Last Name: Bi First Name: Xingjian Student ID: 1096970

1.

a) CQL Query:

DESCRIBE "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:

The Hoofers keyspace maintains 3 copies of data using the NetworkTopologyStrategy replication strategy in the 'us-east1' cloud region. Assuming a replication factor of 3, for strong consistency, R = 2 and W = 2.

2.

a) CQL CREATE Statements:

CREATE TABLE interchange.Users (

user\_id text PRIMARY KEY,

email text,

first\_name text,

last\_name text,

joined\_date date,

street text,

city text,

state text,

zip text,

categories text

);

CREATE TABLE interchange.Items (

item\_id text PRIMARY KEY,

name text,

price decimal,

category text,

description text,

seller\_user\_id text,

list\_date date,

buyer\_user\_id text,

purchase\_date date,

) ;

CREATE TABLE interchange.Ads (

ad\_id text PRIMARY KEY,

plan text,

content text,

pic\_num int,

item\_id text,

seller\_user\_id text,

placed\_date date

);

CREATE TABLE interchange.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 '/Users/bixingjian/Nutstore Files/Nutstore/UCI 2023S/CS122D/HW2/users.csv' WITH(FORMAT CSV, HEADER)

\COPY interchange.Items TO '/Users/bixingjian/Nutstore Files/Nutstore/UCI 2023S/CS122D/HW2/items.csv' WITH(FORMAT CSV, HEADER)

\COPY interchange.Ads TO '/Users/bixingjian/Nutstore Files/Nutstore/UCI 2023S/CS122D/HW2/ads.csv' WITH(FORMAT CSV, HEADER)

\COPY interchange.Ratings TO '/Users/bixingjian/Nutstore Files/Nutstore/UCI 2023S/CS122D/HW2/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';

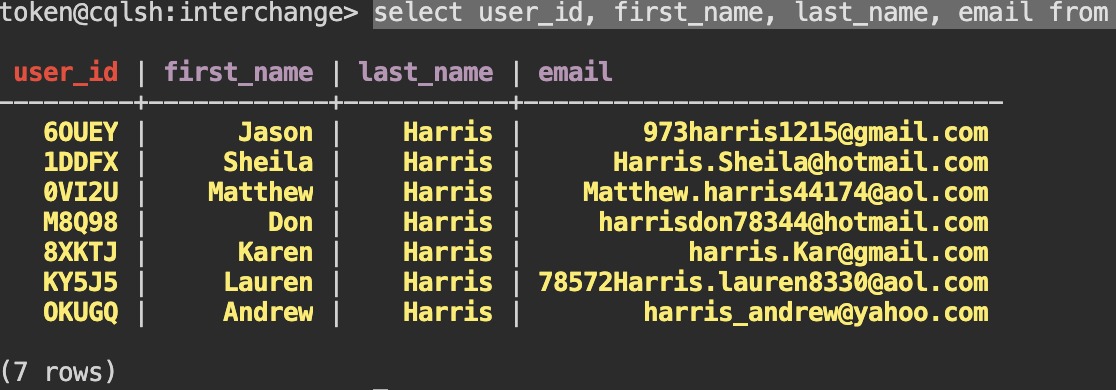
b) Result:

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:



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,

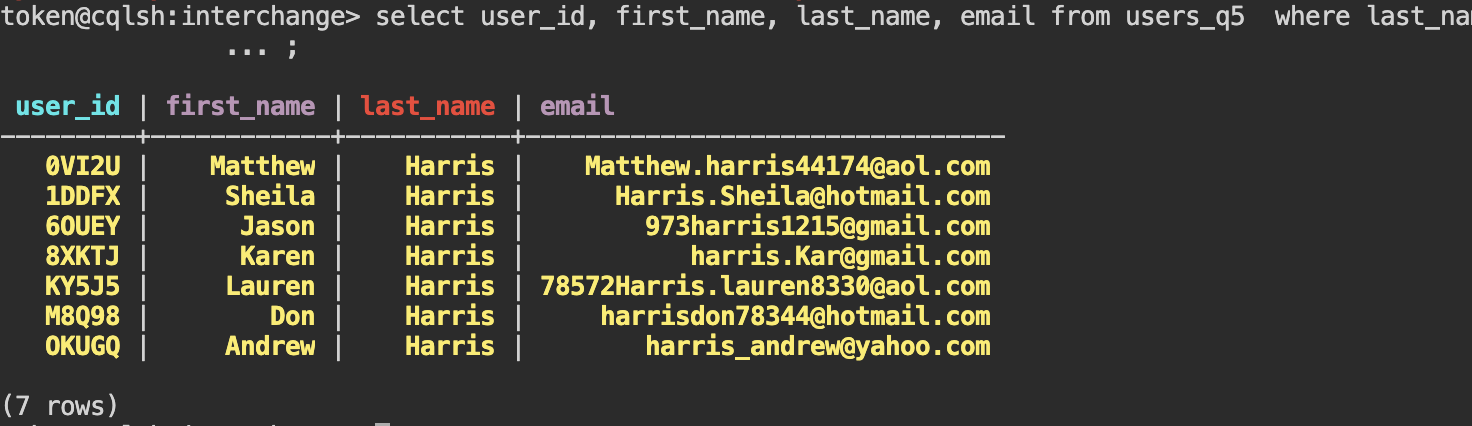
PRIMARY KEY (last\_name, user\_id)

);

b) CQL Query:

select user\_id, first\_name, last\_name, email from users\_q5 where last\_name='Harris' allow filtering

c) Result:



d) Explanation:

Because last\_name is not unique across all users. There may be multiple users with the same last\_name, so we need to include user\_id to make the primary key unique. The primary key is used to locate the data, so it must be unique.

6.

a) CQL Query:

select first\_name, last\_name, email, user\_id from users where last\_name = 'Davis' order by joined\_date desc limit 10 allow filtering;

select first\_name, last\_name, email, user\_id from users\_q5 where last\_name = 'Davis' order by joined\_date desc limit 10 allow filtering;

select first\_name,last\_name,email,user\_id from users\_q6 where last\_name = 'Davis' 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)

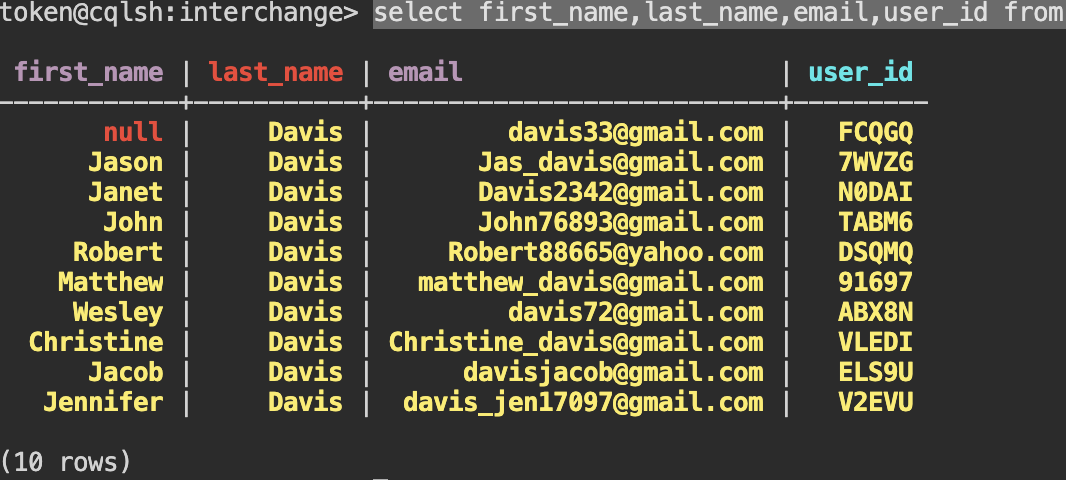
)

WITH CLUSTERING ORDER BY (joined\_date DESC,user\_id DESC);

c) Results:

InvalidRequest: Error from server: code=2200 [Invalid query] message="ORDER BY is only supported when the partition key is restricted by an EQ or an IN."

InvalidRequest: Error from server: code=2200 [Invalid query] message="Order by is currently only supported on the clustered columns of the PRIMARY KEY, got joined\_date"



d) Explanation:

The ordering of the clustering keys matter as they determine the order of the results within each partition. We specify the order, in each partition, each information will be sorted by joined\_date first and then user\_id.

7.

a) CQL Create Statement:

CREATE TABLE items\_7a (

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 DESC);

b) CQL Create Statement:

CREATE TABLE items\_7b (

category text,

item\_id text,

PRIMARY KEY (category, item\_id) )

c) CQL Create Statement:

CREATE TABLE ads\_7c (

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 DESC);

d) CQL Create Statement:

CREATE TABLE ratings\_7d (

buyer\_id text,

seller\_id text,

quality int,

rating\_date date,

PRIMARY KEY (seller\_id,rating\_date,buyer\_id) )

WITH CLUSTERING ORDER BY (rating\_date DESC, buyer\_id DESC);

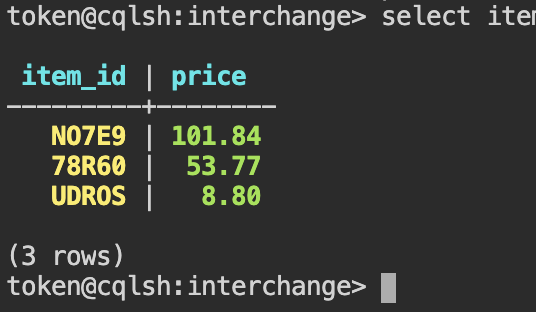
8.

a)

* CQL Query:

select item\_id, price from items\_7a where seller\_user\_id ='67EYU' order by price desc;

* Result:

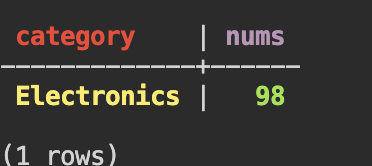


b)

* CQL Query:

select category, count(\*) as nums from items\_7b where category ='Electronics';

* Result:

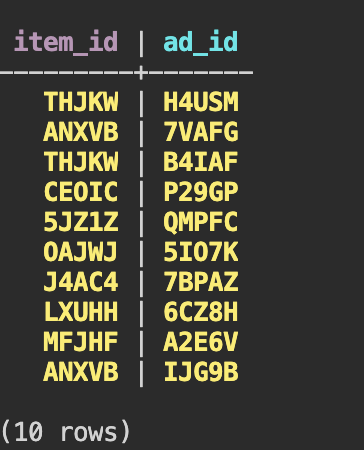


c)

* CQL Query:

select item\_id,ad\_id from ads\_7c where seller\_user\_id = 'DNCLE' order by placed\_date desc;

* Result:

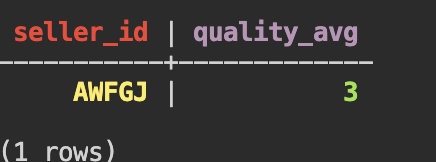


d)

* CQL Query:

select seller\_id, avg(quality) as quality\_avg from ratings\_7d where seller\_id = 'AWFGJ' and rating\_date <='2022-10-01' and rating\_date >='2022-01-01';

* Result:



9.

a) CQL INSERT statements:

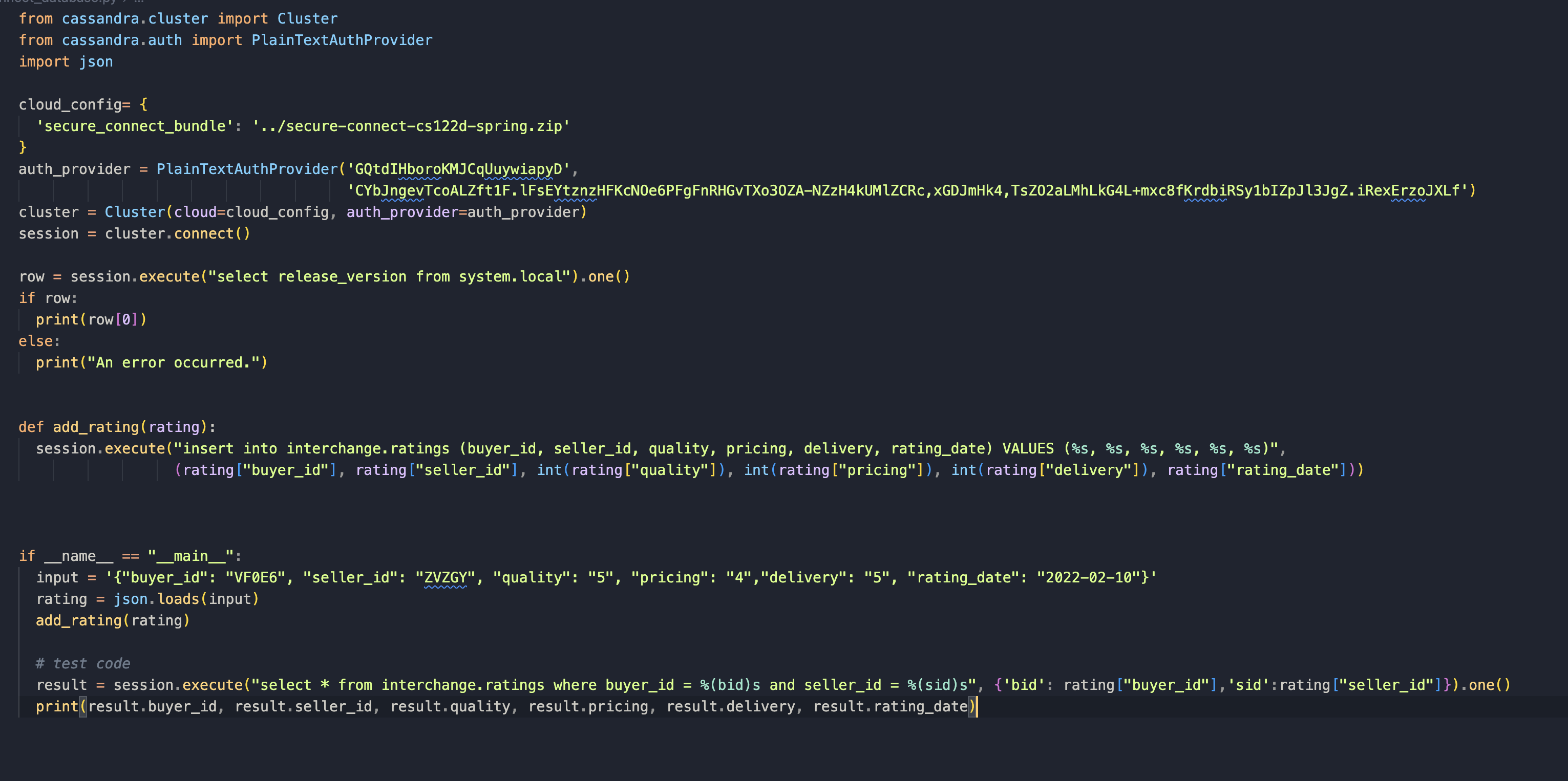
insert into "ratings" (buyer\_id, seller\_id, delivery, pricing, quality, rating\_date) values ('LFIR9', 'AWFGJ', 5, 4, 3, '2022-02-10');

insert into "ratings\_7d" (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

import json

cloud\_config= {

'secure\_connect\_bundle': '../secure-connect-cs122d-spring.zip'

}

auth\_provider = PlainTextAuthProvider('GQtdIHboroKMJCqUuywiapyD',

'CYbJngevTcoALZft1F.lFsEYtznzHFKcNOe6PFgFnRHGvTXo3OZA-NZzH4kUMlZCRc,xGDJmHk4,TsZO2aLMhLkG4L+mxc8fKrdbiRSy1bIZpJl3JgZ.iRexErzoJXLf')

cluster = Cluster(cloud=cloud\_config, auth\_provider=auth\_provider)

session = cluster.connect()

row = session.execute("select release\_version from system.local").one()

if row:

print(row[0])

else:

print("An error occurred.")

def add\_rating(rating):

session.execute("insert into interchange.ratings (buyer\_id, seller\_id, quality, pricing, delivery, rating\_date) VALUES (%s, %s, %s, %s, %s, %s)",

(rating["buyer\_id"], rating["seller\_id"], int(rating["quality"]), int(rating["pricing"]), int(rating["delivery"]), rating["rating\_date"]))

if \_\_name\_\_ == "\_\_main\_\_":

input = '{"buyer\_id": "VF0E6", "seller\_id": "ZVZGY", "quality": "5", "pricing": "4","delivery": "5", "rating\_date": "2022-02-10"}'

rating = json.loads(input)

add\_rating(rating)

# test code

result = session.execute("select \* from interchange.ratings where buyer\_id = %(bid)s and seller\_id = %(sid)s", {'bid': rating["buyer\_id"],'sid':rating["seller\_id"]}).one()

print(result.buyer\_id, result.seller\_id, result.quality, result.pricing, result.delivery, result.rating\_date)