BIG DATA MANAGEMENT SYSTEMS: PROJECT #3 – GRAPH DATABASES/NEO4J

July 5, 2019

Zoe Kotti and Chryssa Nampouri
Department of Management Science and Technology
Athens University of Economics and Business
Athens, Greece
{t8150062, t8150096}@aueb.gr

Supervisor: Prof. Damianos Chatziantoniou

Contents

1 Neo4j Project Description

In this project a dataset was provided and the purpose was to represent it in the relational model, in the graph model, to run some queries and to measure performance. The steps that were followed are the following.

1. Go to Stanford Large Network Dataset Collection. [1]
2. Use the Pokec dataset in Social Networks category. [2]
3. For each user keep user_id, age and gender.
4. Load data in MySQL and Neo4j.
5. Write queries in SQL and Cypher for the following queries and benchmark both.
   - For each user, count his/her friends.
   - For each user, count his/her friends of friends.
   - For each user, count his/her friends that are over 30.
   - For each male user, count how many male and female friends he is having.

[1] https://snap.stanford.edu/data/#socnets
[2] https://snap.stanford.edu/data/soc-Pokec.html

2 Data Loading

For the data loading to both MySQL and Neo4j we first created the following configuration file.

```
# MySQL
HOST = "localhost"
PORT = 3306
USERNAME = "root"
PASSWORD = "password"
DATABASE = "soc_pokec"

# Neo4j
NEO4J_PASSWORD = "password"
```

2.1 MySQL

For the loading of the data to MySQL the following Python script was used.

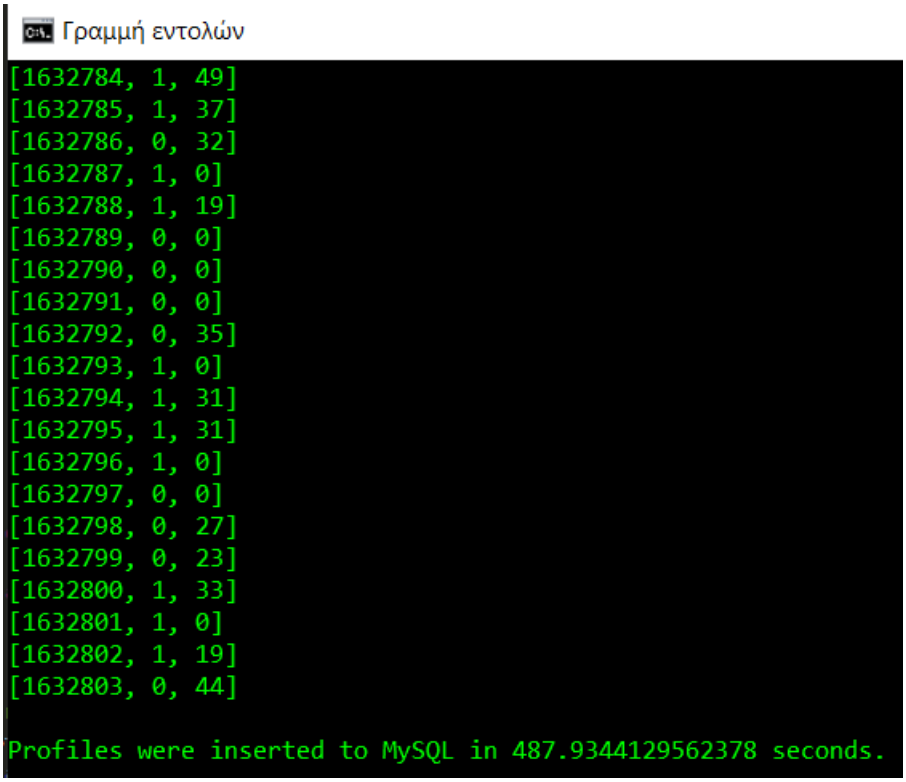
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import config
import csv
import pymysql
import time


def write_profiles_mysql(profiles_file):
    host = config.HOST
    port = config.PORT
    username = config.USERNAME
    password = config.PASSWORD
    database = config.DATABASE

    query = "INSERT INTO `profiles` (`user_id`, `gender`, `age`) " \
            "VALUES(%s, %s, %s)"

    try:
        connection = pymysql.connect(host=host, port=port, \
                                     user=username, passwd=password, \
                                     db=database)
        # Prepare Cursor object
```

```
        cursor = connection.cursor()

        # Execute query
        with open(profiles_file, "r", encoding="utf-8") as profiles:
            csv_reader = csv.reader(profiles, delimiter = "\t")

            for row in csv_reader:
                profile = [None, None, None]
                try:
                    profile[0] = int(row[0])
                except:
                    profile[0] = -1
                try:
                    profile[1] = int(row[3])
                except:
                    profile[1] = -1
                try:
                    profile[2] = int(row[7])
                except:
                    profile[2] = -1
                cursor.execute(query, profile)
                print(profile)

        connection.commit()

    except Exception as e:
        print(e)
    finally:
        connection.close()


def write_relationships_mysql(relationships_file):
    host = config.HOST
    port = config.PORT
    username = config.USERNAME
    password = config.PASSWORD
    database = config.DATABASE

    query = "INSERT INTO `relationships` (`user_id`, `friend_id`) " \
            "VALUES(%s, %s)"

    try:
```

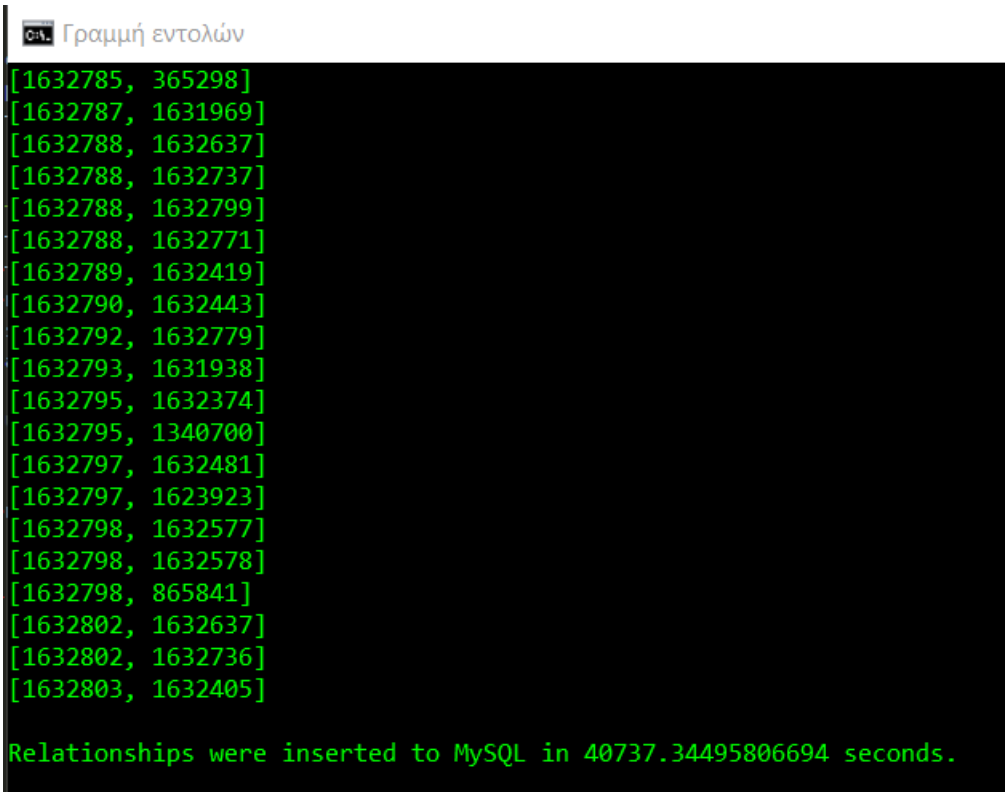
```
        connection = pymysql.connect(host=host, port=port, \
                                     user=username, passwd=password, \
                                     db=database)
        # Prepare Cursor object
        cursor = connection.cursor()

        # Execute query
        with open(relationships_file, "r", encoding="utf-8") as relationships:
            csv_reader = csv.reader(relationships, delimiter = "\t")

            for row in csv_reader:
                relationship = [None, None]
                try:
                    relationship[0] = int(row[0])
                except:
                    relationship[0] = -1
                try:
                    relationship[1] = int(row[1])
                except:
                    relationship[1] = -1
                cursor.execute(query, relationship)
                print(relationship)

        connection.commit()

    except Exception as e:
        print(e)
    finally:
        connection.close()


def main():
    profiles_file = "../dataset/soc-pokec-profiles.csv"
    relationships_file = "../dataset/soc-pokec-relationships.csv"

    start_time = time.time()
    write_profiles_mysql(profiles_file)
    elapsed_time = time.time() - start_time
    print("\nProfiles were inserted to MySQL \
      in {} seconds.\n".format(elapsed_time))

    start_time = time.time()
```

```
    write_relationships_mysql(relationships_file)
    elapsed_time = time.time() - start_time
    print("Relationships were inserted to MySQL \
      in {} seconds.\n".format(elapsed_time))


if __name__ == "__main__":

    main()
```

The time required for the loading of profiles to MySQL is depicted in the following figure.



The time required for the loading of relationships to MySQL is depicted in the following figure.



In the following figure it is demonstrated that all data were inserted correctly to MySQL.

```
MySQL 8.0 Command Line Client
Your MySQL connection id is 38
Server version: 8.0.16 MySQL Community Server - GPL

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use soc_pokec;
Database changed
mysql> select count(*) from profiles;
+----------+
| count(*) |
+----------+
|  1632803 |
+----------+
1 row in set (0.79 sec)

mysql> select count(*) from relationships;
+----------+
| count(*) |
+----------+
| 30622564 |
+----------+
1 row in set (11.81 sec)
```

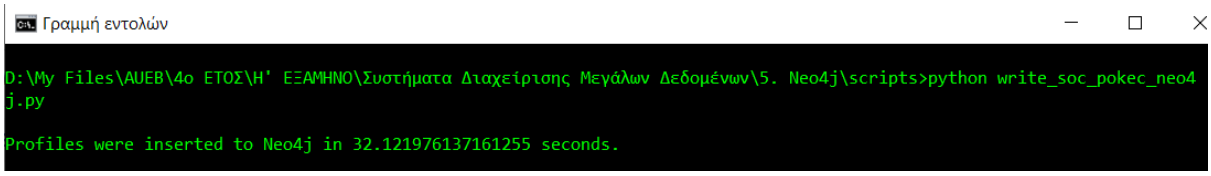
2.2 Neo4j

For the loading of the data to Neo4j the following Python script was initially used.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import config
import csv
from py2neo import Graph
import time


def write_profiles_neo4j():
    password = config.NEO4J_PASSWORD
    graph = Graph(password=password)

    query = """
    USING PERIODIC COMMIT
    LOAD CSV FROM "file:///soc-pokec-profiles.csv" \
    AS row FIELDTERMINATOR '\t'
    CREATE (:User {user_id: toInteger(row[0]), gender: toInteger(row[3]), \
    age: toInteger(row[7])});
    """

    graph.run(query)


def write_relationships_neo4j():
    password = config.NEO4J_PASSWORD
    graph = Graph(password=password)

    query = """
    USING PERIODIC COMMIT
    LOAD CSV FROM "file:///soc-pokec-relationships.csv" \
    AS row FIELDTERMINATOR '\t'
    MERGE (u1:User {user_id: toInteger(row[0])})
    MERGE (u2:User {user_id: toInteger(row[1])})
    MERGE (u1)-[:HAS_FRIEND]->(u2);
    """

    graph.run(query)

```

```python

def main():
    start_time = time.time()
    write_profiles_neo4j()
    elapsed_time = time.time() - start_time
    print("\nProfiles were inserted to Neo4j \
      in {} seconds.\n".format(elapsed_time))

    start_time = time.time()
    write_relationships_neo4j()
    elapsed_time = time.time() - start_time
    print("\nRelationships were inserted to Neo4j in \
      {} seconds.\n".format(elapsed_time))


if __name__ == "__main__":

    main()
```

While profiles were successfully loaded to Neo4j as depicted in the following figure, the function for the loading of relationships was running for 1.5 days and was eventually interrupted by us.



We searched the web and found the solution of using the Neo4j import tool *neo/bin/neo4j-import* to ingest profiles and the relationships between them as described in the Operations Manual of Neo4j. [3]

2.2.1 Data Transformation

The data are provided in tab-separated format (TSV). According to the instructions of the Neo4j import tool, we had to transform the files of profiles and relationships from tab-separated to comma-separated (CSV) and only keep the attributes of profiles that we were interested in (i.e. user_id, age, gender). In addition, the headers of both CSV files had to follow a specific format. Particularly, for the file of profiles the headers of user_id, age, gender are the following:

[3] https://neo4j.com/docs/operations-manual/current/tools/import/syntax/

- userId:ID(User)
- gender:INT
- age:INT

For the file of relationships the headers of user_id and friend_id are the following:

- :START_ID(User)
- :END_ID(User)

For this process of data transformation, we used the following Python script.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import csv
import time


def convert_profiles(profiles_file, profiles_file_new):
    with open(profiles_file_new, "w", newline="") as file:
        writer = csv.writer(file, delimiter=",")

        writer.writerow(["userId:ID(User)", "gender:INT", "age:INT"])

        with open(profiles_file, "r", encoding="utf-8") as profiles:
            csv_reader = csv.reader(profiles, delimiter = "\t")

            for row in csv_reader:
                profile = [None, None, None]
                try:
                    profile[0] = int(row[0])
                except:
                    profile[0] = -1
                try:
                    profile[1] = int(row[3])
                except:
                    profile[1] = -1
                try:
                    profile[2] = int(row[7])
                except:
```

```
                    profile[2] = -1

                writer.writerow(profile)


def convert_relationships(relationships_file, relationships_file_new):
    with open(relationships_file_new, "w", newline="") as file:
        writer = csv.writer(file, delimiter=",")

        writer.writerow([":START_ID(User)", ":END_ID(User)"])

        with open(relationships_file, "r", encoding="utf-8") as relationships:
            csv_reader = csv.reader(relationships, delimiter = "\t")

            for row in csv_reader:
                relationship = [None, None]
                try:
                    relationship[0] = int(row[0])
                except:
                    relationship[0] = -1
                try:
                    relationship[1] = int(row[1])
                except:
                    relationship[1] = -1

                writer.writerow(relationship)


def main():
    profiles_file = "../dataset/soc-pokec-profiles.csv"
    profiles_file_new = "../dataset/soc-pokec-profiles-new.csv"
    relationships_file = "../dataset/soc-pokec-relationships.csv"
    relationships_file_new = "../dataset/soc-pokec-relationships-new.csv"

    start_time = time.time()
    convert_profiles(profiles_file, profiles_file_new)
    elapsed_time_profiles = time.time() - start_time

    start_time = time.time()
    convert_relationships(relationships_file, relationships_file_new)
    elapsed_time_relationships = time.time() - start_time

```

```
    print("Conversion of profiles from tsv to csv \
      took {} seconds.".format(elapsed_time_profiles))
    print("Conversion of relationships from tsv to csv \
      took {} seconds.".format(elapsed_time_relationships))


if __name__ == "__main__":

    main()
```

In the following figure the time required for the conversion of the files is depicted.

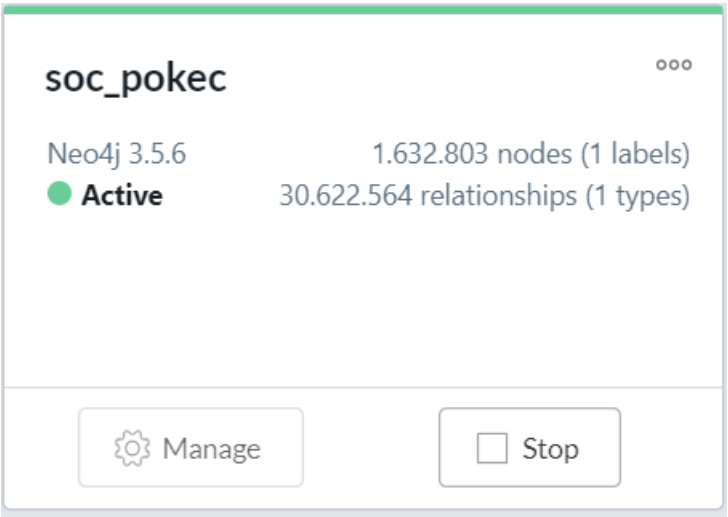
```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python convert_tsv_to_csv.py
Conversion of profiles from tsv to csv took 37.66357469558716 seconds.
Conversion of relationships from tsv to csv took 84.09617686271667 seconds.
```

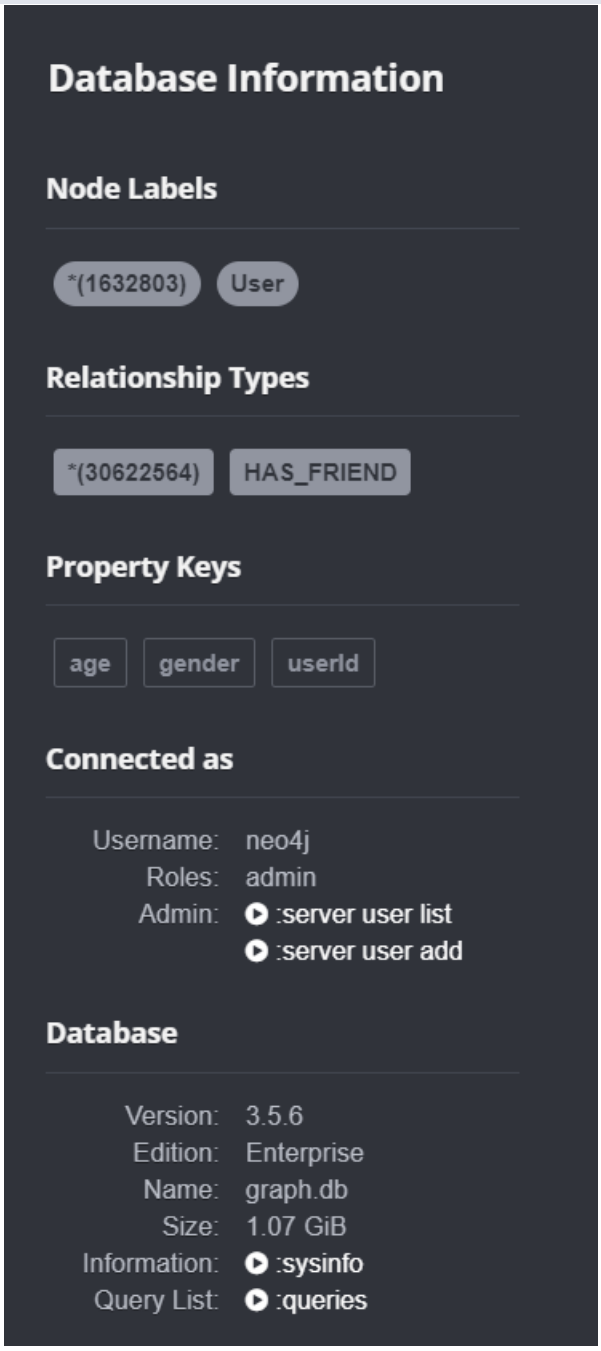
Next, according to the instructions again, the following command was used in terminal in order to load the data to Neo4j.

```
neo4j_home$ bin/neo4j-admin import
--id-type INTEGER
--nodes:User ../import/soc-pokec-profiles-new.csv
--relationships:HAS_FRIEND ../import/soc-pokec-relationships-new.csv
```

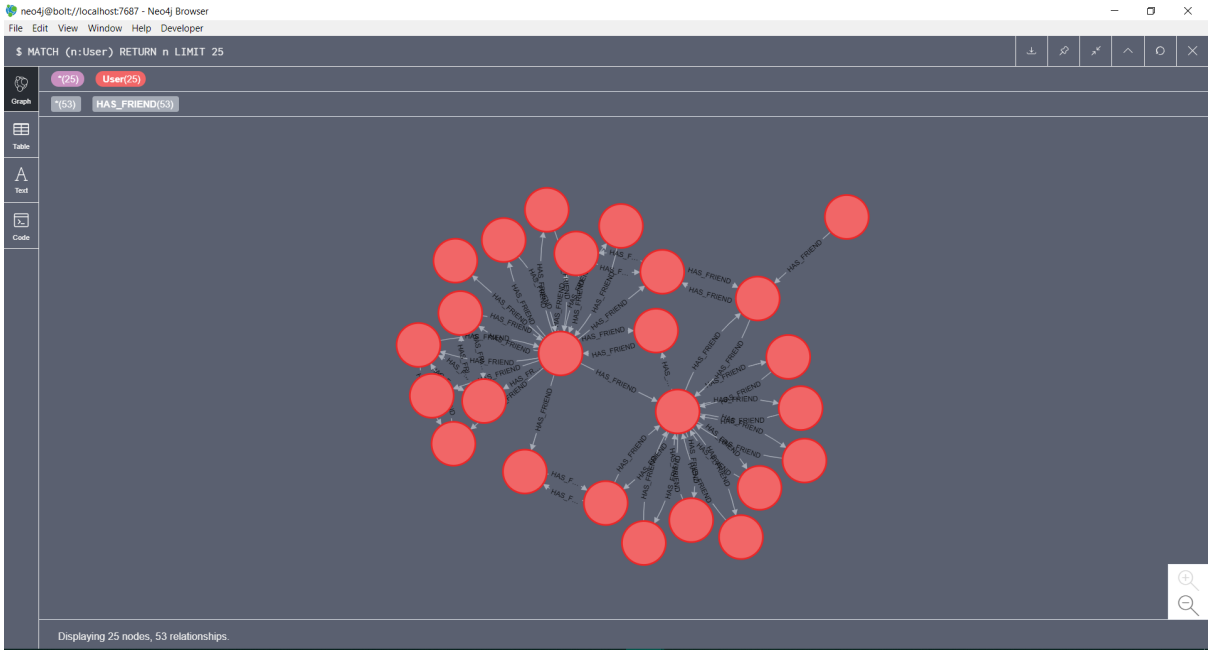
Through this process both profiles and their relationships were inserted to Neo4j successfully as depicted in the following figures.

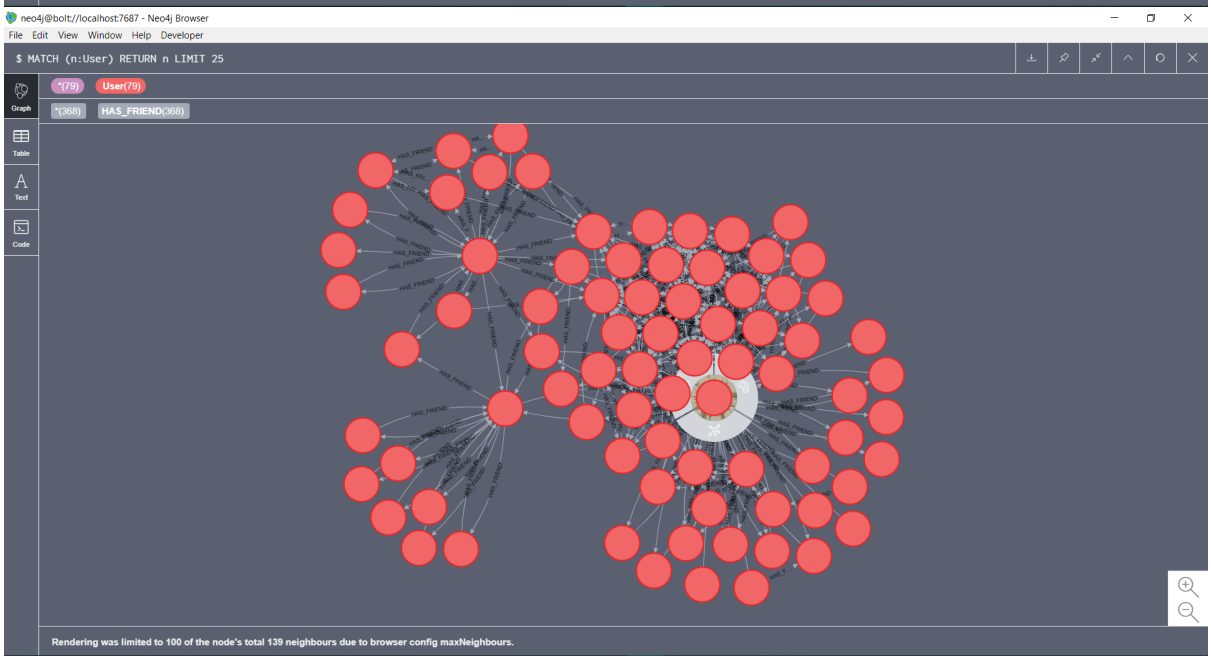
```
Γραμμή εντολών

C:\Users\Zwoul\.Neo4jDesktop\neo4jDatabases\database-bce2d087-52f3-4df1-9201-93a79384937d\installation-3.5.6\bin>neo4j-admin import --id-type INTEGER --nodes:User ../import/soc-pokec-profiles-new.csv --relations
hips:HAS_FRIEND ../import/soc-pokec-relationships-new.csv
Neo4j version: 3.5.6
Importing the contents of these files into C:\Users\Zwoul\.Neo4jDesktop\neo4jDatabases\database-bce2d087-52f3-4df1-9201-93a79384937d\installation-3.5.6\data\databases\graph.db:
Nodes:
  :User
  C:\Users\Zwoul\.Neo4jDesktop\neo4jDatabases\database-bce2d087-52f3-4df1-9201-93a79384937d\installation-3.5.6\bin\..\import\soc-pokec-profiles-new.csv
Relationships:
  :HAS_FRIEND
  C:\Users\Zwoul\.Neo4jDesktop\neo4jDatabases\database-bce2d087-52f3-4df1-9201-93a79384937d\installation-3.5.6\bin\..\import\soc-pokec-relationships-new.csv

Available resources:
  Total machine memory: 7.90 GB
  Free machine memory: 4.27 GB
  Max heap memory : 1.76 GB
  Processors: 4
  Configured max memory: 5.53 GB
  High-IO: false

Import starting 2019-06-17 02:44:46.774+0300
  Estimated number of nodes: 1.72 M
  Estimated number of node properties: 5.16 M
  Estimated number of relationships: 38.60 M
  Estimated number of relationship properties: 0.00
  Estimated disk space usage: 1.31 GB
  Estimated required memory usage: 1.02 GB

InteractiveReporterInteractions command list (end with ENTER):
  c: Print more detailed information about current stage
  i: Print more detailed information

(1/4) Node import 2019-06-17 02:44:46.988+0300
  Estimated number of nodes: 1.72 M
  Estimated disk space usage: 93.47 MB
  Estimated required memory usage: 1.02 GB
..........  ..........  ..........  ..........  ..........   5% ?1s 936ms
..........  ..........  ..........  ..........  ..........  10% ?5ms
..........  ..........  ..........  ..........  ..........  15% ?3ms
..........  ..........  ..........  ..........  ..........  20% ?4ms
..........  ..........  ..........  ..........  ..........  25% ?0ms
..........  ..........  ..........  ..........  ..........  30% ?747ms
..........  ..........  ..........  ..........  ..........  35% ?30ms
..........  ..........  ..........  ..........  ..........  40% ?205ms
..........  ..........  ..........  ..........  ..........  45% ?0ms
..........  ..........  ..........  ..........  ..........  50% ?0ms
..........  ..........  ..........  ..........  ..........  55% ?0ms
..........  ..........  ..........  ..........  ..........  60% ?172ms
..........  ..........  ..........  ..........  ..........  65% ?537ms
..........  ..........  ..........  ..........  ..........  70% ?733ms
..........  ..........  ..........  ..........  ..........  75% ?0ms
..........  ..........  ..........  ..........  ..........  80% ?596ms
..........  ..........  ..........  ..........  ..........  85% ?873ms
..........  ..........  ..........  ..........  ..........  90% ?727ms
..........  ..........  ..........  ..........  ..........  95% ?789ms
..........  ..........  ..........  ..........  .......... 100% ?590ms

(2/4) Relationship import 2019-06-17 02:44:55.279+0300
  Estimated number of relationships: 38.60 M
  Estimated disk space usage: 1.22 GB
  Estimated required memory usage: 1.02 GB
..........  ..........  ..........  ..........  ..........   5% ?848ms
..........  ..........  ..........  ..........  ..........  10% ?823ms
..........  ..........  ..........  ..........  ..........  15% ?1s 192ms
..........  ..........  ..........  ..........  ..........  20% ?802ms
..........  ..........  ..........  ..........  ..........  25% ?1s 148ms
..........  ..........  ..........  ..........  ..........  30% ?1s 36ms
..........  ..........  ..........  ..........  ..........  35% ?1s 735ms
..........  ..........  ..........  ..........  ..........  40% ?1s 943ms
..........  ..........  ..........  ..........  ..........  45% ?2s 117ms
..........  ..........  ..........  ..........  ..........  50% ?1s 640ms
..........  ..........  ..........  ..........  ..........  55% ?1s 714ms
..........  ..........  ..........  ..........  ..........  60% ?1s 926ms
..........  ..........  ..........  ..........  ..........  65% ?1s 801ms
..........  ..........  ..........  ..........  ..........  70% ?2s 102ms
..........  ..........  ..........  ..........  ..........  75% ?1s 956ms
..........  ..........  ..........  ..........  ..........  80% ?984ms
..........  ..........  ..........  ..........  ..........  85% ?0ms
..........  ..........  ..........  ..........  ..........  90% ?22ms
..........  ..........  ..........  ..........  ..........  95% ?0ms
..........  ..........  ..........  ..........  .......... 100% ?0ms

(3/4) Relationship linking 2019-06-17 02:45:19.182+0300
  Estimated required memory usage: 1.01 GB
..........  ..........  ..........  ..........  ..........   5% ?852ms
..........  ..........  ..........  ..........  ..........  10% ?868ms
..........  ..........  ..........  ..........  ..........  15% ?631ms
..........  ..........  ..........  ..........  ..........  20% ?714ms
..........  ..........  ..........  ..........  ..........  25% ?1s 152ms
..........  ..........  ..........  ..........  ..........  30% ?1s 235ms
..........  ..........  ..........  ..........  ..........  35% ?846ms
..........  ..........  ..........  ..........  ..........  40% ?855ms
..........  ..........  ..........  ..........  ..........  45% ?861ms
..........  ..........  ..........  ..........  ..........  50% ?946ms
..........  ..........  ..........  ..........  ..........  55% ?657ms
..........  ..........  ..........  ..........  ..........  60% ?644ms
..........  ..........  ..........  ..........  ..........  65% ?1s 42ms
..........  ..........  ..........  ..........  ..........  70% ?1s 207ms
..........  ..........  ..........  ..........  ..........  75% ?1s 155ms
..........  ..........  ..........  ..........  ..........  80% ?504ms
..........  ..........  ..........  ..........  ..........  85% ?905ms
..........  ..........  ..........  ..........  ..........  90% ?1s 313ms
..........  ..........  ..........  ..........  ..........  95% ?652ms
..........  ..........  ..........  ..........  .......... 100% ?6s 731ms

(4/4) Post processing 2019-06-17 02:45:43.629+0300
  Estimated required memory usage: 1020.01 MB
..........  ..........  ..........  ..........  ..........   5% ?264ms
..........  ..........  ..........  ..........  ..........  10% ?200ms
..........  ..........  ..........  ..........  ..........  15% ?203ms
..........  ..........  ..........  ..........  ..........  20% ?201ms
..........  ..........  ..........  ..........  ..........  25% ?0ms
..........  ..........  ..........  ..........  ..........  30% ?219ms
..........  ..........  ..........  ..........  ..........  35% ?204ms
..........  ..........  ..........  ..........  ..........  40% ?201ms
..........  ..........  ..........  ..........  ..........  45% ?5ms
..........  ..........  ..........  ..........  ..........  50% ?202ms
..........  ..........  ..........  ..........  ..........  55% ?201ms
..........  ..........  ..........  ..........  ..........  60% ?218ms
..........  ..........  ..........  ..........  ..........  65% ?0ms
..........  ..........  ..........  ..........  ..........  70% ?201ms
..........  ..........  ..........  ..........  ..........  75% ?202ms
..........  ..........  ..........  ..........  ..........  80% ?4ms
..........  ..........  ..........  ..........  ..........  85% ?282ms
..........  ..........  ..........  ..........  ..........  90% ?228ms
..........  ..........  ..........  ..........  ..........  95% ?0ms
..........  ..........  ..........  ..........  ..........
IMPORT DONE in 1m 2s 212ms.
Imported:
  1632803 nodes
  30622564 relationships
  4898409 properties
Peak memory usage: 1.05 GB
```


soc_pokec
Neo4j 3.5.6
Active
1.632.803 nodes (1 labels)
30.622.564 relationships (1 types)
Manage
Stop


Database Information
Node Labels
*(1632803)
User
Relationship Types
*(30622564)
HAS_FRIEND
Property Keys
age
gender
userId
Connected as
Username: neo4j
Roles: admin
Admin: :server user list
:server user add
Database
Version: 3.5.6
Edition: Enterprise
Name: graph.db
Size: 1.07 GiB
Information: :sysinfo
Query List: :queries

Two examples of the Neo4j graph are depicted below.





3 Queries

3.1 MySQL

The queries that we formed in SQL in order to answer the questions of the project are the following.

```
# Query 1: For each user, count his/her friends

SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
FROM profiles pro
LEFT JOIN relationships rel
ON pro.user_id = rel.user_id
GROUP BY pro.user_id;

# Query 2: For each user, count his/her friends of friends

SELECT pro.user_id as USER, COUNT(rel2.friend_id) as FRIENDS_OF_FRIENDS
FROM profiles pro
LEFT JOIN relationships rel1
ON pro.user_id = rel1.user_id
LEFT JOIN relationships rel2
ON rel1.friend_id = rel2.user_id
WHERE pro.user_id != rel2.friend_id
GROUP BY pro.user_id;

# Query 3: For each user, count his/her friends that are over 30

SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS_OVER_THIRTY
FROM profiles pro
LEFT JOIN relationships rel
ON pro.user_id = rel.user_id
AND (
    SELECT pro.age
    FROM profiles pro
    WHERE pro.user_id = rel.friend_id AND pro.age > 30)
GROUP BY pro.user_id;

# Query 4: For each male user, count how many male and female friends
# he is having

SELECT genders.user_id as USER_MALE, \
COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, \
```

```
COUNT(CASE WHEN genders.gender = 0 THEN 1 END) as FRIENDS_FEMALE
FROM (
    SELECT temp.user_id, pro.gender
    FROM profiles pro, (
        SELECT pro.user_id, pro.age, rel.friend_id
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
    WHERE pro.user_id = temp.friend_id) as genders
GROUP BY genders.user_id;
```

3.2 Neo4j

The queries that we formed in Cypher in order to answer the questions of the project are the following.

```
# Query 1: For each user, count his/her friends

MATCH (User)
OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
RETURN User.userId AS user, COUNT(u) AS friends
ORDER BY User.userId;

# Query 2: For each user, count his/her friends of friends

MATCH (User)
OPTIONAL MATCH (User)-[:HAS_FRIEND*2]->(f)
WHERE f <> User
RETURN User.userId AS user, COUNT(f) AS friends_of_friends
ORDER BY User.userId;

# Query 3: For each user, count his/her friends that are over 30

MATCH (User)
OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
WHERE u.age > 30
RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
ORDER BY User.userId;

# Query 4: For each male user, count how many male and female friends
# he is having

```

```
MATCH (User)-[:HAS_FRIEND]->(u)
WHERE User.gender = 1
RETURN User.userId AS user_male,
SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
ORDER BY User.userId;
```

4 Execute Queries

The results of all queries were written to CSV files.

4.1 MySQL

In order to execute the queries in MySQL we used the following Python script.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import config
import csv
import pymysql
import time


def print_info(query, time, file):
    print()
    print(query)
    print("MySQL: Executed in {} seconds".format(time))
    print("Wrote results to {}".format(file))


def write_results(rows, file, headings):
    with open(file, mode='w', newline='') as output_file:
        writer = csv.writer(output_file, delimiter=',', quotechar='"', \
            quoting=csv.QUOTE_MINIMAL)
        writer.writerow(headings)

        for row in rows:
            writer.writerow(row)


def execute_query(query):
    host = config.HOST
    port = config.PORT
    username =  config.USERNAME
    password = config.PASSWORD
    database = config.DATABASE

    try:
        connection = pymysql.connect(host=host, port=port, \
```

```
                                    user=username, passwd=password, \
                                    db=database)
        # Prepare Cursor object
        cursor = connection.cursor()

        cursor.execute(query)
        rows = cursor.fetchall()

        connection.commit()

    except Exception as e:
        print(e)
    finally:
        connection.close()

    return rows


# Query 1: For each user, count his/her friends
def count_friends():
    file = "results/q1/count_friends_mysql.csv"
    headings = ["user", "friends"]

    query = """
        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        GROUP BY pro.user_id;
    """

    start_time = time.time()
    results = execute_query(query)
    elapsed_time = time.time() - start_time

    write_results(results, file, headings)
    print_info(query, elapsed_time, file)


# Query 2: For each user, count his/her friends of friends
def count_friends_of_friends():
    file = "results/q2/count_friends_of_friends_mysql.csv"
```

```
    headings = ["user", "friends_of_friends"]

    query = """
        SELECT pro.user_id as USER, COUNT(rel2.friend_id)
        as FRIENDS_OF_FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel1
        ON pro.user_id = rel1.user_id
        LEFT JOIN relationships rel2
        ON rel1.friend_id = rel2.user_id
        WHERE pro.user_id != rel2.friend_id
        GROUP BY pro.user_id;
    """

    start_time = time.time()
    results = execute_query(query)
    elapsed_time = time.time() - start_time

    write_results(results, file, headings)
    print_info(query, elapsed_time, file)


# Query 3: For each user, count his/her friends that are over 30
def count_friends_over_thirty():
    file = "results/q3/count_friends_over_thirty_mysql.csv"
    headings = ["user", "friends_over_thirty"]

    query = """
        SELECT pro.user_id as USER, COUNT(rel.friend_id)
        as FRIENDS_OVER_THIRTY
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        AND (
            SELECT pro.age
            FROM profiles pro
            WHERE pro.user_id = rel.friend_id AND pro.age > 30)
        GROUP BY pro.user_id;
    """

    start_time = time.time()
    results = execute_query(query)
```

```
    elapsed_time = time.time() - start_time

    write_results(results, file, headings)
    print_info(query, elapsed_time, file)


# Query 4: For each male user, count how many male and female friends
# he is having
def count_male_female_friends():
    file = "results/q4/count_male_female_friends_mysql.csv"
    headings = ["user_male", "friends_male", "friends_female"]

    query = """
        SELECT genders.user_id as USER_MALE, \
        COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, \
        COUNT(CASE WHEN genders.gender = 0 THEN 1 END) as FRIENDS_FEMALE
        FROM (
            SELECT temp.user_id, pro.gender
            FROM profiles pro, (
                SELECT pro.user_id, pro.age, rel.friend_id
                FROM profiles pro
                LEFT JOIN relationships rel
                ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
            WHERE pro.user_id = temp.friend_id) as genders
        GROUP BY genders.user_id;
    """

    start_time = time.time()
    results = execute_query(query)
    elapsed_time = time.time() - start_time

    write_results(results, file, headings)
    print_info(query, elapsed_time, file)


def main():
    count_friends()
    count_friends_of_friends()
    count_friends_over_thirty()
    count_male_female_friends()


```

```
if __name__ == "__main__":

    main()
```

4.2 Neo4j

In order to execute the queries in Neo4j we used the following Python script.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import config
import csv
from py2neo import Graph
import time


def print_info(query, time, file):
    print()
    print(query)
    print("Neo4j: Executed in {} seconds".format(time))
    print("Wrote results to {}".format(file))


def write_results(results, file, headings):
    with open(file, mode='w', newline='') as output_file:
        writer = csv.writer(output_file, delimiter=',', quotechar='"', \
            quoting=csv.QUOTE_MINIMAL)
        writer.writerow(headings)

        for result in results:
            writer.writerow(result)


def execute_query(query, fields):
    password = config.NEO4J_PASSWORD
    graph = Graph(password=password)
    cursor = graph.run(query)
    results = []

    while cursor.forward():
        new_result = []
```

```
        for field in fields:
            new_result.append(cursor.current[field])
        results.append(new_result)

    return results


# Query 1: For each user, count his/her friends
def count_friends():
    file = "results/q1/count_friends_neo4j.csv"
    headings = ["user", "friends"]

    query = """
        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        RETURN User.userId AS user, COUNT(u) AS friends
        ORDER BY User.userId;
    """

    start_time = time.time()
    results = execute_query(query, headings)
    elapsed_time = time.time() - start_time

    write_results(results, file, headings)
    print_info(query, elapsed_time, file)


# Query 2: For each user, count his/her friends of friends
def count_friends_of_friends():
    file = "results/q2/count_friends_of_friends_neo4j.csv"
    headings = ["user", "friends_of_friends"]

    query = """
        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND*2]->(f)
        WHERE f <> User
        RETURN User.userId AS user, COUNT(f) AS friends_of_friends
        ORDER BY User.userId;
    """

    start_time = time.time()
    results = execute_query(query, headings)
```

```
    elapsed_time = time.time() - start_time

    write_results(results, file, headings)
    print_info(query, elapsed_time, file)


# Query 3: For each user, count his/her friends that are over 30
def count_friends_over_thirty():
    file = "results/q3/count_friends_over_thirty_neo4j.csv"
    headings = ["user", "friends_over_thirty"]

    query = """
        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE u.age > 30
        RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
        ORDER BY User.userId;
    """

    start_time = time.time()
    results = execute_query(query, headings)
    elapsed_time = time.time() - start_time

    write_results(results, file, headings)
    print_info(query, elapsed_time, file)


# Query 4: For each male user, count how many male and female friends
# he is having
def count_male_female_friends():
    file = "results/q4/count_male_female_friends_neo4j.csv"
    headings = ["user_male", "friends_male", "friends_female"]

    query = """
        MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE User.gender = 1
        RETURN User.userId AS user_male,
        SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
        SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
        ORDER BY User.userId;
    """

```

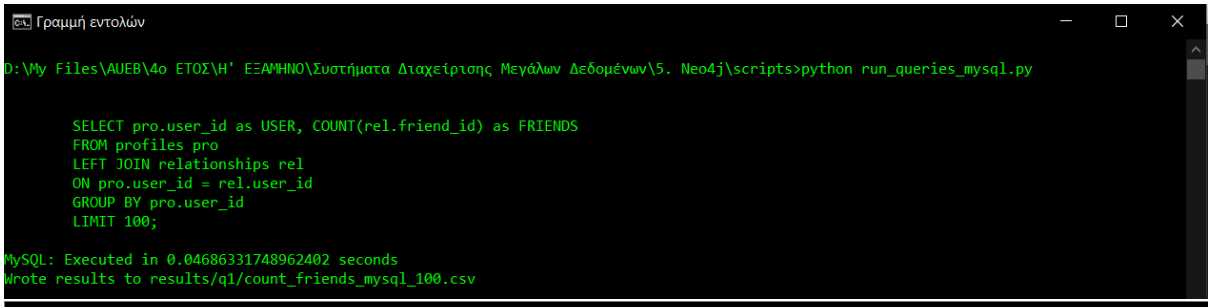
```
    start_time = time.time()
    results = execute_query(query, headings)
    elapsed_time = time.time() - start_time

    write_results(results, file, headings)
    print_info(query, elapsed_time, file)


def main():
    count_friends()
    count_friends_of_friends()
    count_friends_over_thirty()
    count_male_female_friends()


if __name__ == "__main__":

    main()
```

5 Benchmarking

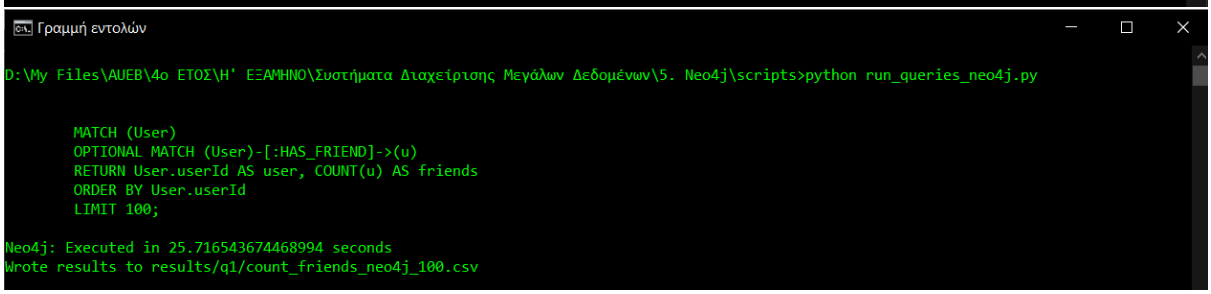
Queries were executed for 100, 1000, 10,000, 100,000 results, and finally for the entire dataset. This was preferred in order to see and compare the performance of MySQL and Neo4j for different data sizes.

5.1 Query 1: For each user, count his/her friends

5.1.1 100 Results



```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        GROUP BY pro.user_id
        LIMIT 100;

MySQL: Executed in 0.04686331748962402 seconds
Wrote results to results/q1/count_friends_mysql_100.csv
```



```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        RETURN User.userId AS user, COUNT(u) AS friends
        ORDER BY User.userId
        LIMIT 100;

Neo4j: Executed in 25.716543674468994 seconds
Wrote results to results/q1/count_friends_neo4j_100.csv
```

5.1.2 1000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        GROUP BY pro.user_id
        LIMIT 1000;

MySQL: Executed in 0.1379842758178711 seconds
Wrote results to results/q1/count_friends_mysql_1000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        RETURN User.userId AS user, COUNT(u) AS friends
        ORDER BY User.userId
        LIMIT 1000;

Neo4j: Executed in 36.73616981506348 seconds
Wrote results to results/q1/count_friends_neo4j_1000.csv
```

5.1.3 10,000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        GROUP BY pro.user_id
        LIMIT 10000;

MySQL: Executed in 2.337333917617798 seconds
Wrote results to results/q1/count_friends_mysql_10000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        RETURN User.userId AS user, COUNT(u) AS friends
        ORDER BY User.userId
        LIMIT 10000;

Neo4j: Executed in 38.42895865440369 seconds
Wrote results to results/q1/count_friends_neo4j_10000.csv
```

5.1.4 100,000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        GROUP BY pro.user_id
        LIMIT 100000;

MySQL: Executed in 22.396183729171753 seconds
Wrote results to results/q1/count_friends_mysql_100000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        RETURN User.userId AS user, COUNT(u) AS friends
        ORDER BY User.userId
        LIMIT 100000;

Neo4j: Executed in 36.49938368797302 seconds
Wrote results to results/q1/count_friends_neo4j_100000.csv
```

5.1.5 Entire Dataset

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        GROUP BY pro.user_id;

MySQL: Executed in 112.34083557128906 seconds
Wrote results to results/q1/count_friends_mysql.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        RETURN User.userId AS user, COUNT(u) AS friends
        ORDER BY User.userId;

Neo4j: Executed in 97.48987746238708 seconds
Wrote results to results/q1/count_friends_neo4j.csv
```

5.2 Query 2: For each user, count his/her friends of friends

5.2.1 100 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel2.friend_id) as FRIENDS_OF_FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel1
        ON pro.user_id = rel1.user_id
        LEFT JOIN relationships rel2
        ON rel1.friend_id = rel2.user_id
        WHERE pro.user_id != rel2.friend_id
        GROUP BY pro.user_id LIMIT 100;

MySQL: Executed in 4.0583343505859375 seconds
Wrote results to results/q2/count_friends_of_friends_mysql_100.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND*2]->(f)
        WHERE f <> User
        RETURN User.userId AS user, COUNT(f) AS friends_of_friends
        ORDER BY User.userId
        LIMIT 100;

Neo4j: Executed in 1445.773386001587 seconds
Wrote results to results/q2/count_friends_of_friends_neo4j_100.csv
```

5.2.2 1000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel2.friend_id) as FRIENDS_OF_FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel1
        ON pro.user_id = rel1.user_id
        LEFT JOIN relationships rel2
        ON rel1.friend_id = rel2.user_id
        WHERE pro.user_id != rel2.friend_id
        GROUP BY pro.user_id LIMIT 1000;

MySQL: Executed in 51.01538801193237 seconds
Wrote results to results/q2/count_friends_of_friends_mysql_1000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND*2]->(f)
        WHERE f <> User
        RETURN User.userId AS user, COUNT(f) AS friends_of_friends
        ORDER BY User.userId
        LIMIT 1000;

Neo4j: Executed in 1604.9941985607147 seconds
Wrote results to results/q2/count_friends_of_friends_neo4j_1000.csv
```

5.2.3 10,000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel2.friend_id) as FRIENDS_OF_FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel1
        ON pro.user_id = rel1.user_id
        LEFT JOIN relationships rel2
        ON rel1.friend_id = rel2.user_id
        WHERE pro.user_id != rel2.friend_id
        GROUP BY pro.user_id LIMIT 10000;

MySQL: Executed in 607.1829915046692 seconds
Wrote results to results/q2/count_friends_of_friends_mysql_10000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND*2]->(f)
        WHERE f <> User
        RETURN User.userId AS user, COUNT(f) AS friends_of_friends
        ORDER BY User.userId
        LIMIT 10000;

Neo4j: Executed in 1547.6105408668518 seconds
Wrote results to results/q2/count_friends_of_friends_neo4j_10000.csv
```

5.2.4 100,000 Results

```
Επιλογή Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel2.friend_id) as FRIENDS_OF_FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel1
        ON pro.user_id = rel1.user_id
        LEFT JOIN relationships rel2
        ON rel1.friend_id = rel2.user_id
        WHERE pro.user_id != rel2.friend_id
        GROUP BY pro.user_id LIMIT 100000;

MySQL: Executed in 5398.460721254349 seconds
Wrote results to results/q2/count_friends_of_friends_mysql_100000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND*2]->(f)
        WHERE f <> User
        RETURN User.userId AS user, COUNT(f) AS friends_of_friends
        ORDER BY User.userId
        LIMIT 100000;

Neo4j: Executed in 1550.3866097927094 seconds
Wrote results to results/q2/count_friends_of_friends_neo4j_100000.csv
```

5.2.5 Entire Dataset

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel2.friend_id) as FRIENDS_OF_FRIENDS
        FROM profiles pro
        LEFT JOIN relationships rel1
        ON pro.user_id = rel1.user_id
        LEFT JOIN relationships rel2
        ON rel1.friend_id = rel2.user_id
        WHERE pro.user_id != rel2.friend_id
        GROUP BY pro.user_id;

MySQL: Executed in 30990.84300327301 seconds
Wrote results to results/q2/count_friends_of_friends_mysql.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND*2]->(f)
        WHERE f <> User
        RETURN User.userId AS user, COUNT(f) AS friends_of_friends
        ORDER BY User.userId;

Neo4j: Executed in 1628.1876990795135 seconds
Wrote results to results/q2/count_friends_of_friends_neo4j.csv
```

5.3 Query 3: For each user, count his/her friends that are over 30

5.3.1 100 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS_OVER_THIRTY
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        AND (
            SELECT pro.age
            FROM profiles pro
            WHERE pro.user_id = rel.friend_id AND pro.age > 30)
        GROUP BY pro.user_id LIMIT 100;

MySQL: Executed in 0.3050878047943115 seconds
Wrote results to results/q3/count_friends_over_thirty_mysql_100.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE u.age > 30
        RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
        ORDER BY User.userId
        LIMIT 100;

Neo4j: Executed in 43.40518403053284 seconds
Wrote results to results/q3/count_friends_over_thirty_neo4j_100.csv
```

5.3.2 1000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS_OVER_THIRTY
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        AND (
            SELECT pro.age
            FROM profiles pro
            WHERE pro.user_id = rel.friend_id AND pro.age > 30)
        GROUP BY pro.user_id LIMIT 1000;

MySQL: Executed in 3.2243916988372803 seconds
Wrote results to results/q3/count_friends_over_thirty_mysql_1000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE u.age > 30
        RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
        ORDER BY User.userId
        LIMIT 1000;

Neo4j: Executed in 37.40926909446716 seconds
Wrote results to results/q3/count_friends_over_thirty_neo4j_1000.csv
```

5.3.3 10,000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS_OVER_THIRTY
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        AND (
            SELECT pro.age
            FROM profiles pro
            WHERE pro.user_id = rel.friend_id AND pro.age > 30)
        GROUP BY pro.user_id LIMIT 10000;

MySQL: Executed in 71.77352595329285 seconds
Wrote results to results/q3/count_friends_over_thirty_mysql_10000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE u.age > 30
        RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
        ORDER BY User.userId
        LIMIT 10000;

Neo4j: Executed in 41.22005748748779 seconds
Wrote results to results/q3/count_friends_over_thirty_neo4j_10000.csv
```

5.3.4 100,000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS_OVER_THIRTY
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        AND (
            SELECT pro.age
            FROM profiles pro
            WHERE pro.user_id = rel.friend_id AND pro.age > 30)
        GROUP BY pro.user_id LIMIT 100000;

MySQL: Executed in 964.6641092300415 seconds
Wrote results to results/q3/count_friends_over_thirty_mysql_100000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE u.age > 30
        RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
        ORDER BY User.userId
        LIMIT 100000;

Neo4j: Executed in 46.05496788024902 seconds
Wrote results to results/q3/count_friends_over_thirty_neo4j_100000.csv
```

5.3.5 Entire Dataset

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS_OVER_THIRTY
        FROM profiles pro
        LEFT JOIN relationships rel
        ON pro.user_id = rel.user_id
        AND (
            SELECT pro.age
            FROM profiles pro
            WHERE pro.user_id = rel.friend_id AND pro.age > 30)
        GROUP BY pro.user_id;

MySQL: Executed in 8873.54482960701 seconds
Wrote results to results/q3/count_friends_over_thirty_mysql.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)
        OPTIONAL MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE u.age > 30
        RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
        ORDER BY User.userId;

Neo4j: Executed in 96.23978805541992 seconds
Wrote results to results/q3/count_friends_over_thirty_neo4j.csv
```

5.4 Query 4: For each male user, count how many male and female friends he is having

5.4.1 100 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT genders.user_id as USER_MALE, COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, COUNT(CASE WHEN genders.gen
der = 0 THEN 1 END) as FRIENDS_FEMALE
        FROM (
            SELECT temp.user_id, pro.gender
            FROM profiles pro, (
                SELECT pro.user_id, pro.age, rel.friend_id
                FROM profiles pro
                LEFT JOIN relationships rel
                ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
            WHERE pro.user_id = temp.friend_id) as genders
        GROUP BY genders.user_id LIMIT 100;

MySQL: Executed in 0.42177391052246094 seconds
Wrote results to results/q4/count_male_female_friends_mysql_100.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE User.gender = 1
        RETURN User.userId AS user_male,
        SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
        SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
        ORDER BY User.userId
        LIMIT 100;

Neo4j: Executed in 44.8416645526886 seconds
Wrote results to results/q4/count_male_female_friends_neo4j_100.csv
```

5.4.2 1000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT genders.user_id as USER_MALE, COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, COUNT(CASE WHEN genders.gender = 0 TH
EN 1 END) as FRIENDS_FEMALE
        FROM (
            SELECT temp.user_id, pro.gender
            FROM profiles pro, (
                SELECT pro.user_id, pro.age, rel.friend_id
                FROM profiles pro
                LEFT JOIN relationships rel
                ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
            WHERE pro.user_id = temp.friend_id) as genders
        GROUP BY genders.user_id LIMIT 1000;

MySQL: Executed in 4.233376741409302 seconds
Wrote results to results/q4/count_male_female_friends_mysql_1000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE User.gender = 1
        RETURN User.userId AS user_male,
        SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
        SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
        ORDER BY User.userId
        LIMIT 1000;

Neo4j: Executed in 43.00988721847534 seconds
Wrote results to results/q4/count_male_female_friends_neo4j_1000.csv
```

5.4.3 10,000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT genders.user_id as USER_MALE, COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, COUNT(CASE WHEN genders.gender = 0 THEN
 1 END) as FRIENDS_FEMALE
        FROM (
            SELECT temp.user_id, pro.gender
            FROM profiles pro, (
                SELECT pro.user_id, pro.age, rel.friend_id
                FROM profiles pro
                LEFT JOIN relationships rel
                ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
            WHERE pro.user_id = temp.friend_id) as genders
        GROUP BY genders.user_id LIMIT 10000;

MySQL: Executed in 71.77998495101929 seconds
Wrote results to results/q4/count_male_female_friends_mysql_10000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE User.gender = 1
        RETURN User.userId AS user_male,
        SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
        SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
        ORDER BY User.userId
        LIMIT 10000;

Neo4j: Executed in 38.40386176109314 seconds
Wrote results to results/q4/count_male_female_friends_neo4j_10000.csv
```

5.4.4 100,000 Results

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT genders.user_id as USER_MALE, COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, COUNT(CASE WHEN genders.gender = 0 THEN 1 END)
as FRIENDS_FEMALE
        FROM (
            SELECT temp.user_id, pro.gender
            FROM profiles pro, (
                SELECT pro.user_id, pro.age, rel.friend_id
                FROM profiles pro
                LEFT JOIN relationships rel
                ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
            WHERE pro.user_id = temp.friend_id) as genders
        GROUP BY genders.user_id LIMIT 100000;

MySQL: Executed in 956.4060463905334 seconds
Wrote results to results/q4/count_male_female_friends_mysql_100000.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE User.gender = 1
        RETURN User.userId AS user_male,
        SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
        SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
        ORDER BY User.userId
        LIMIT 100000;

Neo4j: Executed in 46.279783725738525 seconds
Wrote results to results/q4/count_male_female_friends_neo4j_100000.csv
```

5.4.5 Entire Dataset

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py


        SELECT genders.user_id as USER_MALE, COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, COUNT(CASE WHEN genders.gen
der = 0 THEN 1 END) as FRIENDS_FEMALE
        FROM (
            SELECT temp.user_id, pro.gender
            FROM profiles pro, (
                SELECT pro.user_id, pro.age, rel.friend_id
                FROM profiles pro
                LEFT JOIN relationships rel
                ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
            WHERE pro.user_id = temp.friend_id) as genders
        GROUP BY genders.user_id;

MySQL: Executed in 4069.877683401108 seconds
Wrote results to results/q4/count_male_female_friends_mysql.csv
```

```
Γραμμή εντολών

D:\My Files\AUEB\4ο ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py


        MATCH (User)-[:HAS_FRIEND]->(u)
        WHERE User.gender = 1
        RETURN User.userId AS user_male,
        SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
        SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
        ORDER BY User.userId;

Neo4j: Executed in 62.813342571258545 seconds
Wrote results to results/q4/count_male_female_friends_neo4j.csv
```