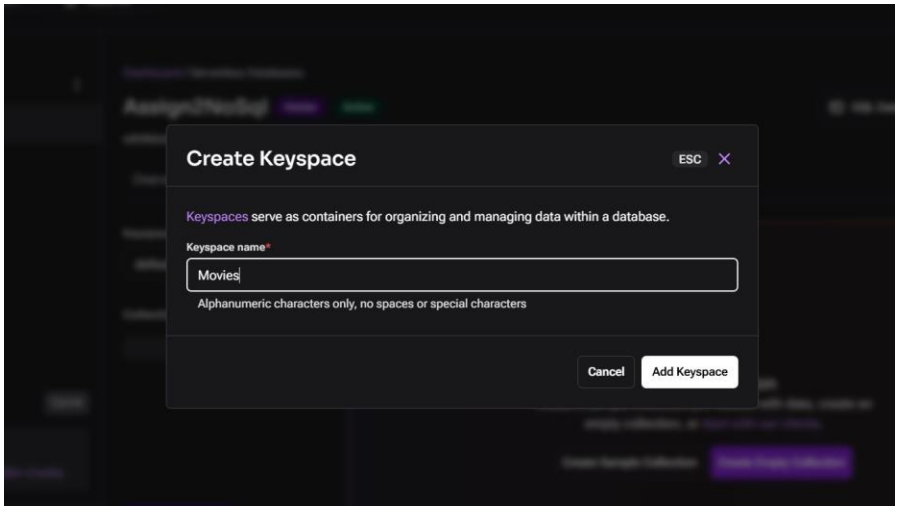
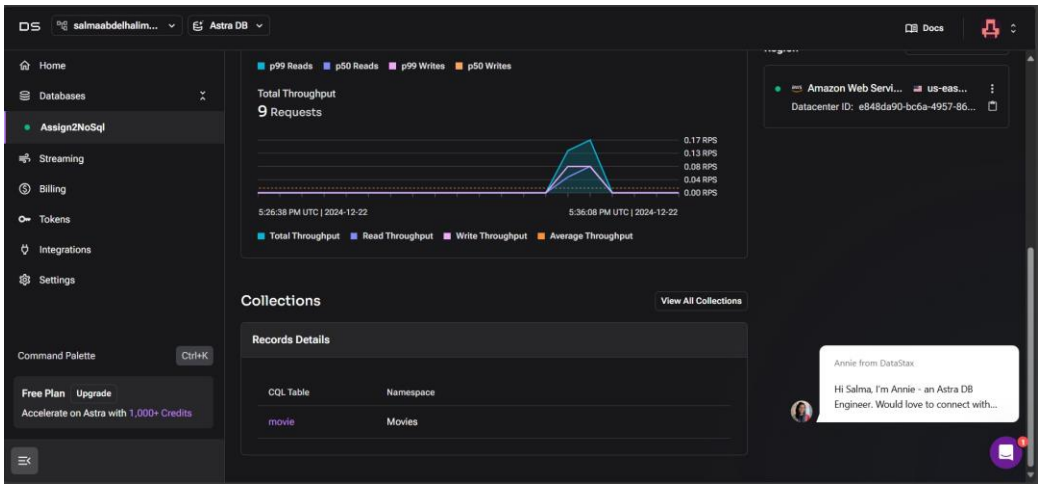
Assignment 2: Cassandra DB

1. Create a KeySpace:






```
Connected as salmaabdelhalim005zidan@gmail.com.
Connected to cndb at cassandra.ingress:9042.
[cqlsh 6.8.0 | Cassandra 4.0.0.6816 | CQL spec 3.4.5 | Native protocol v4 | TLS]
Use HELP for help.
token@cqlsh> describe keyspaces;

data_endpoint_auth  default_keyspace  system_traces  system_virtual_schema
system_auth         datastax_sla      "Movies"
system_schema       system            system_views

token@cqlsh>
```

2. Create a column-family:

```
CREATE TABLE Movie (
id int PRIMARY KEY,
name text,
movie_cast map<text, text>,
movie_poster blob,
created_at timestamp);
```

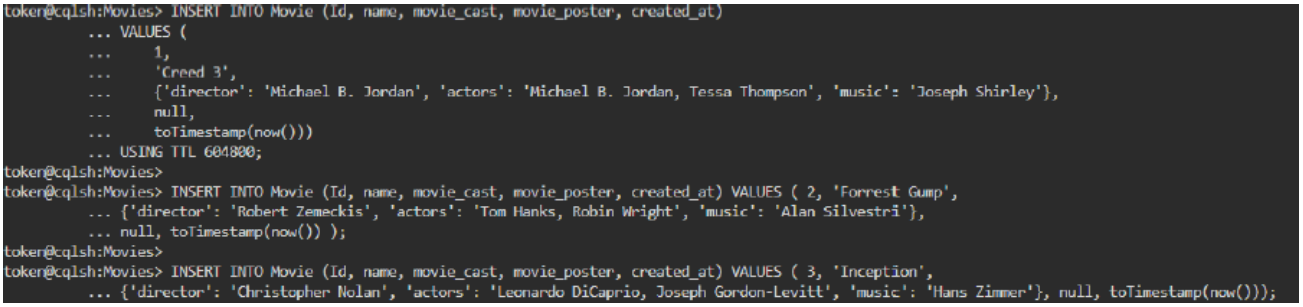
```
token@cqlsh:Movies> CREATE TABLE Movie (
           ...     id int PRIMARY KEY,
           ...     name text,
           ...     movie_cast map<text, text>,
           ...     movie_poster blob,
           ...     created_at timestamp
           ... );
```

3. Check the schema:

```
oken@cqlsh:Movies> describe Movie;

REATE TABLE "Movies".movie (
    id int PRIMARY KEY,
    created_at timestamp,
    movie_cast map<text, text>,
    movie_poster blob,
    name text
) WITH additional_write_policy = '99p'
    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': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 604800
    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 = '99p';

oken@cqlsh:Movies>
```

4. Populate the Movie table with 3 real movies:

```
INSERT INTO Movie (Id, name, movie_cast, movie_poster, created_at)
VALUES (
    1,
    'Creed 3',
    {'director': 'Michael B. Jordan', 'actors': 'Michael B. Jordan, Tessa Thompson',
'music': 'Joseph Shirley'},
    null,
    toTimestamp(now()))
USING TTL 604800;

INSERT INTO Movie (Id, name, movie_cast, movie_poster, created_at)
VALUES ( 2,  'Forrest Gump',
    {'director': 'Robert Zemeckis', 'actors': 'Tom Hanks, Robin Wright', 'music': 'Alan
Silvestri'},
    null, toTimestamp(now()) );

INSERT INTO Movie (Id, name, movie_cast, movie_poster, created_at)
VALUES ( 3, 'Inception',
    {'director': 'Christopher Nolan', 'actors': 'Leonardo DiCaprio, Joseph Gordon-Levitt',
'music': 'Hans Zimmer'}, null, toTimestamp(now()));
```

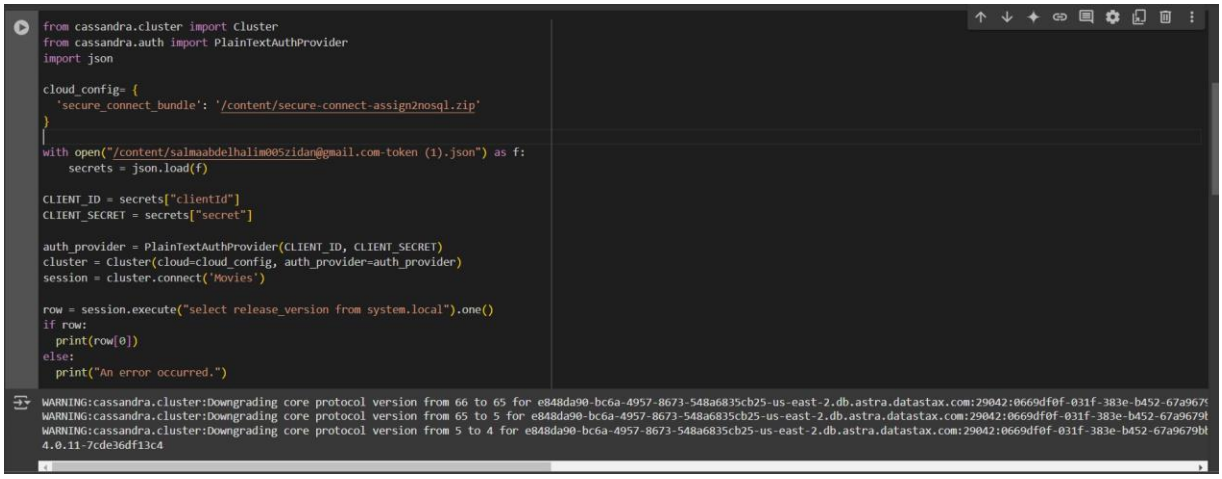


5. The Python function connects to KeySpace transforms the movie poster into a blob datatype and updates the movie-poster.

5.1. Connect to the keySpace(Movies):

connects to KeySpace:

```
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider
import json
cloud_config= {
  'secure_connect_bundle': '/content/secure-connect-assign2nosql.zip'}
with open("/content/salmaabdelhalim005zidan@gmail.com-token (1).json") as f:
    secrets = json.load(f)
CLIENT_ID = secrets["clientId"]
CLIENT_SECRET = secrets["secret"]
auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)
cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
session = cluster.connect('Movies')
row = session.execute("select release_version from system.local").one()
if row:
  print(row[0])
else:
  print("An error occurred.")
```

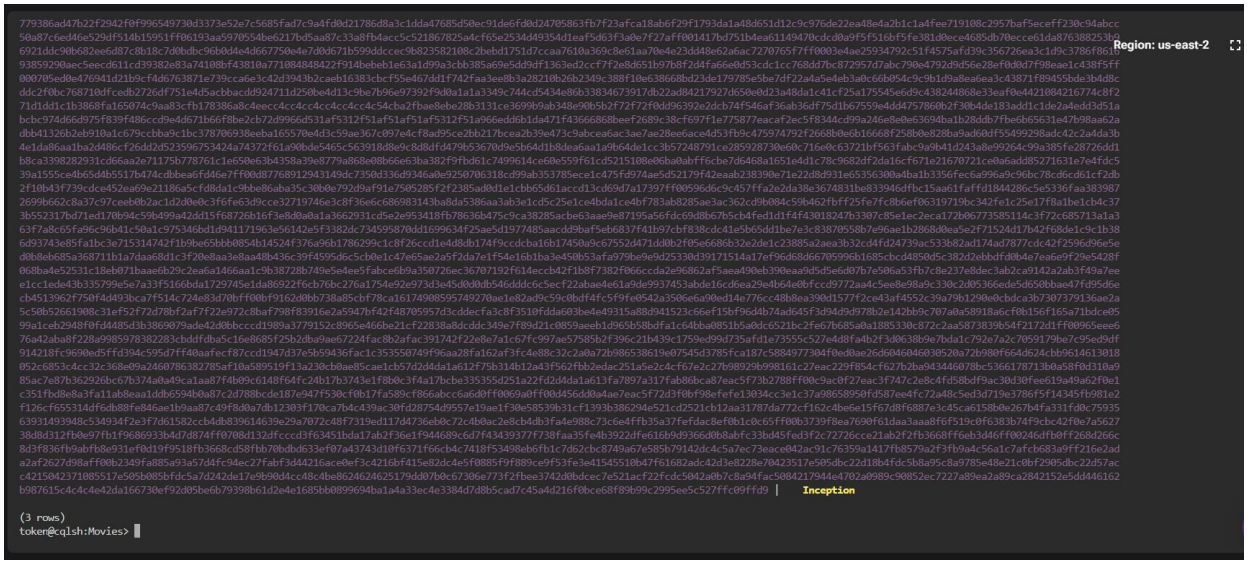


5.2. updates the movie-poster column:

```
updates the movie-poster column:

import os
def image_to_blob(image_path):
    """
    Reads an image and converts it to a binary blob.
    """
    with open(image_path, "rb") as file:
        return file.read()
def update_movie_posters(session, folder_path):
    try:
        poster_files = [f for f in os.listdir(folder_path) if f.endswith(('jpg', 'jpeg', 'png'))]
        rows = session.execute("SELECT Id, name FROM Movie").all()
        if len(poster_files) < len(rows):
            print(f"Not enough posters to match with the movies.")
            Return
        for (movie_id, movie_name), poster_file in zip(rows, poster_files):
            poster_path = os.path.join(folder_path, poster_file)
            poster_blob = image_to_blob(poster_path)
            query = "UPDATE Movie SET movie_poster = %s WHERE Id = %s"
            session.execute(query, (poster_blob, movie_id))
            print(f"Successfully updated the poster for {movie_name} with {poster_file}")
    except Exception as e:
        print(f"An error occurred while updating posters: {str(e)}")
folder_path = r"C:\\Users\\lenovo\\Desktop\\Posters"
update_movie_posters(session, folder_path)
```

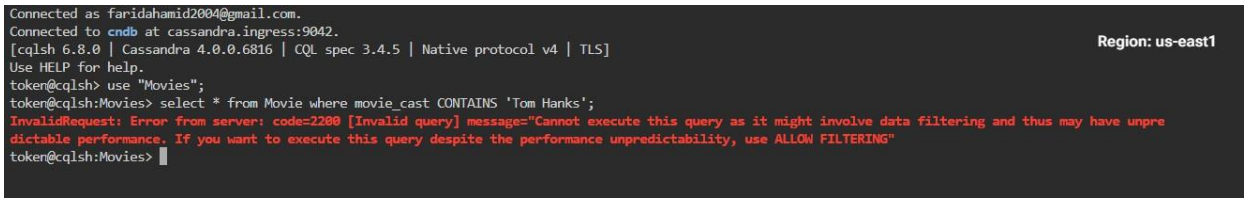
```python
import os
def image_to_blob(image_path):
    """
    Reads an image and converts it to a binary blob.
    """
    with open(image_path, "rb") as file:
        return file.read()
def update_movie_posters(session, movie_files):
    try:
        rows = session.execute("SELECT Id, name FROM Movie").all()
        if len(movie_files) < len(rows):
            print(f"Not enough posters to match with the movies.")
            return
        for (movie_id, movie_name), poster_path in zip(rows, movie_files):
            poster_blob = image_to_blob(poster_path)
            query = "UPDATE Movie SET movie_poster = %s WHERE Id = %s"
            session.execute(query, (poster_blob, movie_id))
            print(f"Successfully updated the poster for {movie_name} with {os.path.basename(poster_path)}")
    except Exception as e:
        print(f"An error occurred while updating posters: {str(e)}")
movie_files = [
    r"/content/Creed_III_poster.png",
    r"/content/Forest Gump.jpeg",
    r"/content/Inception.jpg"
]
update_movie_posters(session, movie_files)
```

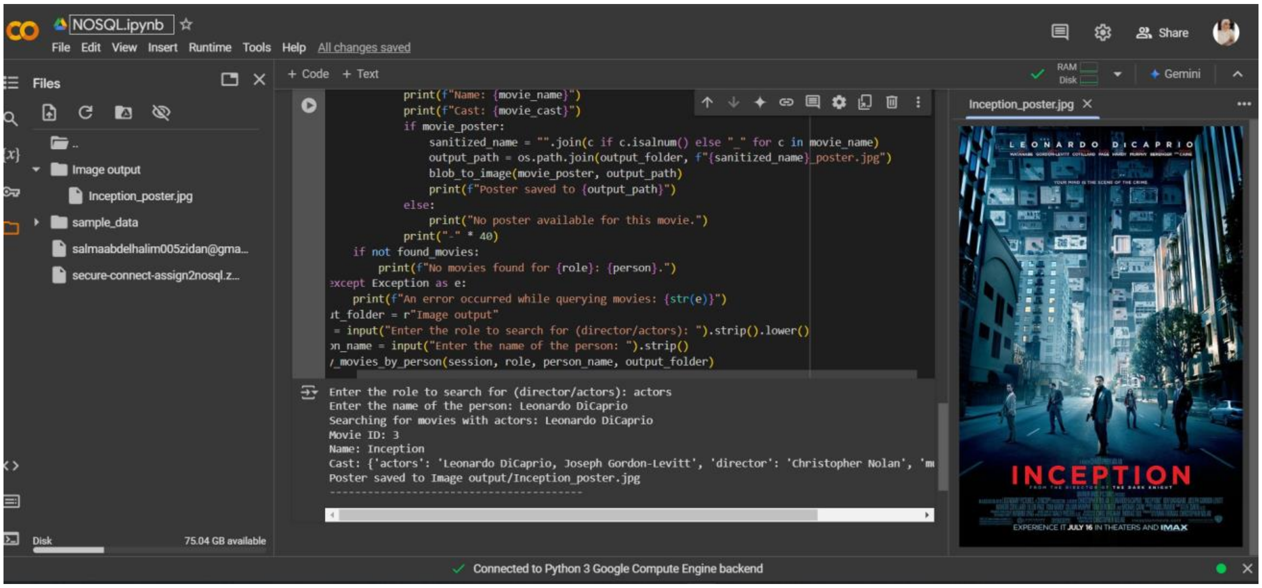
```
Successfully updated the poster for Creed 3 with Creed_III_poster.png
Successfully updated the poster for Forrest Gump with Forest Gump.jpeg
Successfully updated the poster for Inception with Inception.jpg
```



6. python function to query the movies given a certain director or actor:

```
import os
def blob_to_image(blob_data, output_path):
    with open(output_path, "wb") as file:
        file.write(blob_data)
def query_movies_by_person(session, role, person, output_folder):
    try:
        print(f"Searching for movies with {role}: {person}")
        query = "SELECT * FROM Movie ALLOW FILTERING;"
        rows = session.execute(query)
        os.makedirs(output_folder, exist_ok=True)
        found_movies = False
        for row in rows:
            movie_id = row.id
            movie_name = row.name
            movie_cast = row.movie_cast
            movie_poster = row.movie_poster
            if role in movie_cast and person in movie_cast[role]:
                found_movies = True
                print(f"Movie ID: {movie_id}")
                print(f"Name: {movie_name}")
                print(f"Cast: {movie_cast}")
                if movie_poster:
                    sanitized_name = "".join(c if c.isalnum() else "_" for c in movie_name)
                    output_path = os.path.join(output_folder, f"{sanitized_name}_poster.jpg")
                    blob_to_image(movie_poster, output_path)
                    print(f"Poster saved to {output_path}")
                else:
                    print("No poster available for this movie.")
                print("-" * 40)
        if not found_movies:
            print(f"No movies found for {role}: {person}.")
    except Exception as e:
        print(f"An error occurred while querying movies: {str(e)}")
output_folder = r"Image output"
role = input("Enter the role to search for (director/actors): ").strip().lower()
person_name = input("Enter the name of the person: ").strip()
query_movies_by_person(session, role, person_name, output_folder)
```





7. Update a certain movie actors list to add another actor to the list.

UPDATE Movie SET movie_cast = movie_cast + {'actors': 'Michael B. Jordan, Tessa Thompson, Jonathan Majors'} WHERE Id = 1;

```
token@cqlsh> use "Movies";
token@cqlsh:Movies> UPDATE Movie
        ... SET movie_cast = movie_cast + {'actors': 'Michael B. Jordan, Tessa Thompson, Jonathan Majors'}
        ... WHERE Id = 1;
token@cqlsh:Movies> select Id, name, movie_cast, created_at from movie;

 id | name         | movie_cast                                                                                               | created_at
----+--------------+----------------------------------------------------------------------------------------------------------+---------------------------------
  1 |      Creed 3 | {'actors': 'Michael B. Jordan, Tessa Thompson, Jonathan Majors', 'director': 'Michael B. Jordan', 'music': 'Joseph Shirley'} | 2024-12-18 07:50:40.775000+0000
  2 | Forrest Gump |                          {'actors': 'Tom Hanks, Robin Wright', 'director': 'Robert Zemeckis', 'music': 'Alan Silvestri'} | 2024-12-18 07:50:40.782000+0000
  3 |    Inception |            {'actors': 'Leonardo DiCaprio, Joseph Gordon-Levitt', 'director': 'Christopher Nolan', 'music': 'Hans Zimmer'} | 2024-12-18 07:50:42.266000+0000

(3 rows)
token@cqlsh:Movies>
```

8. Update the first row TTL to 3 seconds:

INSERT INTO Movie (Id, name, movie_cast, movie_poster, created_at) VALUES (1, 'Creed 3', {'director': 'Michael B. Jordan', 'actors': 'Michael B. Jordan, Tessa Thompson, Jonathan Majors', 'music': 'Joseph Shirley'}, 0x48656c6f2c20576f726c6421 ,toTimestamp(now())) USING TTL 3;

```
token@cqlsh:Movies> INSERT INTO Movie (Id, name, movie_cast, movie_poster, created_at) VALUES ( 1, 'Creed 3', {'director': 'Michael B. Jordan', 'actors': 'Michael B. Jordan, Tessa Thompson, Jonathan Majors', 'music': 'Joseph Shirley'}, 0x48656c6f2c20576f726c6421 ,toTimestamp(now()) ) USING TTL 3;
token@cqlsh:Movies> select Id, name, movie_cast, created_at from movie;

 id | name         | movie_cast                                                                                      | created_at
----+--------------+-------------------------------------------------------------------------------------------------+---------------------------------
  2 | Forrest Gump |              {'actors': 'Tom Hanks, Robin Wright', 'director': 'Robert Zemeckis', 'music': 'Alan Silvestri'} | 2024-12-18 07:50:40.782000+0000
  3 |    Inception | {'actors': 'Leonardo DiCaprio, Joseph Gordon-Levitt', 'director': 'Christopher Nolan', 'music': 'Hans Zimmer'} | 2024-12-18 07:50:42.266000+0000

(2 rows)
token@cqlsh:Movies>
```