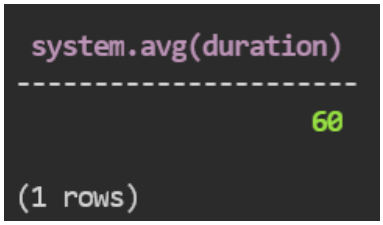
```
 school               | major                | graduation_year
----------------------+----------------------+-----------------
        Yale University |     Computer Science |            1987
                    UCR |           Literature |            1997
 University of Maryland |             Biometry |            2000
                   UCLA | Financial Engineering |            2016
                    UCR |           Literature |            2017

(5 rows)
```

d)

- CQL Query:

```
SELECT AVG(duration)
FROM Meetings_q7d
WHERE instructor_id = 486 AND start_at >= '2020-01-06' AND start_at <= '2020-05-04';
```

- Result:



9.

a) CQL INSERT statements:

```
INSERT INTO Meetings JSON '{
        "meeting_id": "1507",
        "meeting_name": "NoSQL",
        "passcode": "@@R",
        "start_at": "2022-02-10 8:00:00",
        "duration": "90",
        "mute_participants": "TRUE",
        "course_id": "1",
        "instructor_id": "486",
        "recurr_id": "4"
}';

INSERT INTO Recordings JSON '{
        "recording_id": "3730",
        "start_time": "2022-02-10 8:00:00",
        "end_time": "2022-02-24 9:30:00",
        "meeting_id": "1507"
}';

INSERT INTO Meetings_q7d JSON '{
        "meeting_id": "1507",
        "start_at": "2022-02-10 8:00:00",
        "duration": "90",
        "instructor_id": "486"
}';
```

10. [Extra Credit]

Python script: