Last Name: Senyurt First Name: Cem Student ID: 45064076

1.

a) CQL Query:

DESCRIBE keyspace hoofers;

b) Result:

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:

We can answer the following questions with looking into the output,

- How many copies of the data does the Hoofers keyspace maintain?

CREATE KEYSPACE hoofers WITH replication = {'class': 'NetworkTopologyStrategy', 'us-east1': '3'} AND durable_writes = true;
This shows us that there are 3 replicas of the data in Hoofers keyspace.

- Which cloud region does it reside in?

us-east1

- Last but not least, what should the read and write quorum sizes (R & W) be for consistent writes and reads when working with this keyspace?

The read and write quorum sizes: 2

2.

a) CQL CREATE Statements:

```
CREATE TABLE Users(
    user_id     text,
    email       text,
    first_name    text,
    last_name    text,
    joined_date    date,
    street      text,
    city       text,
    state       text,
    zip       text,
    categories       text,
    PRIMARY KEY (user_id)
);
```

```
CREATE TABLE Items (
    item_id       text,
    name          text,
    price         decimal,
    category      text,
    description   text,
    seller_user_id text,
    list_date     date,
    buyer_user_id  text,
    purchase_date  date,
    PRIMARY KEY(item_id)
);

CREATE TABLE Ads (
    ad_id         text,
    plan          text,
    content       text,
    pic_num       int,
    item_id       text,
    seller_user_id text,
    placed_date   date,
    PRIMARY KEY(ad_id)
);

CREATE TABLE Ratings (
    buyer_id      text,
    seller_id     text,
    quality       int,
    pricing       int,
    delivery      int,
    rating_date   date,
    PRIMARY KEY ((buyer_id, seller_id))
);
```

3.

a) PostgreSQL COPY commands:

COPY interchange.users TO '/Applications/Homework2/csv/users.csv' WITH (FORMAT CSV, HEADER);

COPY interchange.ads TO '/Applications/Homework2/csv/ads.csv' WITH (FORMAT CSV, HEADER);

COPY interchange.items TO '/Applications/Homework2/csv/items.csv' WITH (FORMAT CSV, HEADER);

COPY interchange.ratings TO '/Applications/Homework2/csv/ratings.csv' WITH (FORMAT CSV, HEADER);

4.

a) First CQL Query:

SELECT user_id, first_name, last_name, email FROM users WHERE last_name = 'Harris';

b) Result:

InvalidRequest: Error from server: 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"

c) Modified CQL Query:

SELECT user_id, first_name, last_name, email FROM users WHERE last_name = 'Harris' ALLOW FILTERING;

b) Result:

```
 user_id  | first_name | last_name |     email
----------+------------+-----------+---------------------------
  6OUEY |     Jason     |     Harris     |    973harris1215@gmail.com
  1DDFX |     Sheila     |     Harris     |    Harris.Sheila@hotmail.com
  0VI2U  |    Matthew |     Harris     |    Matthew.harris44174@aol.com
  M8Q98 |       Don      |     Harris     |    harrisdon78344@hotmail.com
  8XKTJ  |     Karen     |     Harris     |    harris.Kar@gmail.com
  KY5J5  |     Lauren    |     Harris     |    78572Harris.lauren8330@aol.com
  OKUGQ |     Andrew  |     Harris     |    harris_andrew@yahoo.com

(7 rows)
```

5.

a) CQL Create Statement:

```
CREATE TABLE users_q5(
    user_id     text,
    email       text,
    first_name    text,
    last_name    text,
    joined_date    date,
    street      text,
    city       text,
    state       text,
    zip       text,
    categories       text,
    PRIMARY KEY ((last_name), user_id)
);
```

b) CQL Query:

```
SELECT user_id, first_name, last_name, email FROM users_q5 WHERE last_name = 'Harris';
```

c) Result:

```
 user_id    |  first_name  |  last_name |    email
------------+----------------+----------------+------------------------------
   0VI2U   |    Matthew   |      Harris     |    Matthew.harris44174@aol.com
   1DDFX  |      Sheila      |      Harris     |    Harris.Sheila@hotmail.com
   6OUEY  |      Jason       |      Harris     |    973harris1215@gmail.com
   8XKTJ   |      Karen       |      Harris     |    harris.Kar@gmail.com
   KY5J5   |     Lauren       |      Harris     |    78572Harris.lauren8330@aol.com
   M8Q98 |       Don        |      Harris     |    harrisdon78344@hotmail.com
   OKUGQ |     Andrew     |      Harris     |    harris_andrew@yahoo.com
(7 rows)
```

d) Explanation:

So, in the previous table (users) design the primary key was user_id only. That means, Cassandra stored all the data for each user in a single partition in a single node where the data is separated by the user_id.

In this question with the new table design (users_q5) where the primary key is ((last_name), user_id), this means partition key is last_name, Cassandra partitions the data based on the last_name value, and stores all users with the same last_name value together in the same node. So when we use 'WHERE' (filtering) Cassandra will know which partition to read from.

This design led us to optimize our query which is a huge win. Also the reason why the user_id is also included in the primary key is to maintain and ensure the uniqueness of the rows in the table.

As we talked in the class, we can have multiple users with the same last name, having only last_name as the primary key would result in multiple rows with the same partition key, and we don't want that in Cassandra. Adding user_id will get rid of this case!

6.

a) CQL Query:

```
SELECT first_name, last_name, email, user_id FROM users_q6 WHERE last_name = 'Davis' ORDER BY joined_date DESC LIMIT 10;
```

b) CQL CREATE Statement:

```
CREATE TABLE users_q6(
    user_id     text,
    email       text,
    first_name    text,
    last_name    text,
    joined_date    date,
    street      text,
    city       text,
    state       text,
    zip       text,
    categories       text,
    PRIMARY KEY ((last_name), joined_date, user_id)
);
```

c) Results:

first_name	last_name	email	user_id
null	Davis	davis33@gmail.com	FCQGQ
Jason	Davis	Jas_davis@gmail.com	7WVZG
Janet	Davis	Davis2342@gmail.com	N0DAI
John	Davis	John76893@gmail.com	TABM6
Robert	Davis	Robert88665@yahoo.com	DSQMQ
Matthew	Davis	matthew_davis@gmail.com	91697
Wesley	Davis	davis72@gmail.com	ABX8N
Christine	Davis	Christine_davis@gmail.com	VLEDI
Jacob	Davis	davisjacob@gmail.com	ELS9U
Jennifer	Davis	davis_jen17097@gmail.com	V2EVU

(10 rows)

d) Explanation:

In order to use ORDER BY, we have to use the cluster key in the PRIMARY KEY definition, and in this case it is joined_date. With that Cassandra will be able to perform the ordering command over the joined date, and we can ask by DESC, and limit that by 10 for this particular query.

7.

a) CQL Create Statement:

```
CREATE TABLE Items_q7a (
    item_id       text,
    price         decimal,
    seller_user_id text,
    PRIMARY KEY((seller_user_id), price, item_id)
) WITH CLUSTERING ORDER BY (price DESC, item_id ASC);
```

	item_id [PK] text	price numeric (8,2)	seller_user_id text
1	WN6IS	1445.77	XZJXD
2	W1UN9	1254.63	G6UPG
3	VPUEM	417.77	4CXZQ
4	H9V8N	1840.89	O9NN6
5	EFIJC	498.2	HYR8F
6	0R25W	1708.14	X9W2Z
7	KV1DG	1032.59	G9Q2G
8	XHCBM	1396.52	XMWLA
9	TW1LD	1209.81	8K3RE
10	DKKV7	857.98	R2YNB
11	QJ9A3	31.51	1149O

Total rows: 1000 of 10000 | Query complete 00:00:

b) CQL Create Statement:

```
CREATE TABLE Items_q7b (
    category          text,
    item_id           text,
    buyer_user_id     text,
    PRIMARY KEY((category), item_id)
);
```

	category text	item_id [PK] text	buyer_user_id text
1	Pet Supplies	WN6IS	[null]
2	Sports & Outdoors	W1UN9	[null]
3	Beauty & Personal Care	VPUEM	[null]
4	Beauty & Personal Care	H9V8N	[null]
5	Sports & Outdoors	EFIJC	[null]
6	Arts, Crafts & Sewing	0R25W	[null]
7	Clothing, Shoes & Jewelry	KV1DG	[null]
8	Home & Kitchen	XHCBM	[null]
9	Home & Kitchen	TW1LD	[null]
10	Sports & Outdoors	DKKV7	[null]
11	Office Products	QJ9A3	[null]
12	Office Products	ITY0F	[null]

Total rows: 1000 of 10000 | Query complete 00:00:00.137

c) CQL Create Statement:

```
CREATE TABLE Ads_q7c (
    ad_id         text,
    item_id       text,
    seller_user_id text,
    placed_date    date,
    PRIMARY KEY((seller_user_id), placed_date, ad_id)
) WITH CLUSTERING ORDER BY (placed_date DESC, ad_id ASC);
```

	ad_id [PK] text	item_id text	seller_user_id text	placed_date date
1	Y0CQ6	NKRP6	UEAU3	2022-02-24
2	T4WN6	QVT8S	XWXLZ	2022-08-26
3	GQ8JT	KIE8I	JZ1WL	2022-05-10
4	V49GW	OEYRT	3DEPB	2022-07-17
5	N9427	TZZSO	ZXBS8	2022-02-01
6	9AFZA	GOA1X	MK3V5	2022-03-15
7	VPUEM	41BHM	F98VS	2022-07-27
8	J82FF	9T4E4	BL4U8	2022-07-31
9	5GT9S	ZOXPM	LBHMK	2022-09-28
10	8N9S2	3KPDQ	XU609	2022-08-10
11	YUPSL	Q0D8Y	61A69	2022-05-31

Total rows: 1000 of 4436 Query complete 00:00:00.100

d) CQL Create Statement:

```
CREATE TABLE Ratings_q7d (
    buyer_id     text,
    seller_id    text,
    quality      int,
    rating_date  date,
    PRIMARY KEY ((seller_id), rating_date, buyer_id)
);
```

	buyer_id [PK] text	seller_id [PK] text	quality integer	rating_date date
1	Y0FAM	E9K7C	[null]	2022-05-30
2	Y0FAM	O9NN6	[null]	2022-10-22
3	Y0FAM	LZLDV	3	2022-06-13
4	55M8O	TH583	[null]	2022-01-16
5	55M8O	4CXZQ	[null]	2022-04-11
6	55M8O	U29GK	[null]	2022-07-18
7	DG4JB	MQBSG	[null]	2022-07-08
8	DG4JB	CJZRT	[null]	2022-06-07
9	DG4JB	EGA3T	[null]	2022-04-28
10	CT9EA	H2NH0	2	2022-07-05
11	CT9EA	N7TIV	[null]	2022-05-20

Total rows: 1000 of 2127 Query complete 00:00:00.102

8.

a)

- CQL Query:

```
SELECT item_id, price FROM items_q7a WHERE seller_user_id = '67EYU' ORDER BY price DESC;
```

- Result:

```
 item_id    |   price
--------------+------------
   6N0EN   | 1669.16
   6WBQ4  | 1474.38
   6TI88     | 1298.58
   KS6NE    | 1002.33
   M4FM5  |  934.23
   KUPJA     |  736.35
   9HCUN   |  703.40
   3SY96     |  697.96
   AVLT5     |  212.50
   NO7E9   |  101.84
   78R60    |   53.77
   UDROS   |    8.80
(12 rows)
```

b)

- CQL Query:

```
SELECT COUNT(buyer_user_id) as item_count FROM items_q7b WHERE category = 'Electronics';
```

- Result:

```
 item_count
------------
         98
```

c)

- CQL Query:

```
SELECT item_id, ad_id FROM ads_q7c WHERE seller_user_id = 'DNCLE' ORDER BY placed_date DESC;
```

- Result:

```
 item_id   | ad_id
-------------+----------
  THJKW  | H4USM
  ANXVB | 7VAFG
  THJKW | B4IAF
  CEOIC   | P29GP
  5JZ1Z    | QMPFC
  OAJWJ  | 5IO7K
  J4AC4   | 7BPAZ
  LXUHH | 6CZ8H
  MFJHF  | A2E6V
  ANXVB | IJG9B

(10 rows)
```

d)

- CQL Query:

```
SELECT AVG(CAST(quality as float))
FROM ratings_q7d
WHERE seller_id = 'AWFGJ' and rating_date >= '2022-01-01' and rating_date <= '2022-10-01';
```

- Result:

```
system.avg(cast(quality as float))
-------------------------------------
                              3.5
```

9.

a) CQL INSERT statements:

```
INSERT INTO ratings(buyer_id, seller_id, quality, pricing, delivery, rating_date)
VALUES('LFIR9', 'AWFGJ', 3, 4, 5, '2022-02-10');

INSERT INTO ratings_q7d(buyer_id, seller_id, quality, rating_date)
VALUES('LFIR9', 'AWFGJ', 3, '2022-02-10');
```

10.

Python script:

```
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider

cloud_config = {'secure_connect_bundle': '/Applications/Homework2/secure-connect-cs122d-spring.zip'}
auth_provider = PlainTextAuthProvider('fwuZyErYDdLaUemcsUnyYKbv', 'tPZbiKNJsq_DrP8j2s60c.OiJ_DUPkH+ehDyiL8rkaYtPfly4pLcL6N4t2QQO.jcz50S347odgEKT2nv5KfSoqQ5tBinQEdEI6v,sGH-Uq8Tb3YcsKpSCRsZfItqZzRR')
cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
session = cluster.connect()

def addTo_ratings(buyer_id, seller_id, quality, pricing, delivery, rating_date):
    query1 = f"INSERT INTO interchange.ratings (buyer_id, seller_id, quality, pricing, delivery, rating_date) \
            VALUES ('{buyer_id}', '{seller_id}', {quality}, {pricing}, {delivery}, '{rating_date}')"
    session.execute(query1)
    print("Inserts added successfully to Ratings table")

def addTo_ratings_q7d(buyer_id, seller_id, quality, rating_date):
    query2 = f"INSERT INTO interchange.ratings_q7d (buyer_id, seller_id, quality, rating_date) \
```

```
        VALUES ('{buyer_id}', '{seller_id}', {quality}, '{rating_date}')"
    session.execute(query2)
    print("Inserts added successfully to Ratings_q7d table")

addTo_ratings('LFIR9', 'AWFGJ', 3, 4, 5, '2022-02-10')
addTo_ratings7d('LFIR9', 'AWFGJ', 3, '2022-02-10')

row = session.execute("select release_version from system.local").one()
if row:
    print(row[0])
else:
    print("An error occurred.")
```