# Final Exam

# AI-Lab

# FA21-BCS-136

# ZAID ASGHAR VIRK

**QUESTION NO 1:**

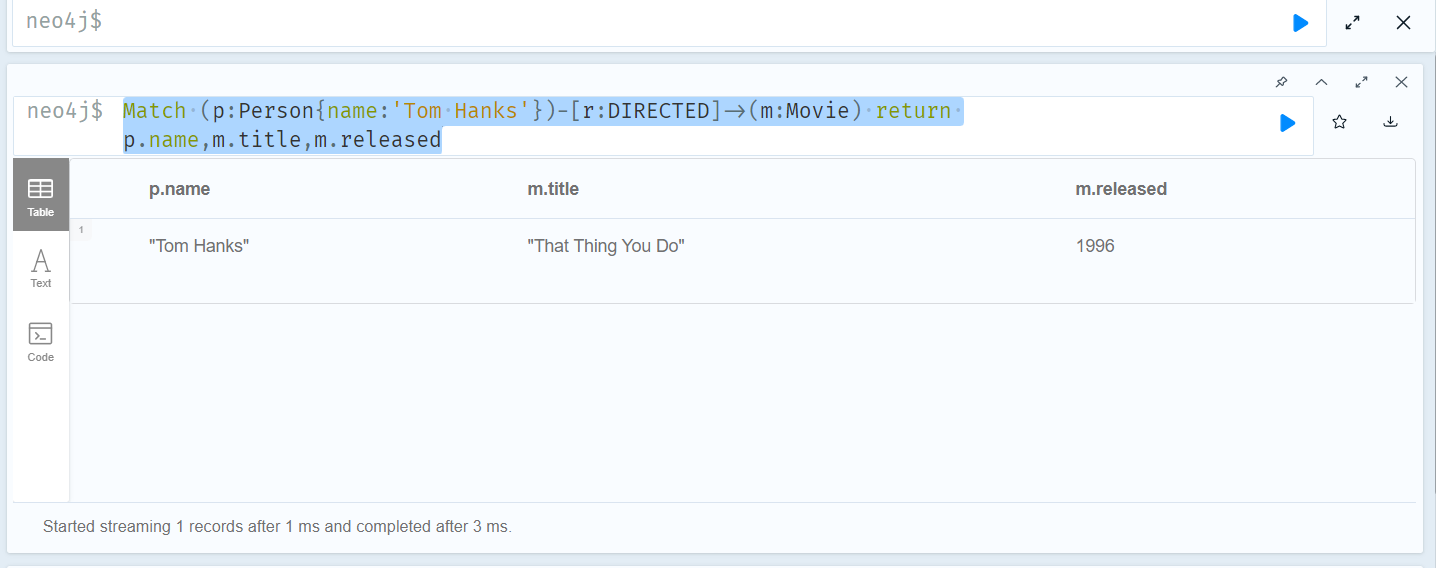
**P1.**

**QUERY:**

Match (p:Person{name:'Tom Hanks'})-[r:DIRECTED]->(m:Movie) return p.name,m.title

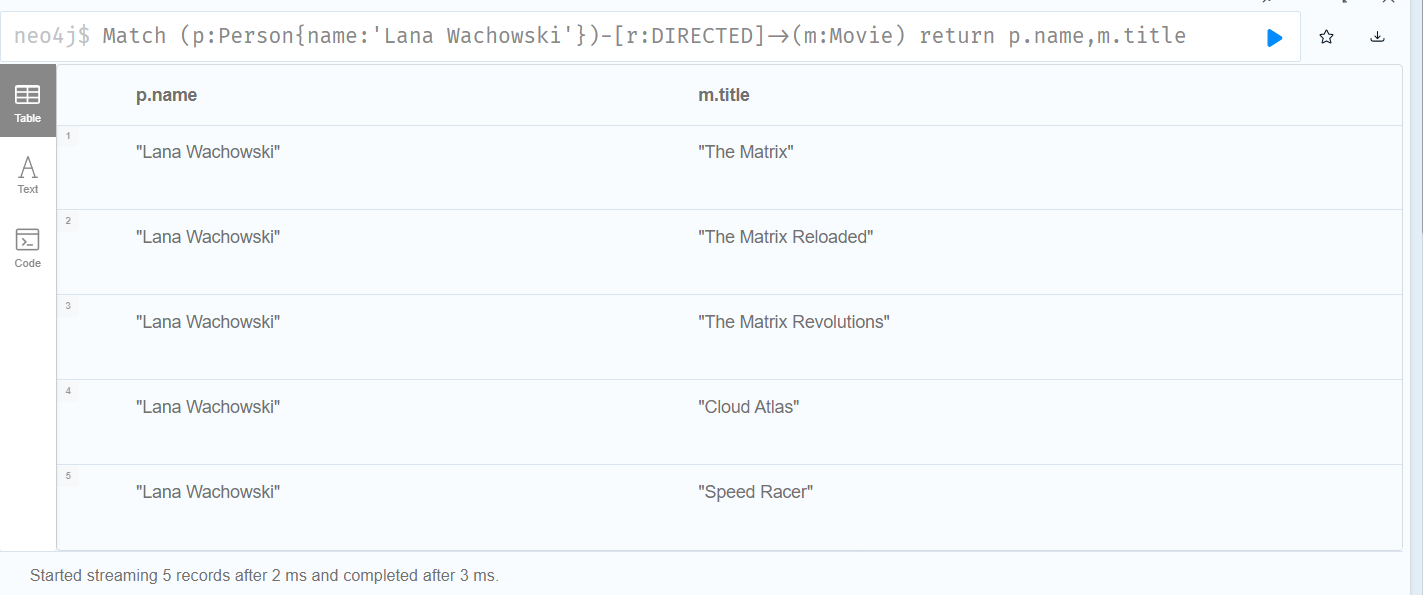
**For the released year too:**

Match (p:Person{name:'Tom Hanks'})-[r:DIRECTED]->(m:Movie) return p.name,m.title,m.released

****

**A screenshot of a computer

Description automatically generated**

****

**P2:**

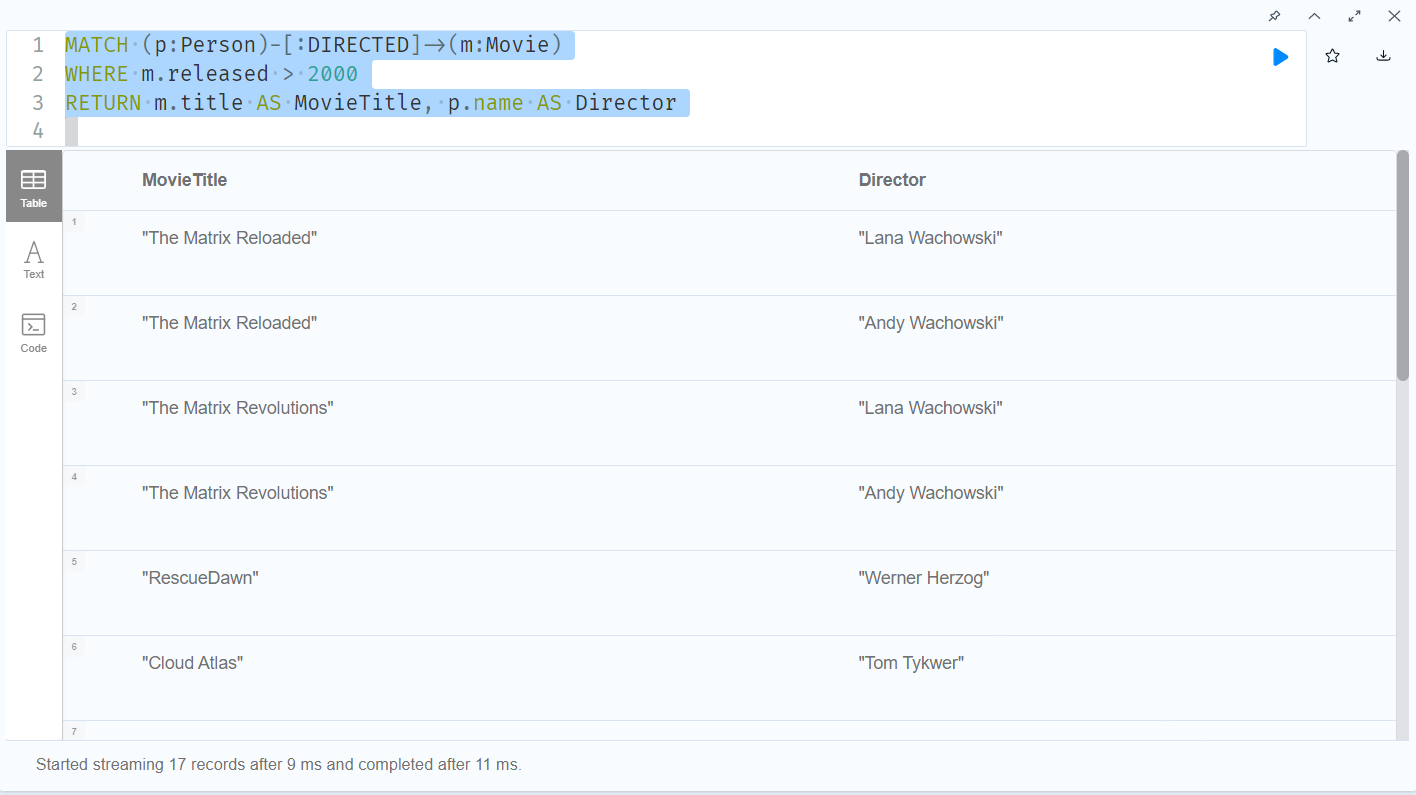
**Query:**

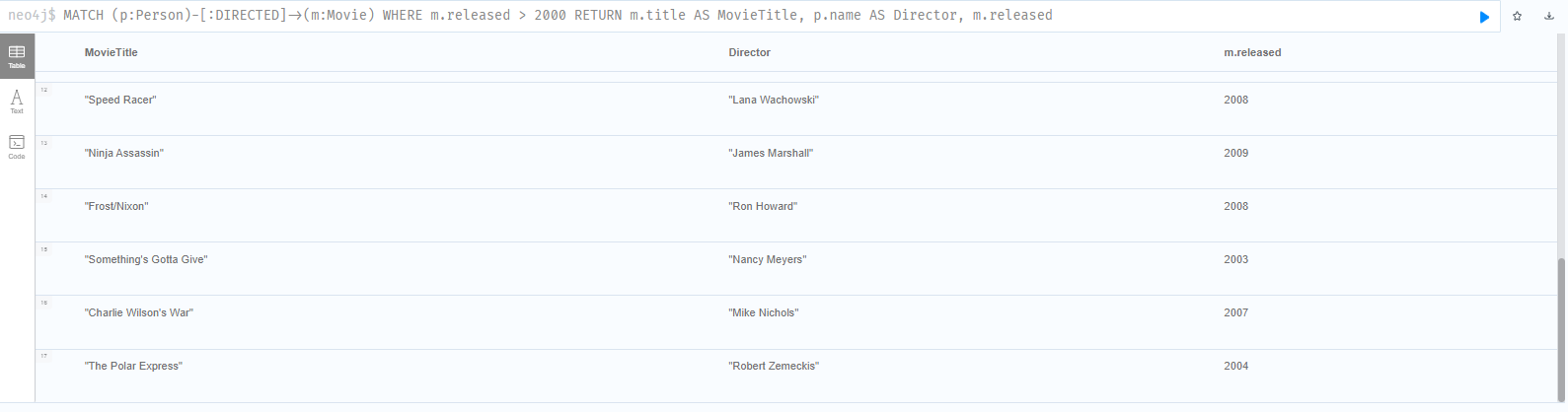
MATCH (p:Person)-[:DIRECTED]->(m:Movie)

WHERE m.released > 2000

RETURN m.title AS MovieTitle, p.name AS Director

**Explanation, Multiple Directors against each movie are being displayed.**

****

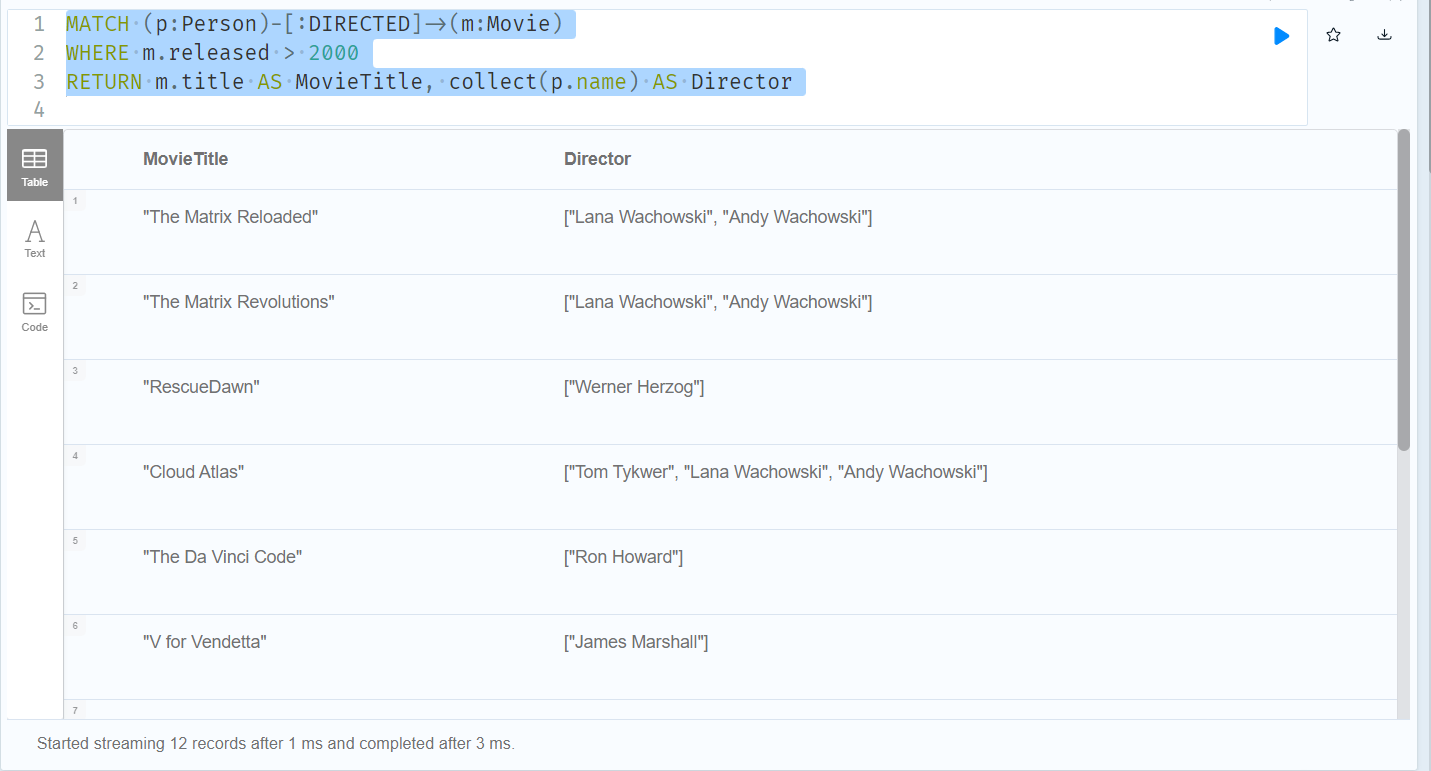
****

**To Print the name of directors as an array against each movie you can use:**

MATCH (p:Person)-[:DIRECTED]->(m:Movie)

WHERE m.released > 2000

RETURN m.title AS MovieTitle, collect(p.name) AS Director



**P3:**

Query:

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WITH p, COUNT(m.title) AS l

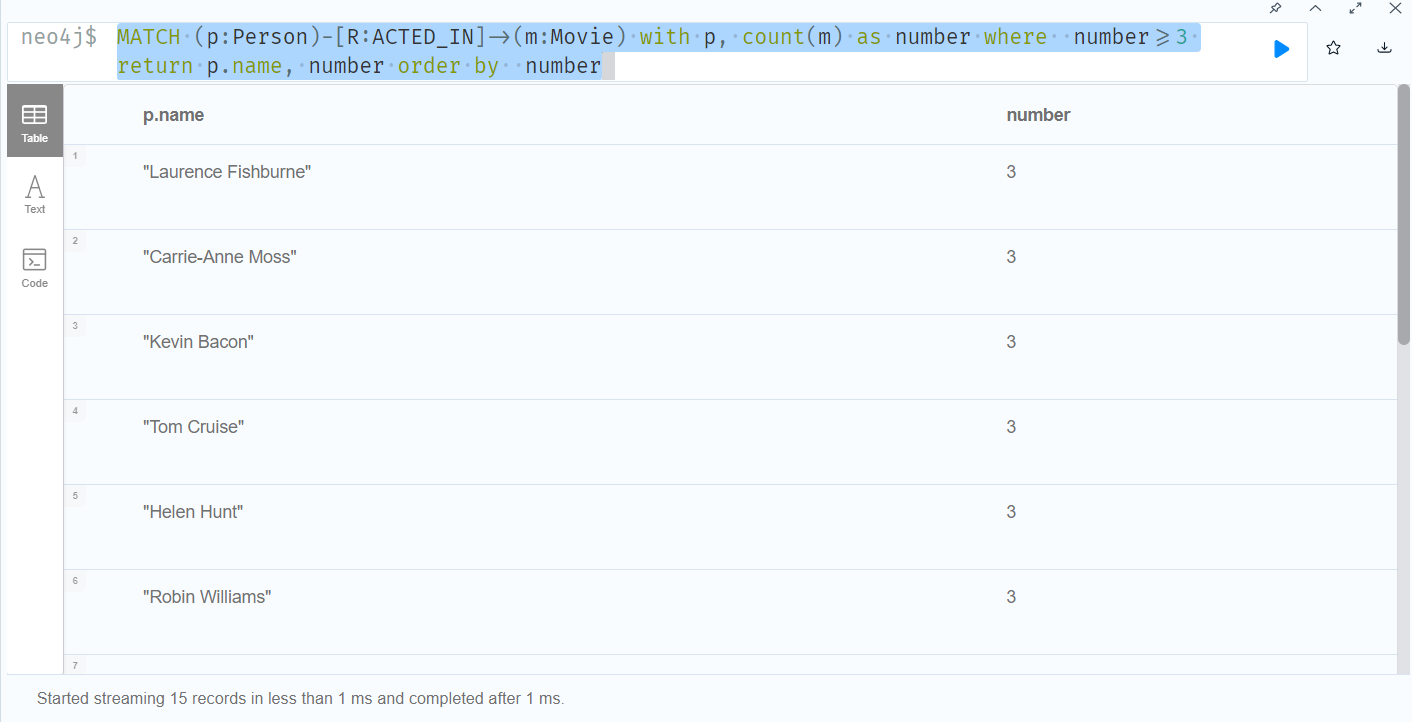
WHERE l >=3

RETURN p.name,l



Also for sorting:

MATCH (p:Person)-[R:ACTED\_IN]->(m:Movie) with p, count(m) as number where  number>=3 return p.name, number order by  number



**P4:**

Query:

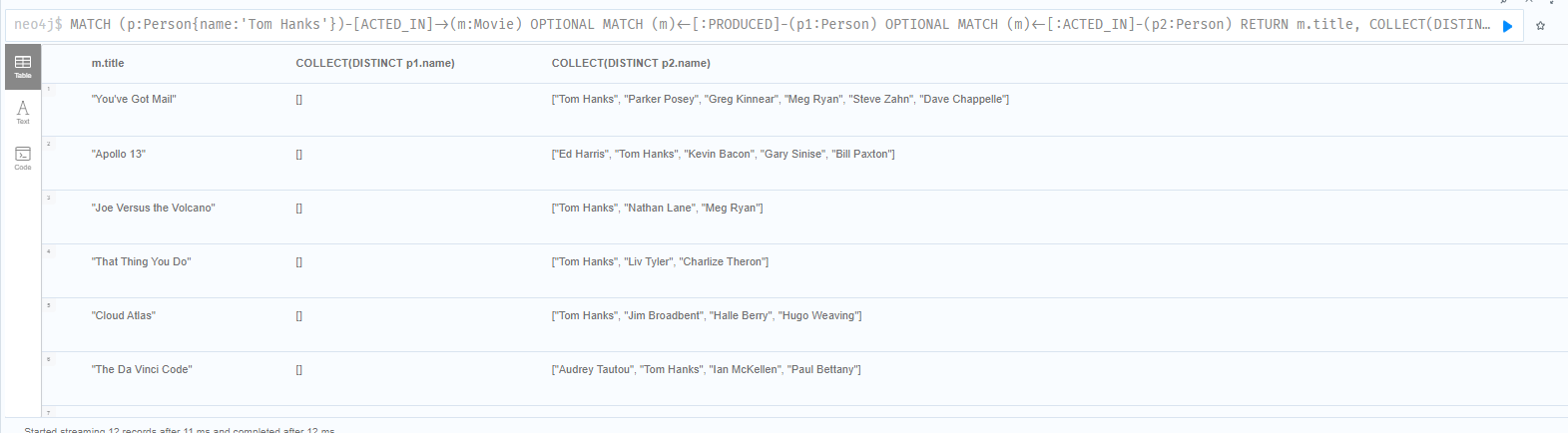
MATCH (p:Person{name:'Tom Hanks'})-[ACTED\_IN]->(m:Movie)

OPTIONAL MATCH (m)<-[:PRODUCED]-(p1:Person)

OPTIONAL MATCH (m)<-[:ACTED\_IN]-(p2:Person)

RETURN m.title, COLLECT(DISTINCT p1.name), COLLECT(DISTINCT p2.name)

No producers have been listed against movies tom hanks was actor in.



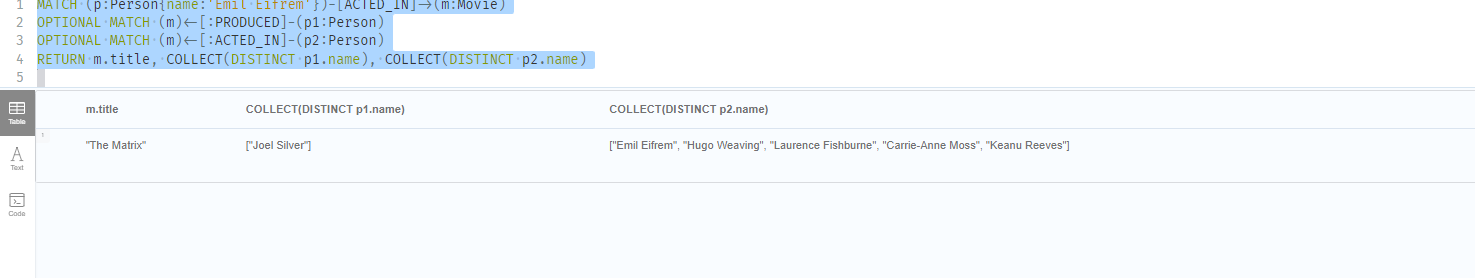
TO verify:

MATCH (p:Person{name:'Emil Eifrem'})-[ACTED\_IN]->(m:Movie)

OPTIONAL MATCH (m)<-[:PRODUCED]-(p1:Person)

OPTIONAL MATCH (m)<-[:ACTED\_IN]-(p2:Person)

RETURN m.title, COLLECT(DISTINCT p1.name), COLLECT(DISTINCT p2.name)



P4(Without Using Optional)

MATCh (n:Person{name:'Tom Hanks'})-[ACTED\_IN]->(m:Movie)

MATCH (a:Person)-[:ACTED\_IN]->(m)

MATCH (p:Person)-[:PRODUCED]->(m)

return m.title as movietitle, COLLECT(DISTINCT p.name) AS Producers,COLLECT(DISTINCT a.name) AS Actors

A computer screen shot of a computer

Description automatically generated

No producers have been listed against movies tom hanks was actor in.

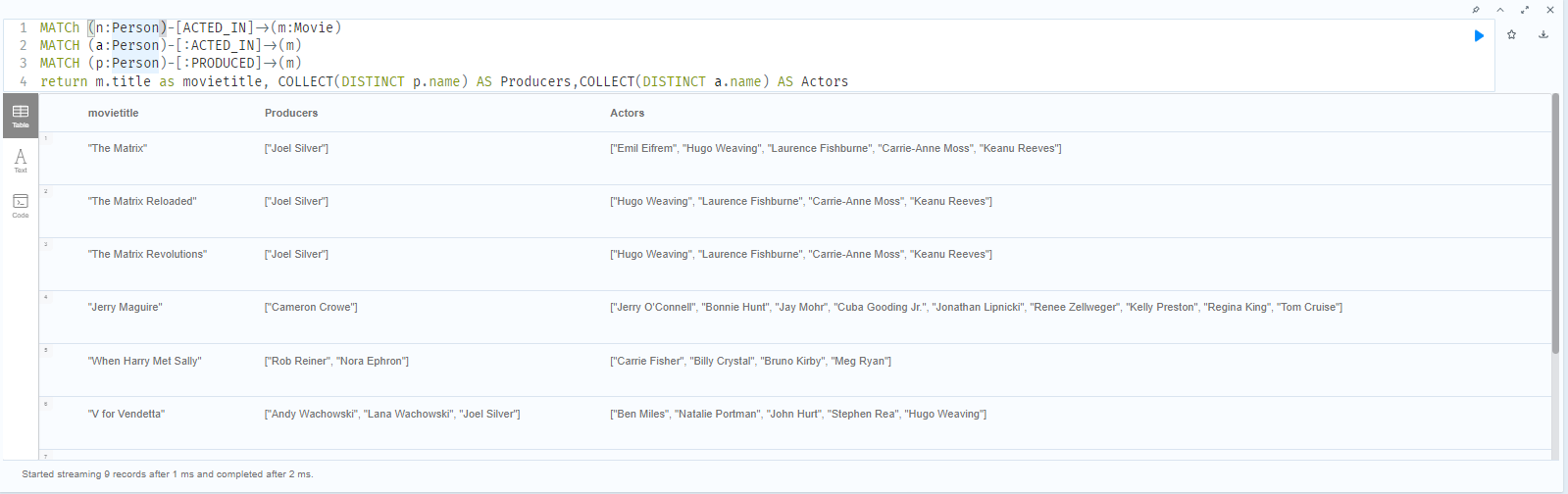
To verify:

MATCh (n:Person)-[ACTED\_IN]->(m:Movie)

MATCH (a:Person)-[:ACTED\_IN]->(m)

MATCH (p:Person)-[:PRODUCED]->(m)

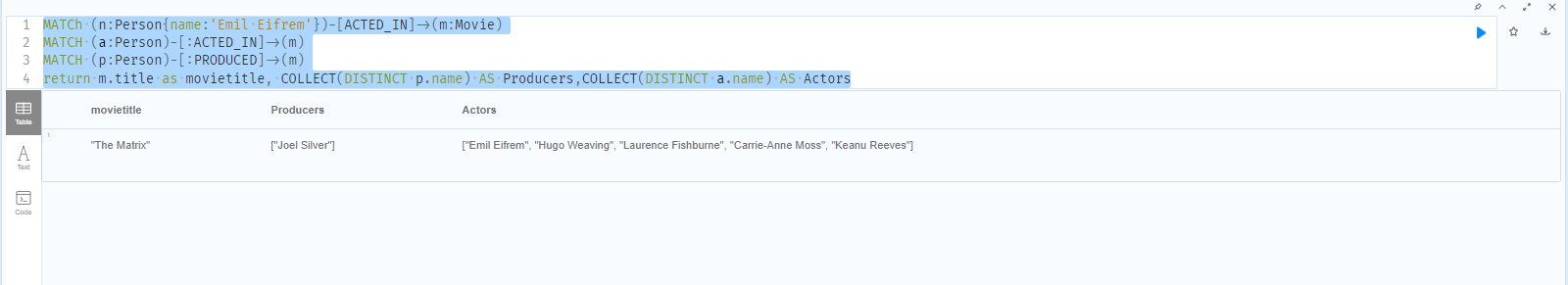
return m.title as movietitle, COLLECT(DISTINCT p.name) AS Producers,COLLECT(DISTINCT a.name) AS Actors

****

MATCh (n:Person{name:'Emil Eifrem'})-[ACTED\_IN]->(m:Movie)

MATCH (a:Person)-[:ACTED\_IN]->(m)

MATCH (p:Person)-[:PRODUCED]->(m)

return m.title as movietitle, COLLECT(DISTINCT p.name) AS Producers,COLLECT(DISTINCT a.name) AS Actors

P5:

MATCH (m:Movie)

DETACH DELETE m

# Question#2:

Code:

import random

#########################

Size\_of\_Population = 70

Mutation\_Rate = 0.01

GENERATIONS = 1000

TARGET = 50

Error\_Thresh\_Hold = 2

#########################

def initialize\_population1(size):

    population = []

    for \_ in range(size):

        x = random.randint(0, TARGET)

        y = random.randint(0, TARGET)

        population.append((x, y))

    return population

#A better alternate using listcomprehension,

def initialize\_population(size):

    return [(random.randint(0, TARGET), random.randint(0, TARGET)) for \_ in range(size)]

def fitness(individual):

    x, y = individual

    return abs(3\*x + 2\*y - TARGET)

def parent\_selection(population):

    tournament\_size = 5

    selected = random.sample(population, tournament\_size)

    selected.sort(key=fitness)

    return selected[0], selected[1]

#Crossover

def crossover(parent1, parent2):

    x1, y1 = parent1

    x2, y2 = parent2

    child1 = (x1, y2)

    child2 = (x2, y1)

    return child1, child2

def mutaion(individual):

    x, y = individual

    if random.random() < Mutation\_Rate:

        x = random.randint(0, 50)

    if random.random() < Mutation\_Rate:

        y = random.randint(0, 50)

    return (x, y)

def next\_generation(current\_population):

    new\_pop = []

    for \_ in range(Size\_of\_Population // 2):

        parent1, parent2 = parent\_selection(current\_population)

        child1, child2 = crossover(parent1, parent2)

        child1 = mutaion(child1)

        child2 = mutaion(child2)

        new\_pop.append(child1)

        new\_pop.append(child2)

    return new\_pop

#genteic algorithm

def geneti\_algo():

    population = initialize\_population(Size\_of\_Population)

    for generation in range(GENERATIONS):

        population.sort(key=fitness)

        best\_individual = population[0]

        best\_fitness = fitness(best\_individual)

        ###Question2,last part Conversion check###

        if best\_fitness < Error\_Thresh\_Hold:

            return best\_individual, generation

        population = next\_generation(population)

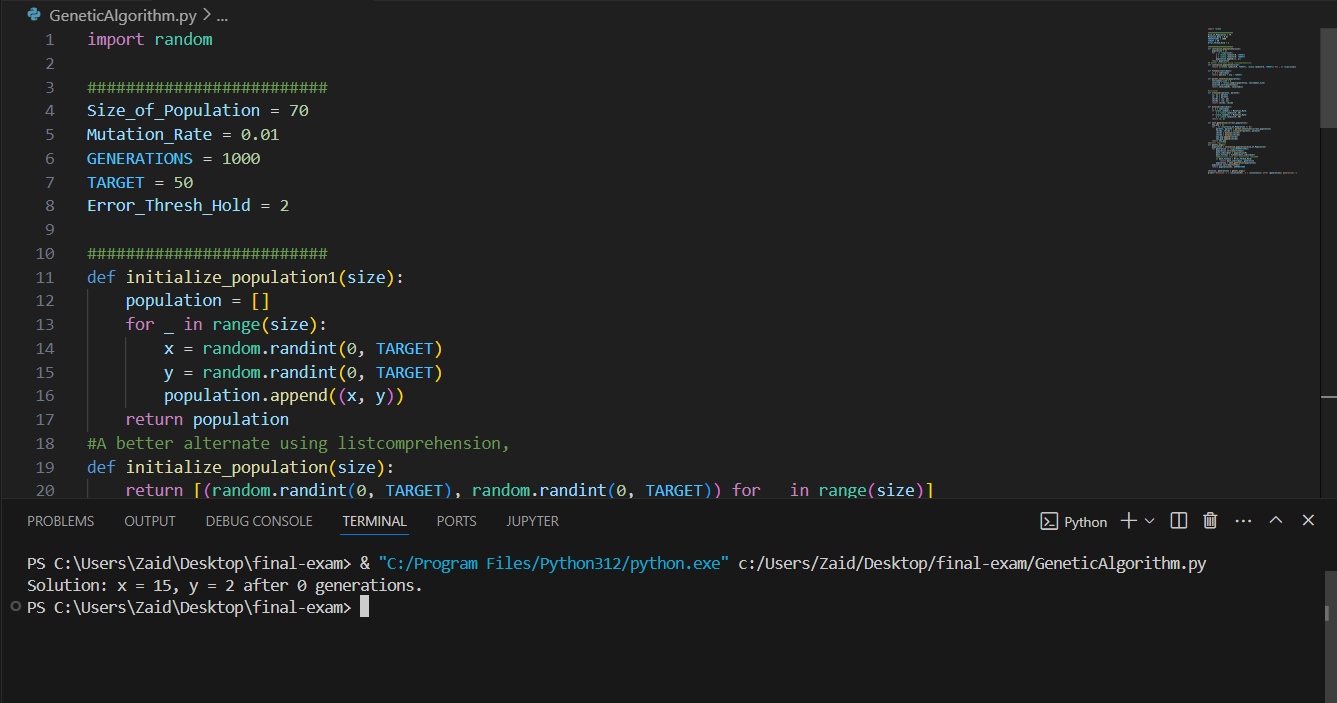
    population.sort(key=fitness)

    return population[0], GENERATIONS

solution, generations = geneti\_algo()

print(f"Solution: x = {solution[0]}, y = {solution[1]} after {generations} generations.")

OUTPUT:



A screen shot of a computer program

Description automatically generated

**Changing Parameters 1:**

Size\_of\_Population = 10

Mutation\_Rate = 0.01

GENERATIONS = 100

TARGET = 50

Error\_Thresh\_Hold = 2

OUTPUT:



A screen shot of a computer

Description automatically generated

**Question NO 3:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import numpy as np

df = pd.read\_csv('./Datasetfile/FuelConsumptionCo2.csv')

print(df.head())

X = df[['ENGINESIZE']]

y = df['CO2EMISSIONS']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

intercept = model.intercept\_

coefficients = model.coef\_

print(f'Intercept: {intercept}')

print(f'Coefficients: {coefficients}')

enginecc=2.5

engine\_size = pd.DataFrame([[enginecc]], columns=['ENGINESIZE'])

predicted\_CO2 = model.predict(engine\_size)

print(f'Predicted CO2 emissions for a vehicle with a {enginecc}-liter engine: {predicted\_CO2[0]}')

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Absolute Error (MAE): {mae}')

print(f'Mean Squared Error (MSE): {mse}')

print(f'R-squared (R2): {r2}')

Output:

