Assignment 2: Cassandra DB

Github

PROJECT DOCUMENTATION

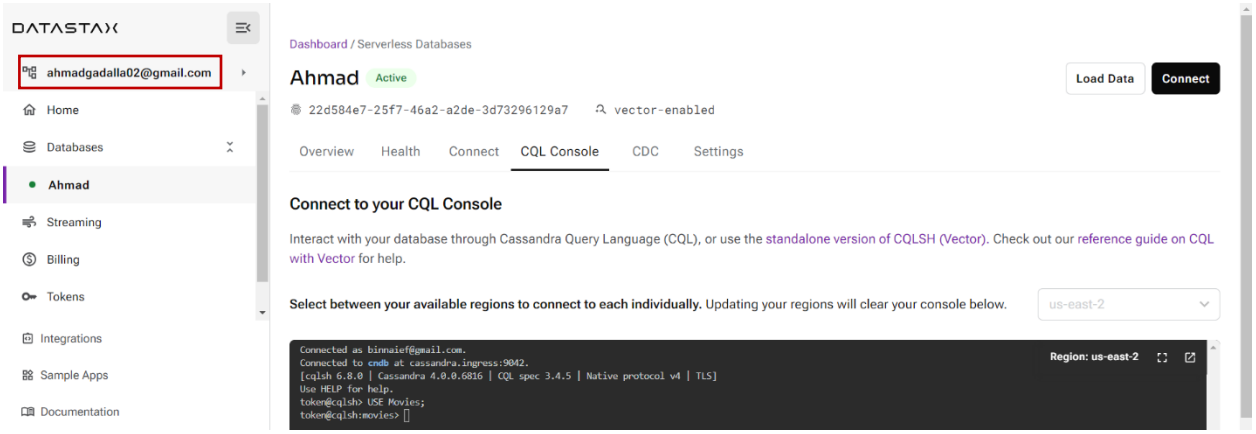
DATE: 25/12/2023

PROJECT TITLE: DataStax Astra & Python: Cassandra Showcase

OBJECTIVE: Utilizing DataStax Astra DB by executing key database tasks—creating keyspaces, populating data, and schema exploration. Additionally, aiming to demonstrate Python integration for image-to-blob transformations, querying based on directors/actors, and assessing TTL's impact on data persistence in Cassandra.

GIVEN TASKS:

1. **Create a Keyspace:**
    - Name: "Movies"
    - Replication factor: 1
    - Strategy: Simple strategy
    - Screenshot required with your email visible.



2. **Create a Column-Family "Movie":**

- Columns: Id (int), name (text), movie-cast (map), movie-poster (blob)

```
CREATE TABLE movies.movie (
    id int PRIMARY KEY,
    movie_cast map<text, text>,
    movie_poster blob,
    name text
)
```

- Set the first row TTL to 7 days.

```
USING TTL 604800;
```

---

3. **Check the schema of the "Movie" column-family.**

```
Select * from movie;
```

```
token@cqlsh:movies> select * from movie;

 id | movie_cast | movie_poster | name
----+------------+--------------+------

(0 rows)
```

```
describe movie;
```

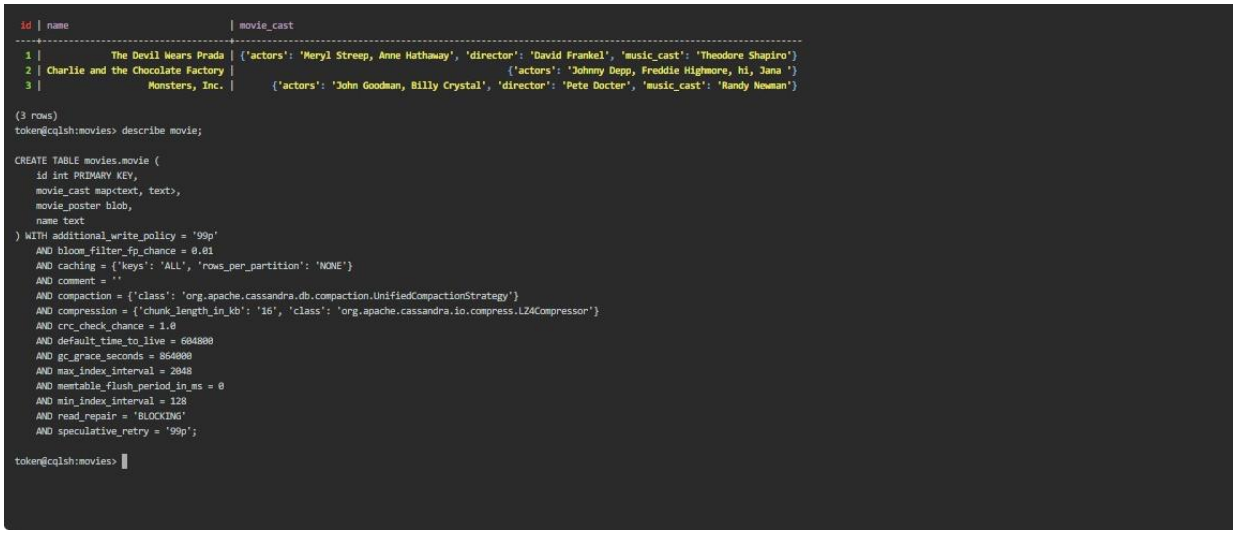
4. **Populate the Movie table:**

- Insert data for 3 movies (ID, name, movie-cast, excluding movie-poster).

```
USE Movies;

INSERT INTO movie (id, movie_cast, movie_poster, name) VALUES
(1, {'director': 'David Frankel', 'actors': 'Meryl Streep, Anne
Hathaway', 'music_cast': 'Theodore Shapiro'}, null, 'The Devil Wears
Prada');

INSERT INTO movie (id, movie_cast, movie_poster, name) VALUES
(2, {'director': 'Tim Burton', 'actors': 'Johnny Depp, Freddie
Highmore', 'music_cast': 'Danny Elfman'}, null, 'Charlie and the
Chocolate Factory');

INSERT INTO movie (id, movie_cast, movie_poster, name) VALUES
(3, {'director': 'Pete Docter', 'actors': 'John Goodman, Billy
Crystal', 'music_cast': 'Randy Newman'}, null, 'Monsters, Inc.');
```

- Check the schema.

```
Select id, name, movie_cast, from movie;
```

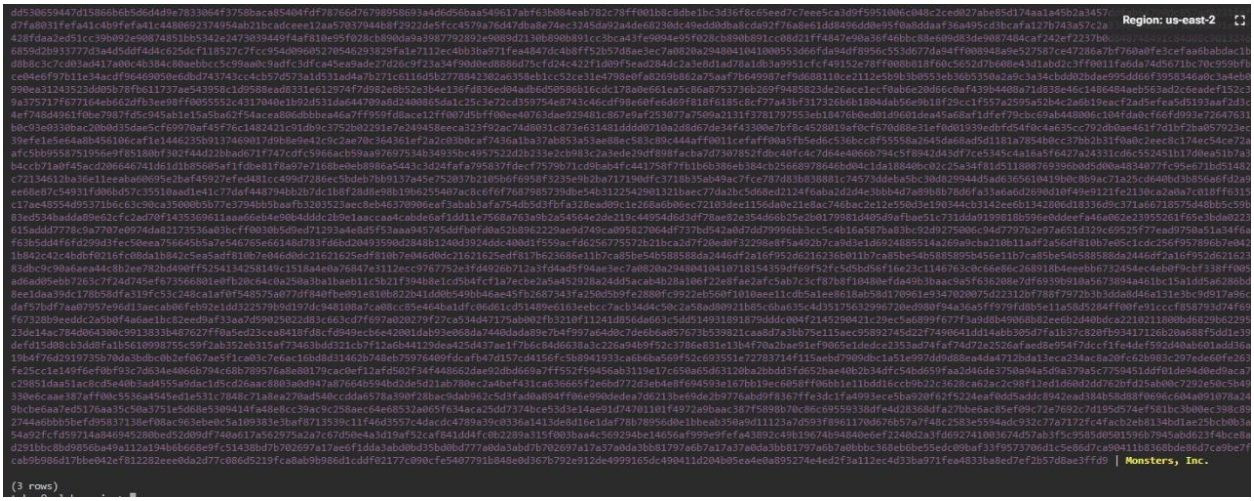


5. **Write a Python function to update movie-poster column:**

   - Function connects to Keyspace, transforms movie posters (from a local folder) to blob datatype, and updates the movie-poster column for the 3 inserted rows.

```
def connect_and_update_blob(folder_path):
    # Load credentials from JSON file
    with open("ahmadgadalla02@gmail.com-token.json") as f:
        secrets = json.load(f)

    CLIENT_ID = secrets["clientId"]
    CLIENT_SECRET = secrets["secret"]

    # Connection setup using secure connect bundle
    cloud_config = {
        'secure_connect_bundle': 'secure-connect-ahmad.zip'
    }

    auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)

    # Create the cluster and session
    cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
    session = cluster.connect()

    # Keyspace to use
    keyspace_name = "movies"

    # Use the keyspace
    session.set_keyspace(keyspace_name)

    # Get all files in the folder
    files = os.listdir(folder_path)[:3]  # Select the first three
files in the folder

    # Update movie-poster column for the three inserted rows
    for idx, file_name in enumerate(files, 1):
        # Read the image as binary data
        with open(os.path.join(folder_path, file_name), 'rb') as
image_file:
            image_data = image_file.read()

        # Update the movie-poster column for the specific row
        query = f"UPDATE movie SET movie_poster = %s WHERE id = {idx}"
        session.execute(query, (image_data,))  # Assuming 'movie'
table and 'id' column

    cluster.shutdown()
```

```
Select * from movie;
```



---

6. **A Python function to query movies based on name/director/actor:**

   - Discuss the feasibility of using "Like" or "Contains" CQL operators for this query.

   In Cassandra, direct 'LIKE' or 'CONTAINS' operators aren't supported. Instead, the 'ALLOW FILTERING' option enables queries on non-indexed columns like name, director, or actor. A Python function is used:

```
SELECT * FROM movie ALLOW FILTERING;
```

   to perform queries based on these criteria, compensating for the absence of traditional string-matching operators.

   - Function takes **movie name input**, queries the DB, displays the row results including the image.

```
def query_and_display_movies_name(search_term, output_folder):
    with open("ahmadgadalla02@gmail.com-token.json") as f:
        secrets = json.load(f)

    CLIENT_ID = secrets["clientId"]
    CLIENT_SECRET = secrets["secret"]
```

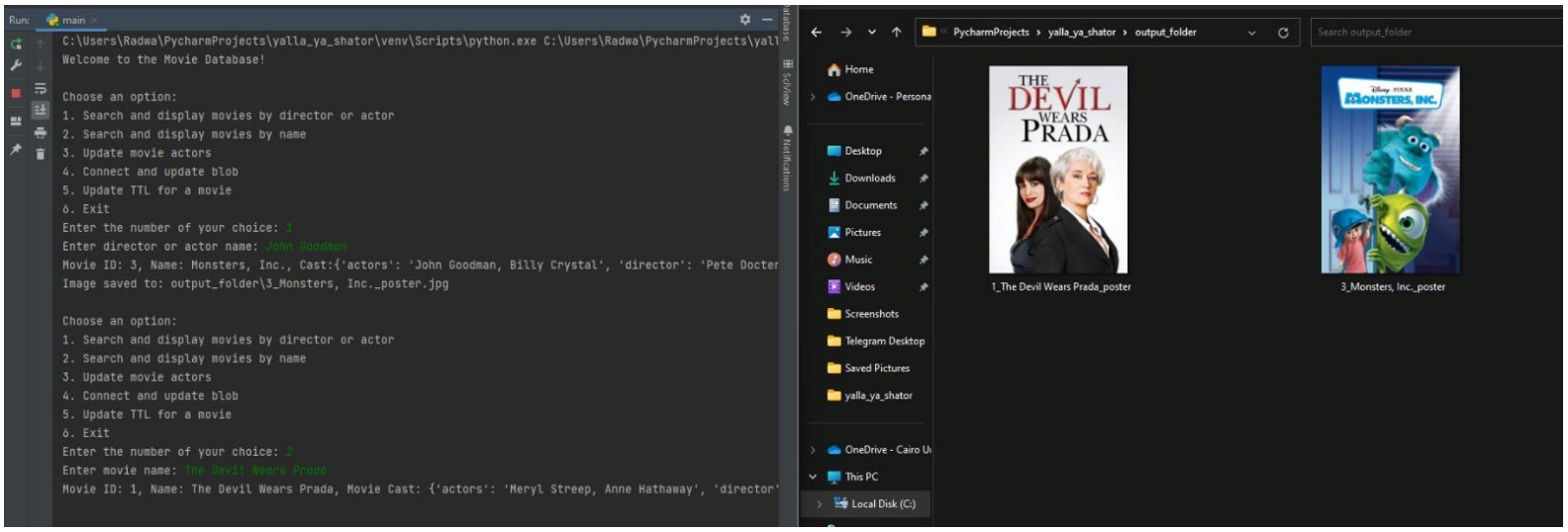
```
    cloud_config = {
        'secure_connect_bundle': 'secure-connect-ahmad.zip'
    }

    auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)

    cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
    session = cluster.connect()

    try:
        keyspace_name = "movies"

        session.set_keyspace(keyspace_name)

        query = f"SELECT id, movie_cast,movie_poster,name FROM movie
WHERE name = '{search_term}' ALLOW FILTERING;"

        rows = session.execute(query)
        os.makedirs(output_folder, exist_ok=True)

        for row in rows:
            print(f"Movie ID: {row.id}, Name: {row.name}, Movie Cast:
{row.movie_cast}\n")
            print(f"____________________")

            # Save the image to the output folder
            if row.movie_poster:
                image_data = row.movie_poster
                image = Image.open(BytesIO(image_data))
                image_path = os.path.join(output_folder,
f"{row.id}_{row.name}_poster.jpg")
                image.save(image_path)
                print(f"Image saved to: {image_path}")

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        cluster.shutdown()
```

- Function **director/actor input**, queries the DB, displays the row results including the image.

```
def query_and_display_movies_by_director_or_actor(search_term,
output_folder):

    with open("ahmadgadalla02@gmail.com-token.json") as f:
        secrets = json.load(f)

    CLIENT_ID = secrets["clientId"]
    CLIENT_SECRET = secrets["secret"]

    cloud_config = {
        'secure_connect_bundle': 'secure-connect-ahmad.zip'
    }

    auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)

    cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
    session = cluster.connect()

    try:
        keyspace_name = "movies"

        session.set_keyspace(keyspace_name)

        query = f"SELECT * FROM movie ALLOW FILTERING;"

        rows = session.execute(query)

        # Create the output folder if it doesn't exist
        os.makedirs(output_folder, exist_ok=True)

        for row in rows:
            movie_id = row.id
            movie_name = row.name
            movie_cast = row.movie_cast
            movie_cast = str(movie_cast)
            if search_term in movie_cast:
                movie_poster = row.movie_poster
                print(f"Movie ID: {movie_id}, Name: {movie_name},
Cast:{movie_cast}")
                image_data = row.movie_poster
                image = Image.open(BytesIO(image_data))
                image_path = os.path.join(output_folder,
f"{row.id}_{row.name}_poster.jpg")
```

```
                image.save(image_path)
                print(f"Image saved to: {image_path}")
    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        cluster.shutdown()
```

- Output Check.



---

7. **<u>Update a certain movie's actors list:</u>**

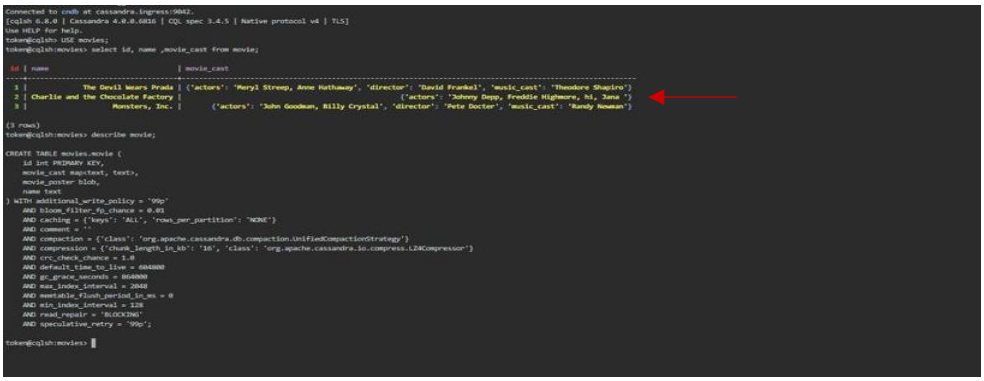
- Add another actor to the existing list.

```
def update_movie_actors(movie_id, new_actor):
    with open("ahmadgadalla02@gmail.com-token.json") as f:
        secrets = json.load(f)

    CLIENT_ID = secrets["clientId"]
    CLIENT_SECRET = secrets["secret"]
    cloud_config = {
        'secure_connect_bundle': 'secure-connect-ahmad.zip'
    }

    auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)

    cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
    session = cluster.connect()
```

```
    try:

        keyspace_name = "movies"

        session.set_keyspace(keyspace_name)

        select_query = f"SELECT movie_cast FROM movie WHERE id =
{movie_id};"
        result = session.execute(select_query).one()


        existing_actors_str = result.movie_cast.get('actors', '') if
result.movie_cast else ''

        updated_actors_str = f"{existing_actors_str}, {new_actor}" if
existing_actors_str else new_actor


        update_query = f"UPDATE movie SET movie_cast = {{ 'actors':
'{updated_actors_str}' }} WHERE id = {movie_id};"
        session.execute(update_query)

        print(f"Updated movie {movie_id} with {new_actor} added to the
actor list.")

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        cluster.shutdown()
```

- Check Schema.

```
select id, name, movie_cast from movie;
```



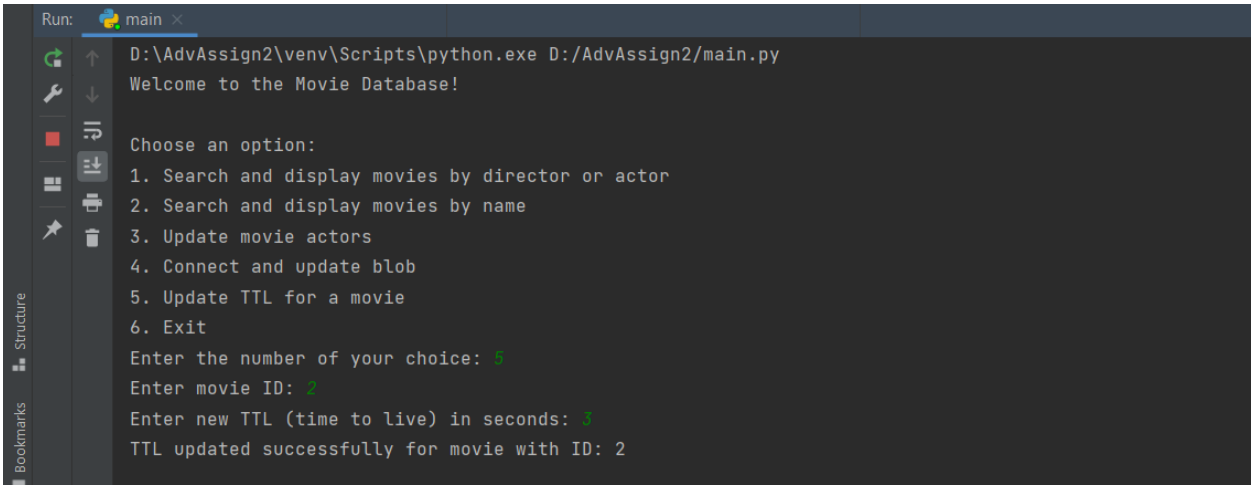
---

8. **<u>Update the first row TTL to 3 seconds:</u>**
    - Re-query the table after the TTL expires to observe the changes.

```
def update_ttl(movie_id, ttl):

    with open("ahmadgadalla02@gmail.com-token.json") as f:
        secrets = json.load(f)

    CLIENT_ID = secrets["clientId"]
    CLIENT_SECRET = secrets["secret"]

    cloud_config = {
        'secure_connect_bundle': 'secure-connect-ahmad.zip'
    }

    auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)

    cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
    session = cluster.connect()

    keyspace_name = "movies"
    session.set_keyspace(keyspace_name)

    # Step 1: Read the existing row with the desired movie_id
    query = "SELECT * FROM movie WHERE id = %s"
    result = session.execute(query, (movie_id,))
    for row in result:
        movie_details = row

    if not movie_details:
        print(f"No movie found with ID: {movie_id}")
```

```
            return
        try:
            # Step 2: Delete the old row
            delete_query = "DELETE FROM movie WHERE id = %s"
            session.execute(delete_query, (movie_id,))

            # Step 3: Insert the row back with the new TTL value
            insert_query = """
            INSERT INTO movie (id, name, movie_cast, movie_poster)
            VALUES (%s, %s, %s, %s) USING TTL %s
            """
            session.execute(insert_query, (movie_details.id,
movie_details.name, movie_details.movie_cast,
movie_details.movie_poster, ttl))
            print(f"TTL updated successfully for movie with ID:
{movie_id}")

        except Exception as e:
            print(f"An error occurred: {e}")


        session.shutdown()
        cluster.shutdown()
```

- Check output and schema.



```
select TTL (name) from movie where id = 1;
```

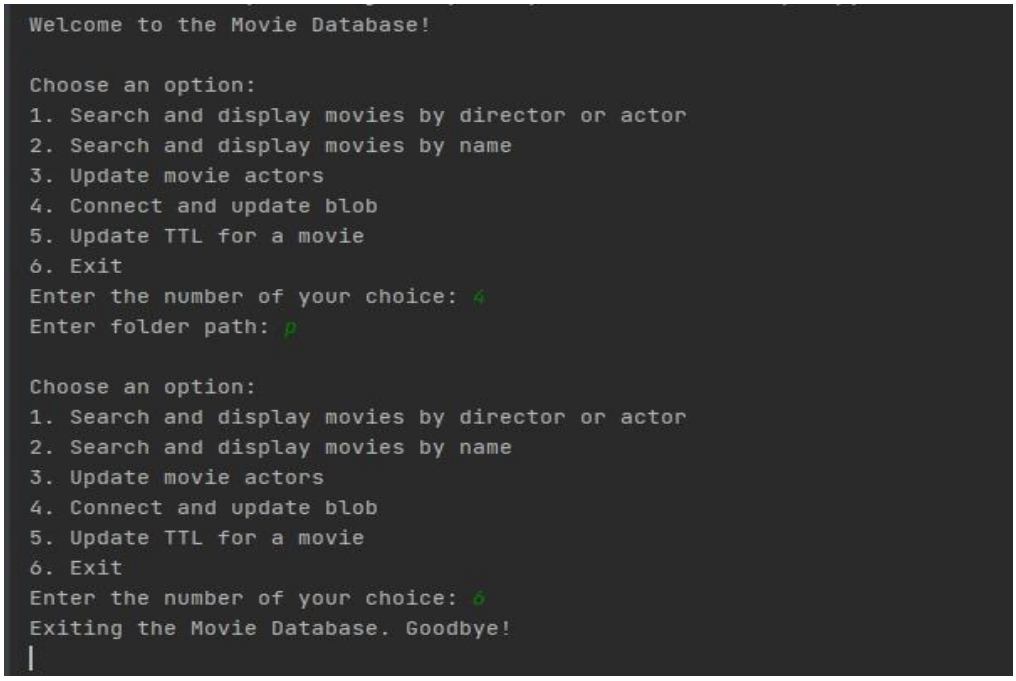
```
(1 rows)
token@cqlsh:movies> select TTL (name) from movie where id = 1;

 ttl(name)
-----------
        10

(1 rows)
token@cqlsh:movies> select TTL (name) from movie where id = 1;

 ttl(name)
-----------
         7

(1 rows)
token@cqlsh:movies> select TTL (name) from movie where id = 1;

 ttl(name)
-----------
         4

(1 rows)
token@cqlsh:movies> select TTL (name) from movie where id = 1;

 ttl(name)
-----------
         3

(1 rows)
token@cqlsh:movies> select TTL (name) from movie where id = 1;

 ttl(name)
-----------

(0 rows)
token@cqlsh:movies>
```

9. **Menu:**

```
def main_menu():
    print("Welcome to the Movie Database!")

    while True:
        print("\nChoose an option:")
        print("1. Search and display movies by director or actor")
        print("2. Search and display movies by name")
        print("3. Update movie actors")
        print("4. Connect and update blob")
        print("5. Update TTL for a movie")
        print("6. Exit")

        choice = input("Enter the number of your choice: ")

        if choice == "1":
            search_term = input("Enter director or actor name: ")
            output_folder = input("Enter output folder: ")
            query_and_display_movies_by_director_or_actor(search_term,
output_folder)

        elif choice == "2":
```

```
            search_term = input("Enter movie name: ")
            output_folder = input("Enter output folder: ")
            query_and_display_movies_name(search_term, output_folder)

        elif choice == "3":
            movie_id = input("Enter movie ID: ")
            new_actor = input("Enter new actor's name: ")
            update_movie_actors(int(movie_id), new_actor)

        elif choice == "4":
            folder_path = input("Enter folder path: ")
            connect_and_update_blob(folder_path)

        elif choice == "5":
            movie_id = input("Enter movie ID: ")
            ttl = input("Enter new TTL (time to live) in seconds: ")
            update_ttl(int(movie_id), int(ttl))

        elif choice == "6":
            print("Exiting the Movie Database. Goodbye!")
            break

        else:
            print("Invalid choice. Please enter a valid number.")


main_menu()
```

- Check output.



Cooked by:

1-	Menna Elminshawy	20217011
2-	Ahmad Wael Abdelaziz	20216016
3-	Ahmed Khaled Ahmed	20216004
4-	Jana Mohamed Nayef	20216129
5-	Radwa Belal Abdallah	20217005

Group ID: DS2