

---

# 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 . . . . .	3
2	Data Loading . . . . .	4
2.1	MySQL . . . . .	4
2.2	Neo4j . . . . .	10
2.2.1	Data Transformation . . . . .	11
3	Queries . . . . .	18
3.1	MySQL . . . . .	18
3.2	Neo4j . . . . .	19
4	Execute Queries . . . . .	21
4.1	MySQL . . . . .	21
4.2	Neo4j . . . . .	25
5	Benchmarking . . . . .	29
5.1	Query 1: For each user, count his/her friends . . . . .	29
5.1.1	100 Results . . . . .	29
5.1.2	1000 Results . . . . .	30
5.1.3	10,000 Results . . . . .	30
5.1.4	100,000 Results . . . . .	31
5.1.5	Entire Dataset . . . . .	31
5.2	Query 2: For each user, count his/her friends of friends . . . . .	32
5.2.1	100 Results . . . . .	32
5.2.2	1000 Results . . . . .	33
5.2.3	10,000 Results . . . . .	34
5.2.4	100,000 Results . . . . .	35
5.2.5	Entire Dataset . . . . .	36
5.3	Query 3: For each user, count his/her friends that are over 30 . . . . .	37
5.3.1	100 Results . . . . .	37
5.3.2	1000 Results . . . . .	38

5.3.3	10,000 Results . . . . .	39
5.3.4	100,000 Results . . . . .	40
5.3.5	Entire Dataset . . . . .	41
5.4	Query 4: For each male user, count how many male and female friends he is having . . . . .	42
5.4.1	100 Results . . . . .	42
5.4.2	1000 Results . . . . .	43
5.4.3	10,000 Results . . . . .	44
5.4.4	100,000 Results . . . . .	45
5.4.5	Entire Dataset . . . . .	46

## 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. <sup>1</sup>
2. Use the Pokec dataset in Social Networks category. <sup>2</sup>
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.

---

<sup>1</sup><https://snap.stanford.edu/data/#socnets>

<sup>2</sup><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.

```

1 # MySQL
2 HOST = "localhost"
3 PORT = 3306
4 USERNAME = "root"
5 PASSWORD = "password"
6 DATABASE = "soc_pokec"
7
8 # Neo4j
9 NEO4J_PASSWORD = "password"

```

### 2.1 MySQL

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

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import config
5 import csv
6 import pymysql
7 import time
8
9
10 def write_profiles_mysql(profiles_file):
11     host = config.HOST
12     port = config.PORT
13     username = config.USERNAME
14     password = config.PASSWORD
15     database = config.DATABASE
16
17     query = "INSERT INTO 'profiles' ('user_id', 'gender', 'age') " \
18            "VALUES(%s, %s, %s)"
19
20     try:
21         connection = pymysql.connect(host=host, port=port, \
22                                     user=username, passwd=password, \
23                                     db=database)
24
25         # Prepare Cursor object

```

```

25     cursor = connection.cursor()
26
27     # Execute query
28     with open(profiles_file, "r", encoding="utf-8") as profiles:
29         csv_reader = csv.reader(profiles, delimiter = "\t")
30
31         for row in csv_reader:
32             profile = [None, None, None]
33             try:
34                 profile[0] = int(row[0])
35             except:
36                 profile[0] = -1
37             try:
38                 profile[1] = int(row[3])
39             except:
40                 profile[1] = -1
41             try:
42                 profile[2] = int(row[7])
43             except:
44                 profile[2] = -1
45             cursor.execute(query, profile)
46             print(profile)
47
48         connection.commit()
49
50     except Exception as e:
51         print(e)
52     finally:
53         connection.close()
54
55
56 def write_relationships_mysql(relationships_file):
57     host = config.HOST
58     port = config.PORT
59     username = config.USERNAME
60     password = config.PASSWORD
61     database = config.DATABASE
62
63     query = "INSERT INTO 'relationships' ('user_id', 'friend_id') " \
64            "VALUES(%s, %s)"
65
66     try:

```

```

67     connection = pymysql.connect(host=host, port=port, \
68                                 user=username, passwd=password, \
69                                 db=database)
70     # Prepare Cursor object
71     cursor = connection.cursor()
72
73     # Execute query
74     with open(relationships_file, "r", encoding="utf-8") as relationships:
75         csv_reader = csv.reader(relationships, delimiter = "\t")
76
77         for row in csv_reader:
78             relationship = [None, None]
79             try:
80                 relationship[0] = int(row[0])
81             except:
82                 relationship[0] = -1
83             try:
84                 relationship[1] = int(row[1])
85             except:
86                 relationship[1] = -1
87             cursor.execute(query, relationship)
88             print(relationship)
89
90     connection.commit()
91
92     except Exception as e:
93         print(e)
94     finally:
95         connection.close()
96
97
98 def main():
99     profiles_file = "../dataset/soc-pokec-profiles.csv"
100    relationships_file = "../dataset/soc-pokec-relationships.csv"
101
102    start_time = time.time()
103    write_profiles_mysql(profiles_file)
104    elapsed_time = time.time() - start_time
105    print("\nProfiles were inserted to MySQL \
106          in {} seconds.\n".format(elapsed_time))
107
108    start_time = time.time()

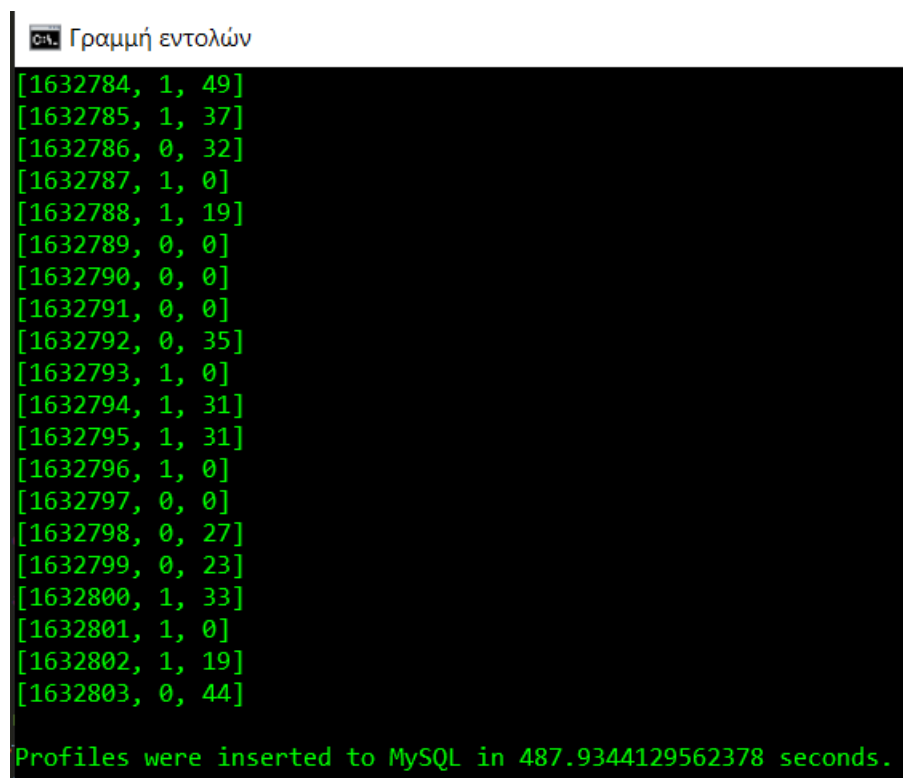
```

```

109     write_relationships_mysql(relationships_file)
110     elapsed_time = time.time() - start_time
111     print("Relationships were inserted to MySQL \
112         in {} seconds.\n".format(elapsed_time))
113
114
115 if __name__ == "__main__":
116
117     main()

```

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



ΟΛΑ Γραμμή εντολών

```

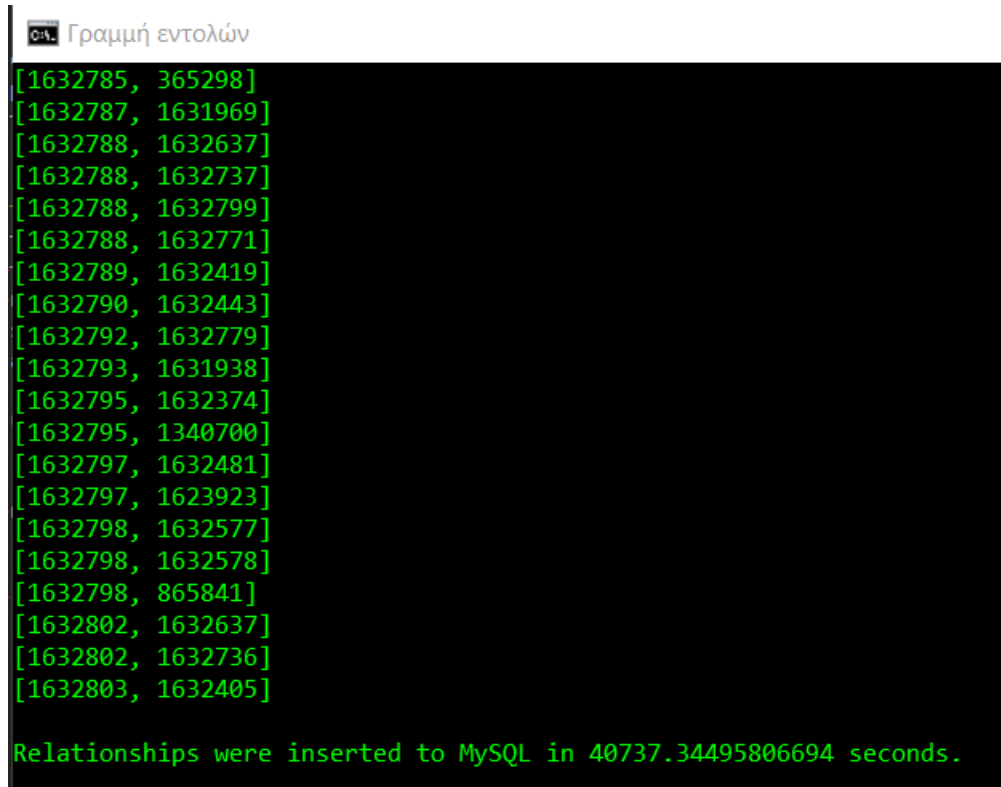
[1632784, 1, 49]
[1632785, 1, 37]
[1632786, 0, 32]
[1632787, 1, 0]
[1632788, 1, 19]
[1632789, 0, 0]
[1632790, 0, 0]
[1632791, 0, 0]
[1632792, 0, 35]
[1632793, 1, 0]
[1632794, 1, 31]
[1632795, 1, 31]
[1632796, 1, 0]
[1632797, 0, 0]
[1632798, 0, 27]
[1632799, 0, 23]
[1632800, 1, 33]
[1632801, 1, 0]
[1632802, 1, 19]
[1632803, 0, 44]

Profiles were inserted to MySQL in 487.9344129562378 seconds.

```



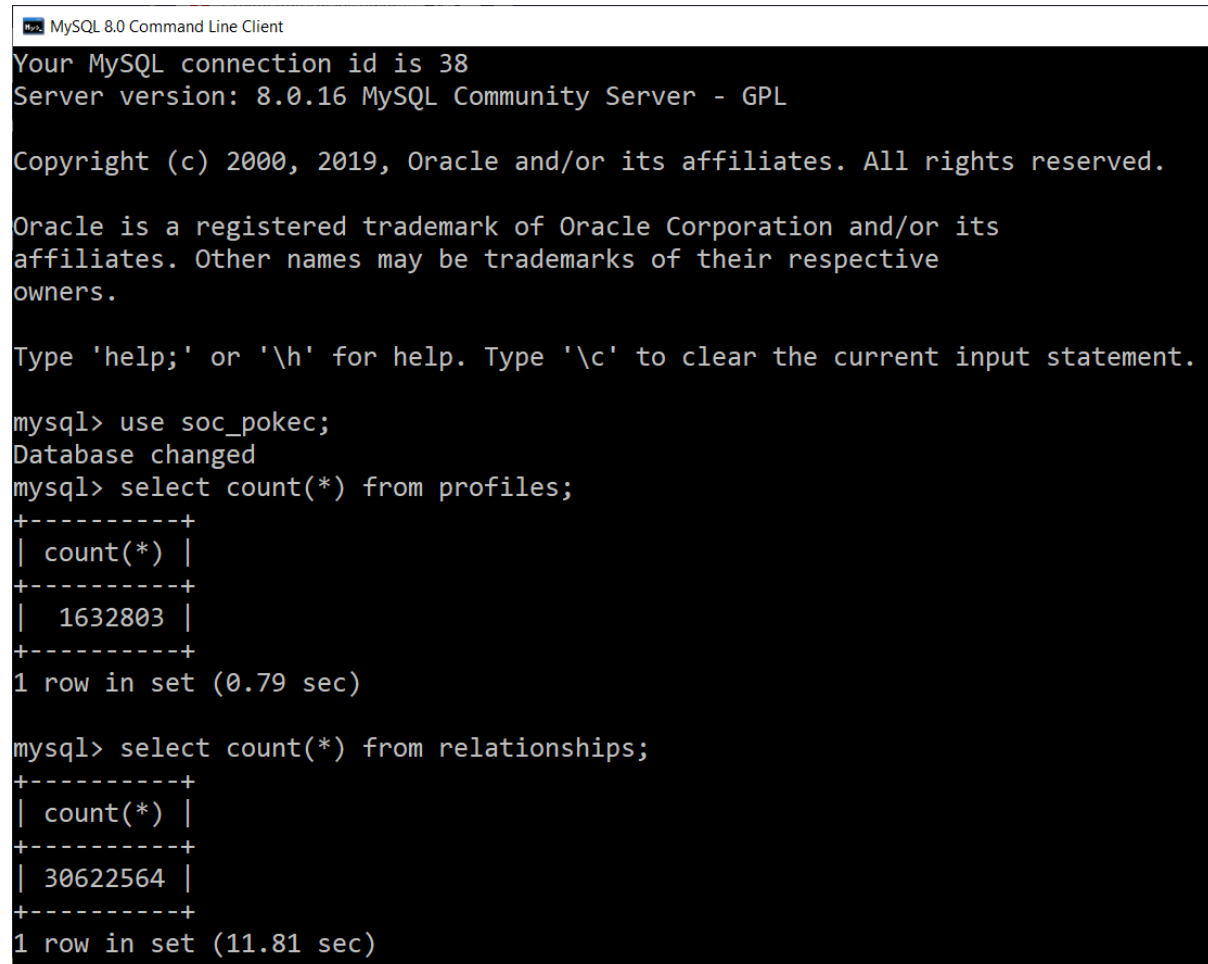
The time required for the loading of relationships to MySQL is depicted in the following figure.



```
CHL Γραμμή εντολών
[1632785, 365298]
[1632787, 1631969]
[1632788, 1632637]
[1632788, 1632737]
[1632788, 1632799]
[1632788, 1632771]
[1632789, 1632419]
[1632790, 1632443]
[1632792, 1632779]
[1632793, 1631938]
[1632795, 1632374]
[1632795, 1340700]
[1632797, 1632481]
[1632797, 1623923]
[1632798, 1632577]
[1632798, 1632578]
[1632798, 865841]
[1632802, 1632637]
[1632802, 1632736]
[1632803, 1632405]

Relationships were inserted to MySQL in 40737.34495806694 seconds.
```

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.

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import config
5 import csv
6 from py2neo import Graph
7 import time
8
9
10 def write_profiles_neo4j():
11     password = config.NEO4J_PASSWORD
12     graph = Graph(password=password)
13
14     query = """
15     USING PERIODIC COMMIT
16     LOAD CSV FROM "file:///soc-pokec-profiles.csv" \
17     AS row FIELDTERMINATOR '\t'
18     CREATE (:User {user_id: toInteger(row[0]), gender: toInteger(row[3]), \
19     age: toInteger(row[7])});
20     """
21
22     graph.run(query)
23
24
25 def write_relationships_neo4j():
26     password = config.NEO4J_PASSWORD
27     graph = Graph(password=password)
28
29     query = """
30     USING PERIODIC COMMIT
31     LOAD CSV FROM "file:///soc-pokec-relationships.csv" \
32     AS row FIELDTERMINATOR '\t'
33     MERGE (u1:User {user_id: toInteger(row[0])})
34     MERGE (u2:User {user_id: toInteger(row[1])})
35     MERGE (u1)-[:HAS_FRIEND]->(u2);
36     """
37
38     graph.run(query)
39

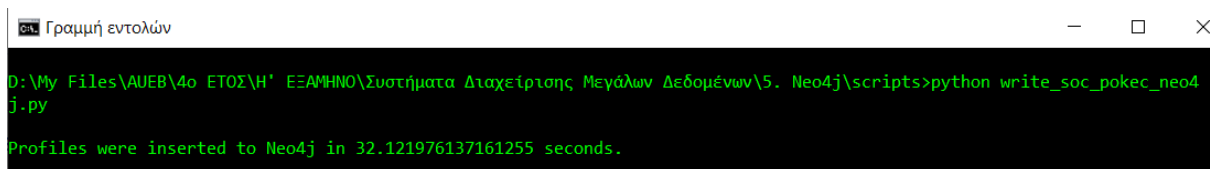
```

```

40
41 def main():
42     start_time = time.time()
43     write_profiles_neo4j()
44     elapsed_time = time.time() - start_time
45     print("\nProfiles were inserted to Neo4j \
46         in {} seconds.\n".format(elapsed_time))
47
48     start_time = time.time()
49     write_relationships_neo4j()
50     elapsed_time = time.time() - start_time
51     print("\nRelationships were inserted to Neo4j in \
52         {} seconds.\n".format(elapsed_time))
53
54
55 if __name__ == "__main__":
56
57     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.<sup>3</sup>

### 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:

<sup>3</sup><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.

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import csv
5 import time
6
7
8 def convert_profiles(profiles_file , profiles_file_new):
9     with open(profiles_file_new , "w", newline="") as file:
10         writer = csv.writer(file , delimiter=",")
11
12         writer.writerow(["userId:ID(User)", "gender:INT", "age:INT"])
13
14         with open(profiles_file , "r", encoding="utf-8") as profiles:
15             csv_reader = csv.reader(profiles , delimiter = "\t")
16
17             for row in csv_reader:
18                 profile = [None, None, None]
19                 try:
20                     profile[0] = int(row[0])
21                 except:
22                     profile[0] = -1
23                 try:
24                     profile[1] = int(row[3])
25                 except:
26                     profile[1] = -1
27                 try:
28                     profile[2] = int(row[7])
29                 except:

```

```

30         profile[2] = -1
31
32         writer.writerow(profile)
33
34
35 def convert_relationships(relationships_file, relationships_file_new):
36     with open(relationships_file_new, "w", newline="") as file:
37         writer = csv.writer(file, delimiter=",")
38
39         writer.writerow([" :START_ID(User)", " :END_ID(User)"])
40
41     with open(relationships_file, "r", encoding="utf-8") as relationships:
42         csv_reader = csv.reader(relationships, delimiter = "\t")
43
44         for row in csv_reader:
45             relationship = [None, None]
46             try:
47                 relationship[0] = int(row[0])
48             except:
49                 relationship[0] = -1
50             try:
51                 relationship[1] = int(row[1])
52             except:
53                 relationship[1] = -1
54
55             writer.writerow(relationship)
56
57
58 def main():
59     profiles_file = "../dataset/soc-pokec-profiles.csv"
60     profiles_file_new = "../dataset/soc-pokec-profiles-new.csv"
61     relationships_file = "../dataset/soc-pokec-relationships.csv"
62     relationships_file_new = "../dataset/soc-pokec-relationships-new.csv"
63
64     start_time = time.time()
65     convert_profiles(profiles_file, profiles_file_new)
66     elapsed_time_profiles = time.time() - start_time
67
68     start_time = time.time()
69     convert_relationships(relationships_file, relationships_file_new)
70     elapsed_time_relationships = time.time() - start_time
71

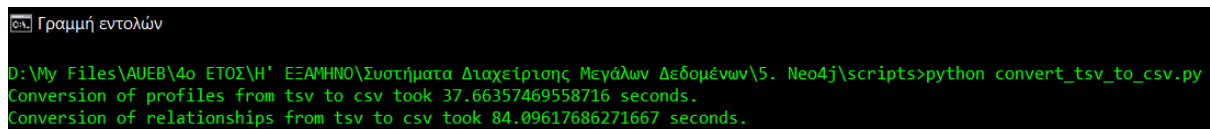
```

```

72     print("Conversion of profiles from tsv to csv \
73           took {} seconds.".format(elapsed_time_profiles))
74     print("Conversion of relationships from tsv to csv \
75           took {} seconds.".format(elapsed_time_relationships))
76
77
78 if __name__ == "__main__":
79
80     main()

```

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



```

0:\My Files\AUUEB\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.

```

C:\Users\Zwoull\NeodJDesktop\neodJdatabases\database-bce2d087-52f3-4df1-9201-93a79384937d\installation-3.5.6\bin>neodJ-admin import --id-type INTEGER --nodes:User ../import/soc-pokec-profiles-new.csv --relationships:HAS_FRIEND ../import/soc-pokec-relationships-new.csv
NeodJ version: 3.5.6
Importing the contents of these files into C:\Users\Zwoull\NeodJDesktop\neodJdatabases\database-bce2d087-52f3-4df1-9201-93a79384937d\installation-3.5.6\data\databases\graph.db:
Nodes:
  User
  C:\Users\Zwoull\NeodJDesktop\neodJdatabases\database-bce2d087-52f3-4df1-9201-93a79384937d\installation-3.5.6\bin\..\import\soc-pokec-profiles-new.csv
Relationships:
  HAS_FRIEND
  C:\Users\Zwoull\NeodJDesktop\neodJdatabases\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-ID: 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% 21s 936ms
  ..... 10% 75ms
  ..... 15% 73ms
  ..... 20% 74ms
  ..... 25% 70ms
  ..... 30% 7747ms
  ..... 35% 730ms
  ..... 40% 7205ms
  ..... 45% 70ms
  ..... 50% 70ms
  ..... 55% 70ms
  ..... 60% 7172ms
  ..... 65% 7537ms
  ..... 70% 773ms

  ..... 75% 76ms
  ..... 80% 7996ms
  ..... 85% 7873ms
  ..... 90% 7727ms
  ..... 95% 7789ms
  ..... 100% 7590ms

(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% 7848ms
  ..... 10% 7823ms
  ..... 15% 71s 192ms
  ..... 20% 7802ms
  ..... 25% 71s 148ms
  ..... 30% 71s 36ms
  ..... 35% 71s 735ms
  ..... 40% 71s 943ms
  ..... 45% 72s 117ms
  ..... 50% 71s 640ms
  ..... 55% 71s 714ms
  ..... 60% 71s 926ms
  ..... 65% 71s 801ms
  ..... 70% 72s 102ms
  ..... 75% 71s 956ms
  ..... 80% 7984ms
  ..... 85% 70ms
  ..... 90% 722ms
  ..... 95% 70ms
  ..... 100% 70ms

(3/4) Relationship linking 2019-06-17 02:45:19.182+0300
  Estimated required memory usage: 1.01 GB
  ..... 5% 7852ms
  ..... 10% 7868ms
  ..... 15% 7631ms
  ..... 20% 7714ms
  ..... 25% 71s 152ms
  ..... 30% 71s 225ms
  ..... 35% 7846ms
  ..... 40% 7855ms
  ..... 45% 7861ms
  ..... 50% 7846ms
  ..... 55% 7657ms
  ..... 60% 7644ms
  ..... 65% 71s 42ms
  ..... 70% 71s 207ms
  ..... 75% 71s 355ms
  ..... 80% 7504ms
  ..... 85% 7905ms
  ..... 90% 71s 313ms
  ..... 95% 7652ms
  ..... 100% 76s 731ms

(4/4) Post processing 2019-06-17 02:45:43.629+0300
  Estimated required memory usage: 1020.01 MB
  ..... 5% 7264ms
  ..... 10% 7208ms
  ..... 15% 7203ms
  ..... 20% 7201ms
  ..... 25% 70ms
  ..... 30% 7219ms
  ..... 35% 7204ms
  ..... 40% 7201ms
  ..... 45% 75ms
  ..... 50% 7202ms
  ..... 55% 7201ms
  ..... 60% 7211ms
  ..... 65% 70ms
  ..... 70% 7201ms
  ..... 75% 7202ms
  ..... 80% 74ms
  ..... 85% 7202ms
  ..... 90% 7228ms
  ..... 95% 70ms

IMPORT DONE in 1m 2s 212ms.
Imported:
  1632883 nodes
  30622564 relationships
  4898409 properties
Peak memory usage: 1.05 GB

```




**soc\_pokec** ooo

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.

```

1 # Query 1: For each user, count his/her friends
2
3 SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
4 FROM profiles pro
5 LEFT JOIN relationships rel
6 ON pro.user_id = rel.user_id
7 GROUP BY pro.user_id;
8
9 # Query 2: For each user, count his/her friends of friends
10
11 SELECT pro.user_id as USER, COUNT(rel2.friend_id) as FRIENDS_OF_FRIENDS
12 FROM profiles pro
13 LEFT JOIN relationships rel1
14 ON pro.user_id = rel1.user_id
15 LEFT JOIN relationships rel2
16 ON rel1.friend_id = rel2.user_id
17 WHERE pro.user_id != rel2.friend_id
18 GROUP BY pro.user_id;
19
20 # Query 3: For each user, count his/her friends that are over 30
21
22 SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS_OVER_THIRTY
23 FROM profiles pro
24 LEFT JOIN relationships rel
25 ON pro.user_id = rel.user_id
26 AND (
27     SELECT pro.age
28     FROM profiles pro
29     WHERE pro.user_id = rel.friend_id AND pro.age > 30)
30 GROUP BY pro.user_id;
31
32 # Query 4: For each male user, count how many male and female friends
33 # he is having
34
35 SELECT genders.user_id as USER_MALE, \
36 COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, \

```

```

37 COUNT(CASE WHEN genders.gender = 0 THEN 1 END) as FRIENDS_FEMALE
38 FROM (
39     SELECT temp.user_id, pro.gender
40     FROM profiles pro, (
41         SELECT pro.user_id, pro.age, rel.friend_id
42         FROM profiles pro
43         LEFT JOIN relationships rel
44         ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
45     WHERE pro.user_id = temp.friend_id) as genders
46 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.

```

1 # Query 1: For each user, count his/her friends
2
3 MATCH (User)
4 OPTIONAL MATCH (User) -[:HAS_FRIEND]->(u)
5 RETURN User.userId AS user, COUNT(u) AS friends
6 ORDER BY User.userId;
7
8 # Query 2: For each user, count his/her friends of friends
9
10 MATCH (User)
11 OPTIONAL MATCH (User) -[:HAS_FRIEND*2]->(f)
12 WHERE f <> User
13 RETURN User.userId AS user, COUNT(f) AS friends_of_friends
14 ORDER BY User.userId;
15
16 # Query 3: For each user, count his/her friends that are over 30
17
18 MATCH (User)
19 OPTIONAL MATCH (User) -[:HAS_FRIEND]->(u)
20 WHERE u.age > 30
21 RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
22 ORDER BY User.userId;
23
24 # Query 4: For each male user, count how many male and female friends
25 # he is having
26

```

```
27 MATCH (User) -[:HAS_FRIEND]->(u)
28 WHERE User.gender = 1
29 RETURN User.userId AS user_male,
30 SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
31 SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
32 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.

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import config
5 import csv
6 import pymysql
7 import time
8
9
10 def print_info(query, time, file):
11     print()
12     print(query)
13     print("MySQL: Executed in {} seconds".format(time))
14     print("Wrote results to {}".format(file))
15
16
17 def write_results(rows, file, headings):
18     with open(file, mode='w', newline='') as output_file:
19         writer = csv.writer(output_file, delimiter=',', quotechar='"', \
20                             quoting=csv.QUOTE_MINIMAL)
21         writer.writerow(headings)
22
23         for row in rows:
24             writer.writerow(row)
25
26
27 def execute_query(query):
28     host = config.HOST
29     port = config.PORT
30     username = config.USERNAME
31     password = config.PASSWORD
32     database = config.DATABASE
33
34     try:
35         connection = pymysql.connect(host=host, port=port, \

```

```

36         user=username, passwd=password, \
37         db=database)
38     # Prepare Cursor object
39     cursor = connection.cursor()
40
41     cursor.execute(query)
42     rows = cursor.fetchall()
43
44     connection.commit()
45
46 except Exception as e:
47     print(e)
48 finally:
49     connection.close()
50
51 return rows
52
53
54 # Query 1: For each user, count his/her friends
55 def count_friends():
56     file = "results/q1/count_friends_mysql.csv"
57     headings = ["user", "friends"]
58
59     query = """
60     SELECT pro.user_id as USER, COUNT(rel.friend_id) as FRIENDS
61     FROM profiles pro
62     LEFT JOIN relationships rel
63     ON pro.user_id = rel.user_id
64     GROUP BY pro.user_id;
65     """
66
67     start_time = time.time()
68     results = execute_query(query)
69     elapsed_time = time.time() - start_time
70
71     write_results(results, file, headings)
72     print_info(query, elapsed_time, file)
73
74
75 # Query 2: For each user, count his/her friends of friends
76 def count_friends_of_friends():
77     file = "results/q2/count_friends_of_friends_mysql.csv"

```

```

78     headings = ["user", "friends_of_friends"]
79
80     query = """
81         SELECT pro.user_id as USER, COUNT(rel2.friend_id)
82             as FRIENDS_OF_FRIENDS
83         FROM profiles pro
84         LEFT JOIN relationships rel1
85             ON pro.user_id = rel1.user_id
86         LEFT JOIN relationships rel2
87             ON rel1.friend_id = rel2.user_id
88         WHERE pro.user_id != rel2.friend_id
89         GROUP BY pro.user_id;
90     """
91
92     start_time = time.time()
93     results = execute_query(query)
94     elapsed_time = time.time() - start_time
95
96     write_results(results, file, headings)
97     print_info(query, elapsed_time, file)
98
99
100 # Query 3: For each user, count his/her friends that are over 30
101 def count_friends_over_thirty():
102     file = "results/q3/count_friends_over_thirty_mysql.csv"
103     headings = ["user", "friends_over_thirty"]
104
105     query = """
106         SELECT pro.user_id as USER, COUNT(rel.friend_id)
107             as FRIENDS_OVER_THIRTY
108         FROM profiles pro
109         LEFT JOIN relationships rel
110             ON pro.user_id = rel.user_id
111         AND (
112             SELECT pro.age
113             FROM profiles pro
114             WHERE pro.user_id = rel.friend_id AND pro.age > 30)
115         GROUP BY pro.user_id;
116     """
117
118     start_time = time.time()
119     results = execute_query(query)

```



```

120     elapsed_time = time.time() - start_time
121
122     write_results(results, file, headings)
123     print_info(query, elapsed_time, file)
124
125
126 # Query 4: For each male user, count how many male and female friends
127 # he is having
128 def count_male_female_friends():
129     file = "results/q4/count_male_female_friends_mysql.csv"
130     headings = ["user_male", "friends_male", "friends_female"]
131
132     query = """
133         SELECT genders.user_id as USER_MALE, \
134             COUNT(CASE WHEN genders.gender = 1 THEN 1 END) as FRIENDS_MALE, \
135             COUNT(CASE WHEN genders.gender = 0 THEN 1 END) as FRIENDS_FEMALE
136         FROM (
137             SELECT temp.user_id, pro.gender
138             FROM profiles pro, (
139                 SELECT pro.user_id, pro.age, rel.friend_id
140                 FROM profiles pro
141                 LEFT JOIN relationships rel
142                     ON pro.user_id = rel.user_id AND pro.gender = 1) as temp
143             WHERE pro.user_id = temp.friend_id) as genders
144         GROUP BY genders.user_id;
145     """
146
147     start_time = time.time()
148     results = execute_query(query)
149     elapsed_time = time.time() - start_time
150
151     write_results(results, file, headings)
152     print_info(query, elapsed_time, file)
153
154
155 def main():
156     count_friends()
157     count_friends_of_friends()
158     count_friends_over_thirty()
159     count_male_female_friends()
160
161

```

```

162 if __name__ == "__main__":
163
164     main()

```

## 4.2 Neo4j

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

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import config
5 import csv
6 from py2neo import Graph
7 import time
8
9
10 def print_info(query, time, file):
11     print()
12     print(query)
13     print("Neo4j: Executed in {} seconds".format(time))
14     print("Wrote results to {}".format(file))
15
16
17 def write_results(results, file, headings):
18     with open(file, mode='w', newline='') as output_file:
19         writer = csv.writer(output_file, delimiter=',', quotechar='"', \
20                             quoting=csv.QUOTE_MINIMAL)
21         writer.writerow(headings)
22
23         for result in results:
24             writer.writerow(result)
25
26
27 def execute_query(query, fields):
28     password = config.NEO4J_PASSWORD
29     graph = Graph(password=password)
30     cursor = graph.run(query)
31     results = []
32
33     while cursor.forward():
34         new_result = []

```

```

35         for field in fields:
36             new_result.append(cursor.current[field])
37         results.append(new_result)
38
39     return results
40
41
42 # Query 1: For each user, count his/her friends
43 def count_friends():
44     file = "results/q1/count_friends_neo4j.csv"
45     headings = ["user", "friends"]
46
47     query = """
48         MATCH (User)
49         OPTIONAL MATCH (User) -[:HAS_FRIEND]->(u)
50         RETURN User.userId AS user, COUNT(u) AS friends
51         ORDER BY User.userId;
52     """
53
54     start_time = time.time()
55     results = execute_query(query, headings)
56     elapsed_time = time.time() - start_time
57
58     write_results(results, file, headings)
59     print_info(query, elapsed_time, file)
60
61
62 # Query 2: For each user, count his/her friends of friends
63 def count_friends_of_friends():
64     file = "results/q2/count_friends_of_friends_neo4j.csv"
65     headings = ["user", "friends_of_friends"]
66
67     query = """
68         MATCH (User)
69         OPTIONAL MATCH (User) -[:HAS_FRIEND*2]->(f)
70         WHERE f <> User
71         RETURN User.userId AS user, COUNT(f) AS friends_of_friends
72         ORDER BY User.userId;
73     """
74
75     start_time = time.time()
76     results = execute_query(query, headings)

```

```

77     elapsed_time = time.time() - start_time
78
79     write_results(results, file, headings)
80     print_info(query, elapsed_time, file)
81
82
83 # Query 3: For each user, count his/her friends that are over 30
84 def count_friends_over_thirty():
85     file = "results/q3/count_friends_over_thirty_neo4j.csv"
86     headings = ["user", "friends_over_thirty"]
87
88     query = """
89         MATCH (User)
90         OPTIONAL MATCH (User) -[:HAS_FRIEND]->(u)
91         WHERE u.age > 30
92         RETURN User.userId AS user, COUNT(u) AS friends_over_thirty
93         ORDER BY User.userId;
94     """
95
96     start_time = time.time()
97     results = execute_query(query, headings)
98     elapsed_time = time.time() - start_time
99
100    write_results(results, file, headings)
101    print_info(query, elapsed_time, file)
102
103
104 # Query 4: For each male user, count how many male and female friends
105 # he is having
106 def count_male_female_friends():
107     file = "results/q4/count_male_female_friends_neo4j.csv"
108     headings = ["user_male", "friends_male", "friends_female"]
109
110    query = """
111        MATCH (User) -[:HAS_FRIEND]->(u)
112        WHERE User.gender = 1
113        RETURN User.userId AS user_male,
114            SUM(CASE WHEN (u.gender = 1) THEN 1 ELSE 0 END) AS friends_male,
115            SUM(CASE WHEN (u.gender = 0) THEN 1 ELSE 0 END) AS friends_female
116        ORDER BY User.userId;
117    """
118

```

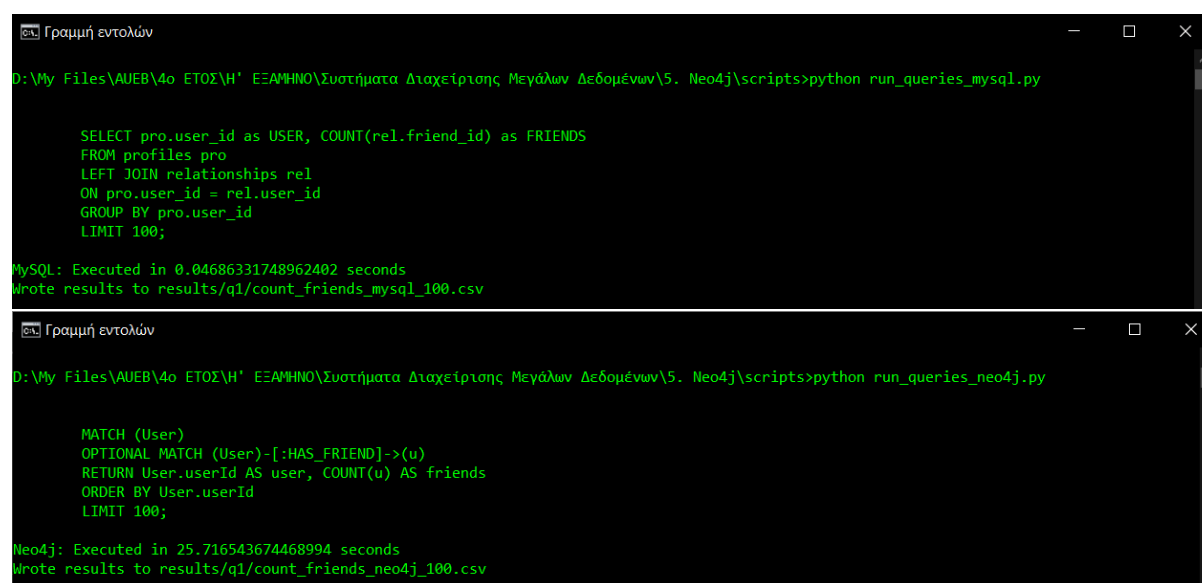
```
119     start_time = time.time()
120     results = execute_query(query, headings)
121     elapsed_time = time.time() - start_time
122
123     write_results(results, file, headings)
124     print_info(query, elapsed_time, file)
125
126
127 def main():
128     count_friends()
129     count_friends_of_friends()
130     count_friends_over_thirty()
131     count_male_female_friends()
132
133
134 if __name__ == "__main__":
135
136     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



The image shows two screenshots of a Windows command prompt window. The top screenshot shows the execution of a MySQL query. The command is `python run_queries_mysql.py`. The query is a SQL statement that selects the user ID and the count of friends for each user, limited to 100 results. The output shows the execution time in seconds and the file where the results were written.

```
0:\My Files\AUEB\4o ETOS\H' 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
```

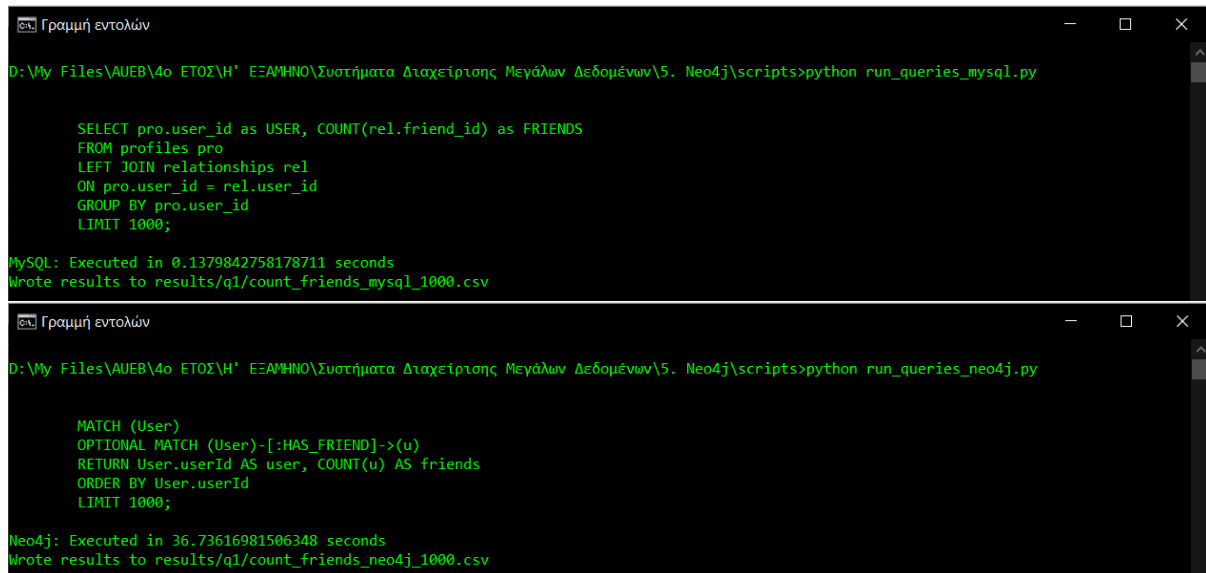
The bottom screenshot shows the execution of a Neo4j query. The command is `python run_queries_neo4j.py`. The query is a Cypher statement that matches users and counts their friends, limited to 100 results. The output shows the execution time in seconds and the file where the results were written.

```
0:\My Files\AUEB\4o ETOS\H' 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



The image shows two screenshots of a Windows command prompt window titled "Γραμμή εντολών". The first screenshot shows the execution of a Python script that runs a MySQL query. The query is a SQL statement that selects the user ID and the count of friends for each user, limited to 1000 results. The output shows the query was executed in 0.1379842758178711 seconds and the results were written to a CSV file. The second screenshot shows the execution of a Python script that runs a Neo4j query. The query is a Cypher statement that matches users and counts their friends, limited to 1000 results. The output shows the query was executed in 36.73616981506348 seconds and the results were written to a CSV file.

```
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;

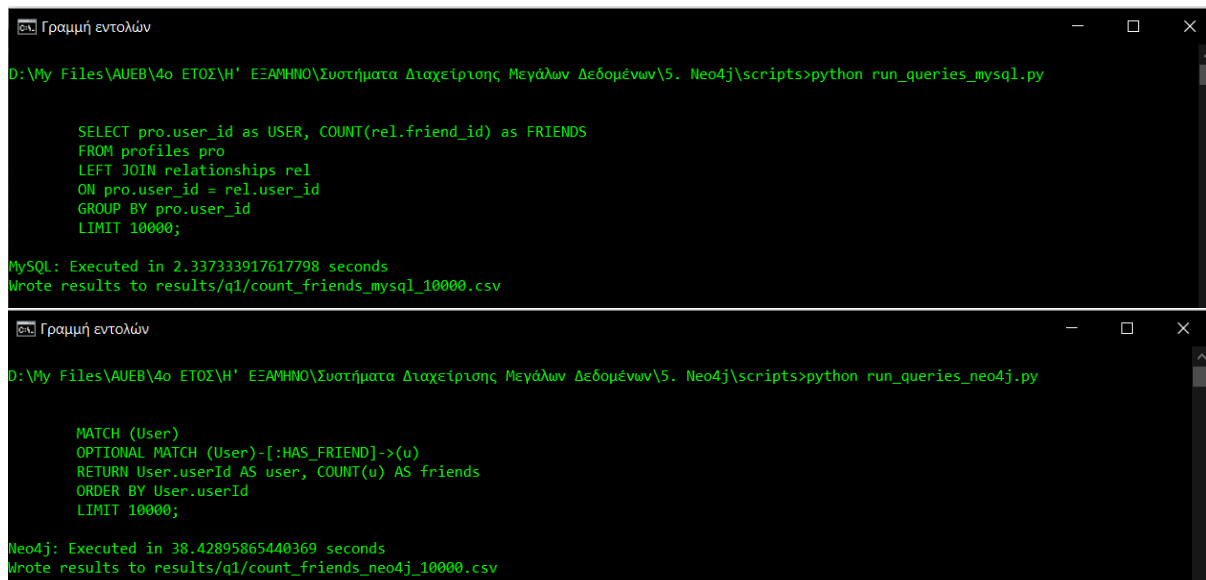
MySQL: Executed in 0.1379842758178711 seconds
Wrote results to results/q1/count_friends_mysql_1000.csv
```

```
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



The image shows two screenshots of a Windows command prompt window titled "Γραμμή εντολών". The first screenshot shows the execution of a Python script that runs a MySQL query. The query is a SQL statement that selects the user ID and the count of friends for each user, limited to 10000 results. The output shows the query was executed in 2.337333917617798 seconds and the results were written to a CSV file. The second screenshot shows the execution of a Python script that runs a Neo4j query. The query is a Cypher statement that matches users and counts their friends, limited to 10000 results. The output shows the query was executed in 38.42895865440369 seconds and the results were written to a CSV file.

```
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
```

```
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

The image shows two screenshots of a Windows command prompt window. The first screenshot shows a MySQL query being executed. The query is a SELECT statement that joins the 'profiles' table (aliased as 'pro') with the 'relationships' table (aliased as 'rel') on the 'user\_id' field. It counts the number of friends for each user, limiting the results to 100,000. The output shows the query was executed in 22.396183729171753 seconds and the results were written to 'results/q1/count\_friends\_mysql\_100000.csv'. The second screenshot shows a Neo4j query being executed. The query is a MATCH statement that finds all users and counts their friends, also limiting the results to 100,000. The output shows the query was executed in 36.49938368797302 seconds and the results were written to 'results/q1/count\_friends\_neo4j\_100000.csv'.

```

D:\My Files\AUEB\4o 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

D:\My Files\AUEB\4o 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

The image shows two screenshots of a Windows command prompt window. The first screenshot shows a MySQL query being executed. The query is a SELECT statement that joins the 'profiles' table (aliased as 'pro') with the 'relationships' table (aliased as 'rel') on the 'user\_id' field. It counts the number of friends for each user, limiting the results to 100,000. The output shows the query was executed in 112.34083557128906 seconds and the results were written to 'results/q1/count\_friends\_mysql.csv'. The second screenshot shows a Neo4j query being executed. The query is a MATCH statement that finds all users and counts their friends, also limiting the results to 100,000. The output shows the query was executed in 97.48987746238708 seconds and the results were written to 'results/q1/count\_friends\_neo4j.csv'.

```

D:\My Files\AUEB\4o 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

D:\My Files\AUEB\4o 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;

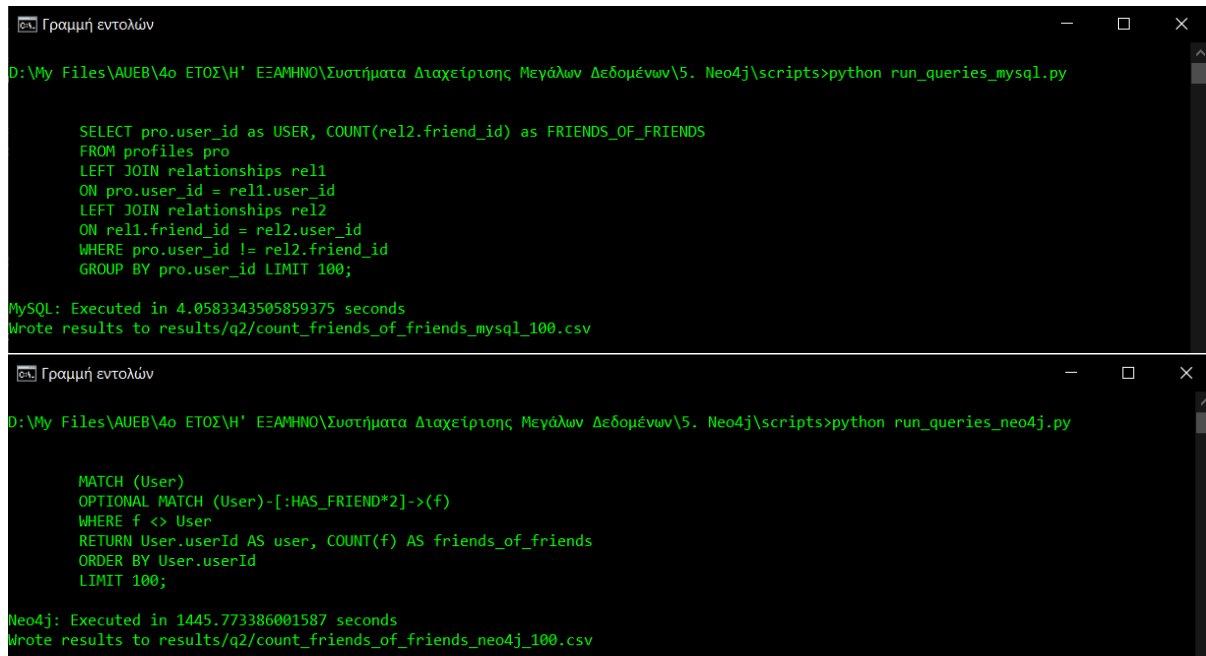
Neo4j: Executed in 97.48987746238708 seconds
Wrote results to results/q1/count_friends_neo4j.csv

```



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

### 5.2.1 100 Results



The image shows two screenshots of a Windows command prompt window. The top screenshot shows the execution of a Python script that runs a MySQL query. The query is a complex JOIN query that counts the number of friends of friends for each user. The bottom screenshot shows the execution of a Python script that runs a Neo4j query. The query is a MATCH query that counts the number of friends of friends for each user.

```
0:\My Files\AUEB\4o 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;

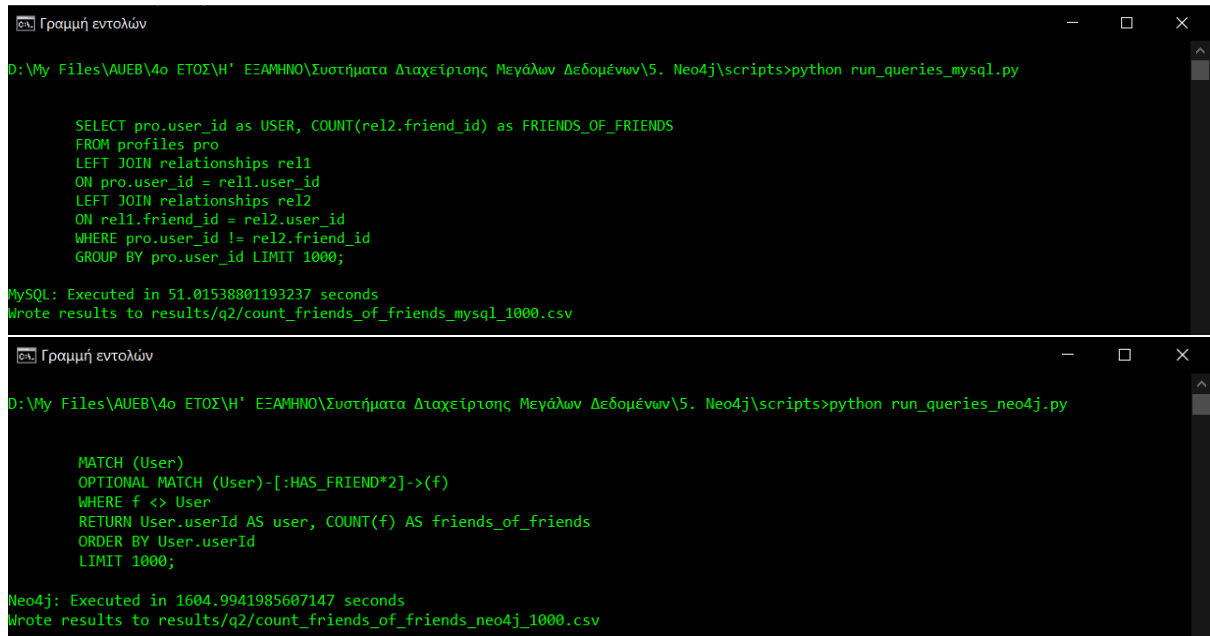
MySQL: Executed in 4.0583343505859375 seconds
Wrote results to results/q2/count_friends_of_friends_mysql_100.csv
```

```
0:\My Files\AUEB\4o 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



The image shows two screenshots of a Windows command prompt window. The first screenshot shows a MySQL query being executed. The query is a SQL statement that joins the 'profiles' table with the 'relationships' table twice to find the number of friends for each user. The query is:   
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;  
The output shows the query was executed in 51.01538801193237 seconds and the results were written to a CSV file named 'results/q2/count\_friends\_of\_friends\_mysql\_1000.csv'.  
The second screenshot shows a Neo4j query being executed. The query is a Cypher statement that matches users and counts their friends. The query is:   
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;  
The output shows the query was executed in 1604.9941985607147 seconds and the results were written to a CSV file named 'results/q2/count\_friends\_of\_friends\_neo4j\_1000.csv'.

```
Γραμμή εντολών
D:\My Files\AUEB\4o 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;

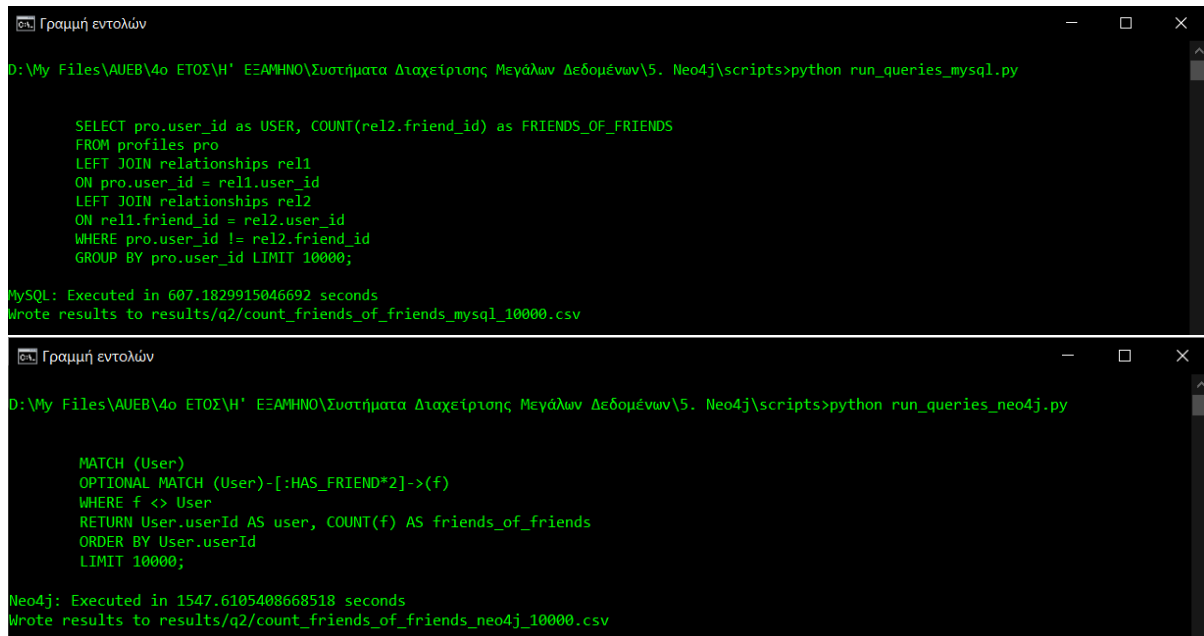
MySQL: Executed in 51.01538801193237 seconds
Wrote results to results/q2/count_friends_of_friends_mysql_1000.csv

Γραμμή εντολών
D:\My Files\AUEB\4o 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



The image shows two screenshots of a Windows command prompt window. The top screenshot shows the execution of a MySQL query. The command is `python run_queries_mysql.py`. The query is a SQL statement that selects the user ID and the count of friends of friends for 10,000 users. The output shows the query was executed in 607.1829915046692 seconds and the results were written to `results/q2/count_friends_of_friends_mysql_10000.csv`. The bottom screenshot shows the execution of a Neo4j query. The command is `python run_queries_neo4j.py`. The query is a Cypher statement that matches users and counts their friends of friends. The output shows the query was executed in 1547.6105408668518 seconds and the results were written to `results/q2/count_friends_of_friends_neo4j_10000.csv`.

```
Γραμμή εντολών
D:\My Files\AUEB\4o 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;

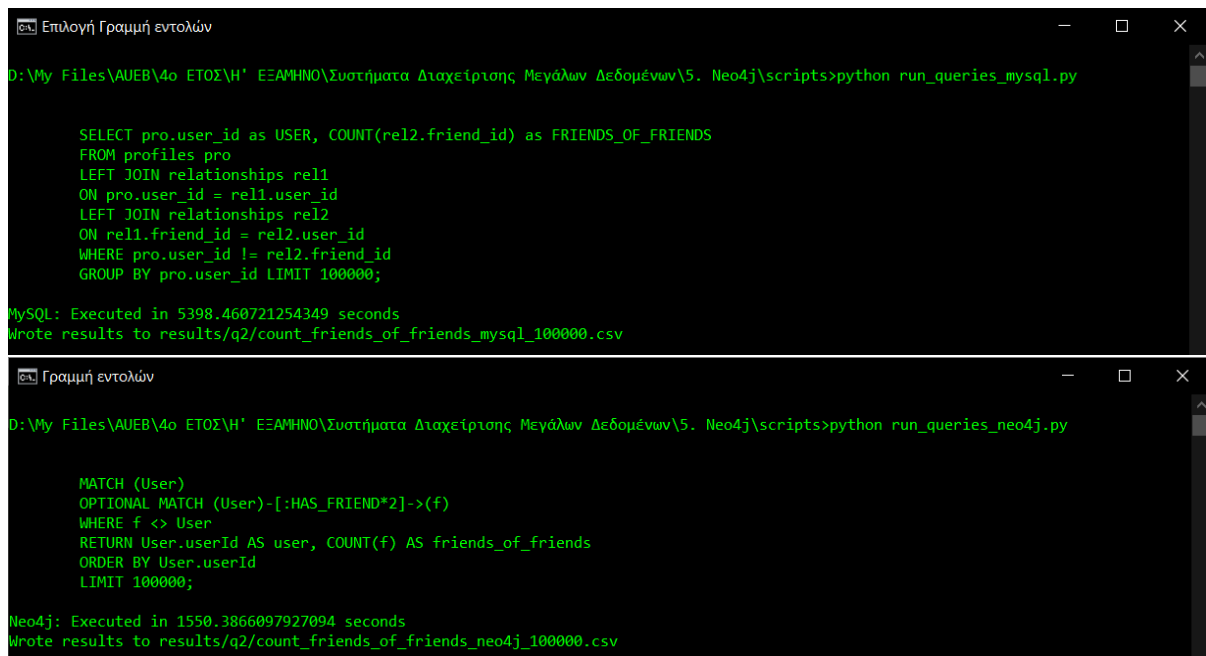
MySQL: Executed in 607.1829915046692 seconds
Wrote results to results/q2/count_friends_of_friends_mysql_10000.csv

Γραμμή εντολών
D:\My Files\AUEB\4o 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;

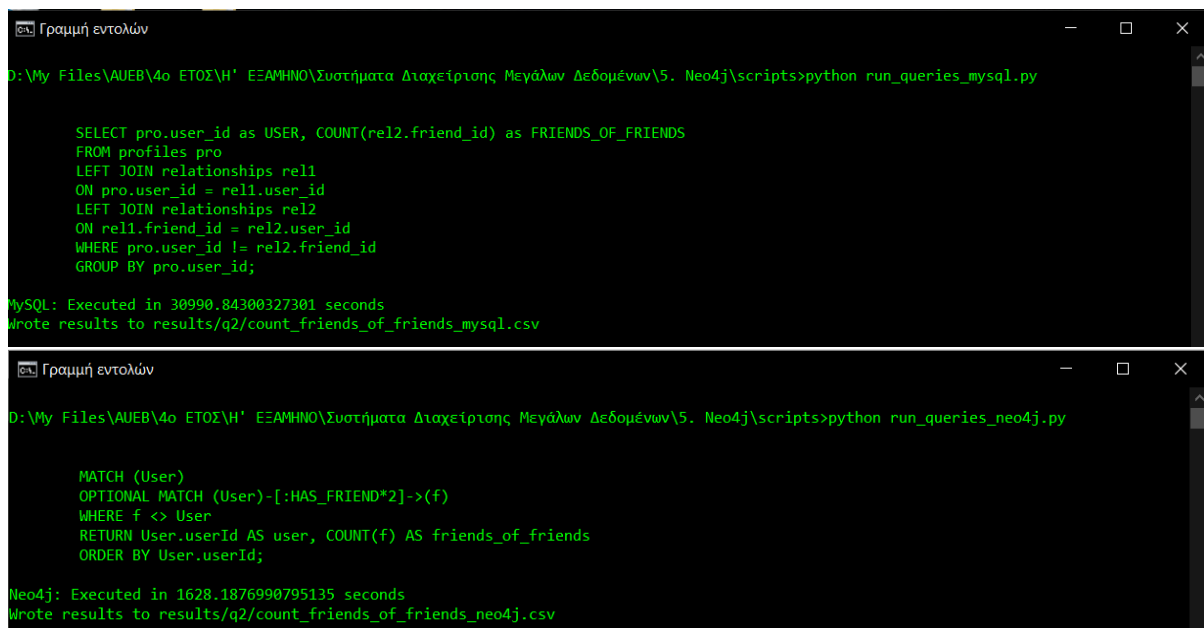
MySQL: Executed in 5398.460721254349 seconds
Wrote results to results/q2/count_friends_of_friends_mysql_100000.csv
```

```
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 1550.386097927094 seconds
Wrote results to results/q2/count_friends_of_friends_neo4j_100000.csv
```

### 5.2.5 Entire Dataset



The image shows two screenshots of a Windows command prompt window. The top screenshot shows the execution of a MySQL query using a Python script. The query is a SQL statement that counts the number of friends for each user by joining the 'profiles' and 'relationships' tables. The output shows the query was executed in 30990.84300327301 seconds and the results were written to a CSV file. The bottom screenshot shows the execution of a Neo4j query using a Python script. The query is a Cypher statement that counts the number of friends for each user by matching the 'User' node and counting the 'f' nodes. The output shows the query was executed in 1628.1876990795135 seconds and the results were written to a CSV file.

```
D:\My Files\AUEB\4o 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
```

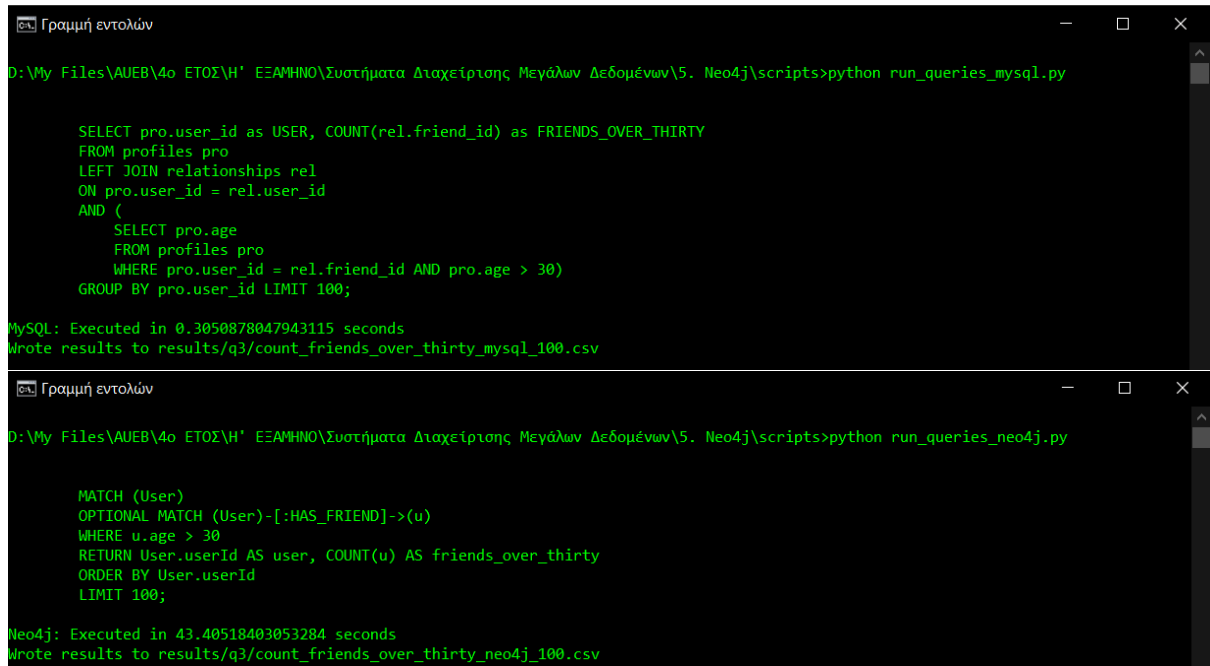
```
D:\My Files\AUEB\4o ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py

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



The image shows two screenshots of a Windows command prompt window. The top screenshot shows the execution of a Python script that runs a MySQL query. The query selects the user ID and the count of friends over 30 from the profiles and relationships tables. The bottom screenshot shows the execution of a Python script that runs a Neo4j Cypher query. The query matches users with at least 30 friends and returns their user ID and the count of friends over 30.

```
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;

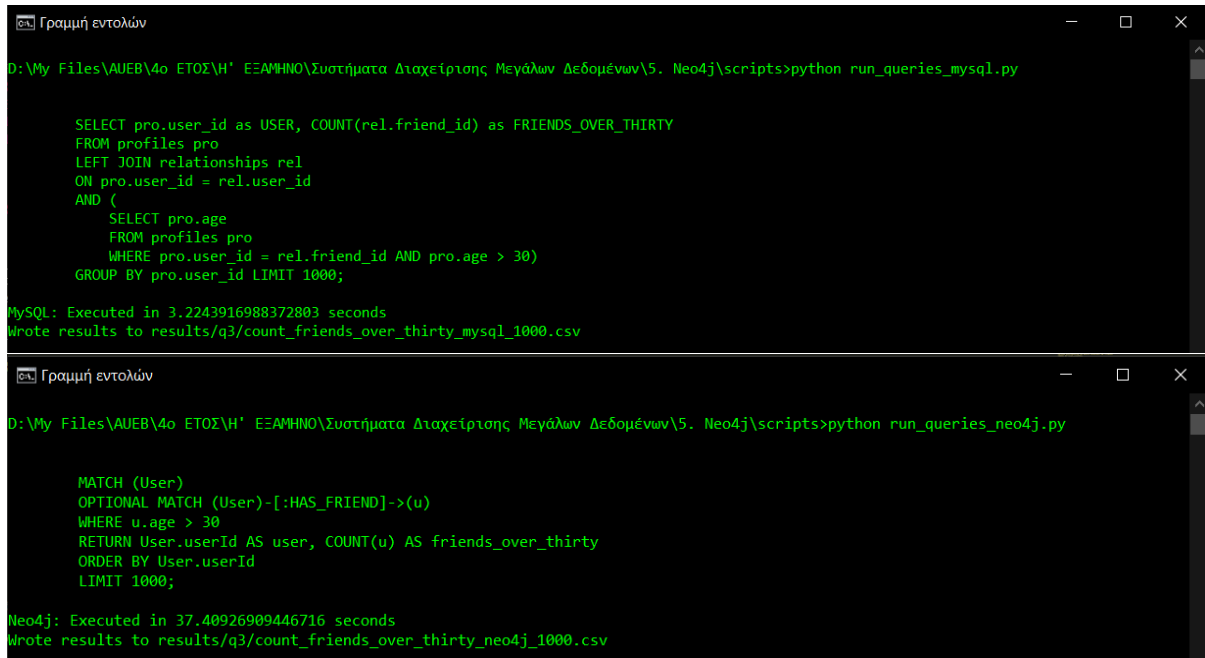
MySQL: Executed in 0.3050878047943115 seconds
Wrote results to results/q3/count_friends_over_thirty_mysql_100.csv

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

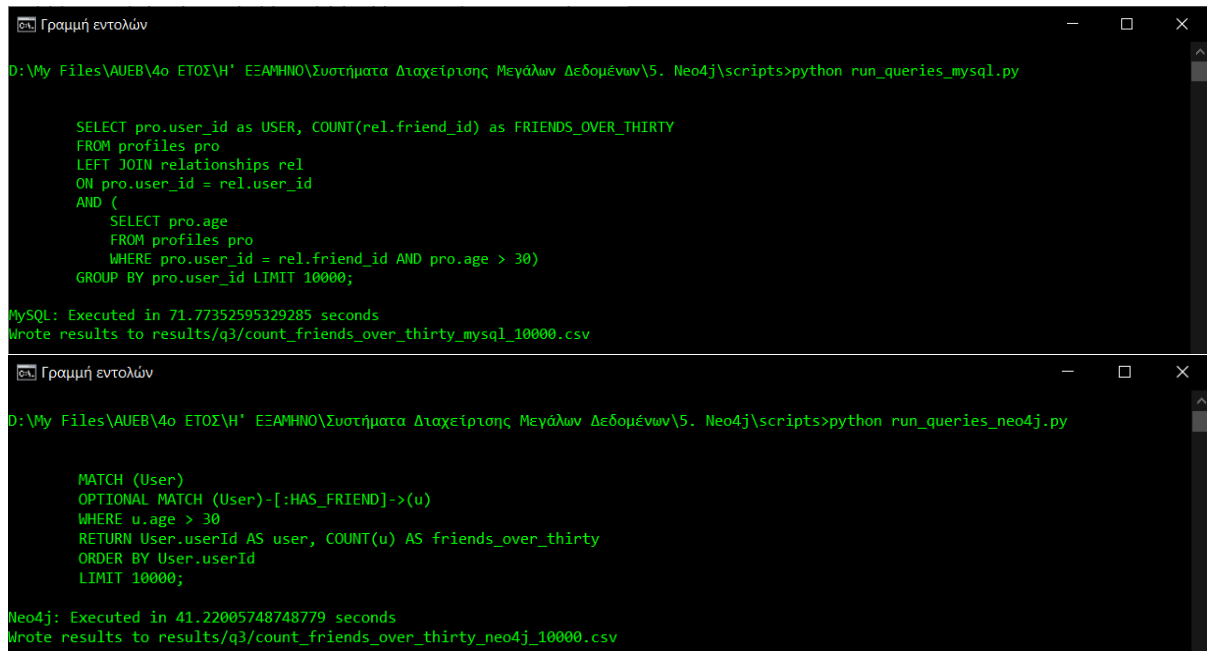


The image shows two screenshots of a Windows command prompt window titled "Γραμμή εντολών".

The first screenshot shows the execution of a Python script `run_queries_mysql.py`. The command is `D:\My Files\AUEB\4o ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py`. The script executes a MySQL 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 LIMIT 1000;`. The output shows the query was executed in 3.2243916988372803 seconds and results were written to `results/q3/count_friends_over_thirty_mysql_1000.csv`.

The second screenshot shows the execution of a Python script `run_queries_neo4j.py`. The command is `D:\My Files\AUEB\4o ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py`. The script executes a Neo4j 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 LIMIT 1000;`. The output shows the query was executed in 37.40926909446716 seconds and results were written to `results/q3/count_friends_over_thirty_neo4j_1000.csv`.

### 5.3.3 10,000 Results



The image shows two screenshots of a Windows command prompt window titled "Γραμμή εντολών".

The first screenshot shows the execution of a Python script that runs a MySQL query. The command is: `D:\My Files\AUEB\4o ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_mysql.py`. The query is a MySQL SELECT statement that counts the number of friends over 30 for each user. The output shows the query was executed in 71.77352595329285 seconds and the results were written to `results/q3/count_friends_over_thirty_mysql_10000.csv`.

The second screenshot shows the execution of a Python script that runs a Neo4j query. The command is: `D:\My Files\AUEB\4o ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py`. The query is a Cypher MATCH statement that counts the number of friends over 30 for each user. The output shows the query was executed in 41.22005748748779 seconds and the results were written to `results/q3/count_friends_over_thirty_neo4j_10000.csv`.

```
Γραμμή εντολών
D:\My Files\AUEB\4o 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

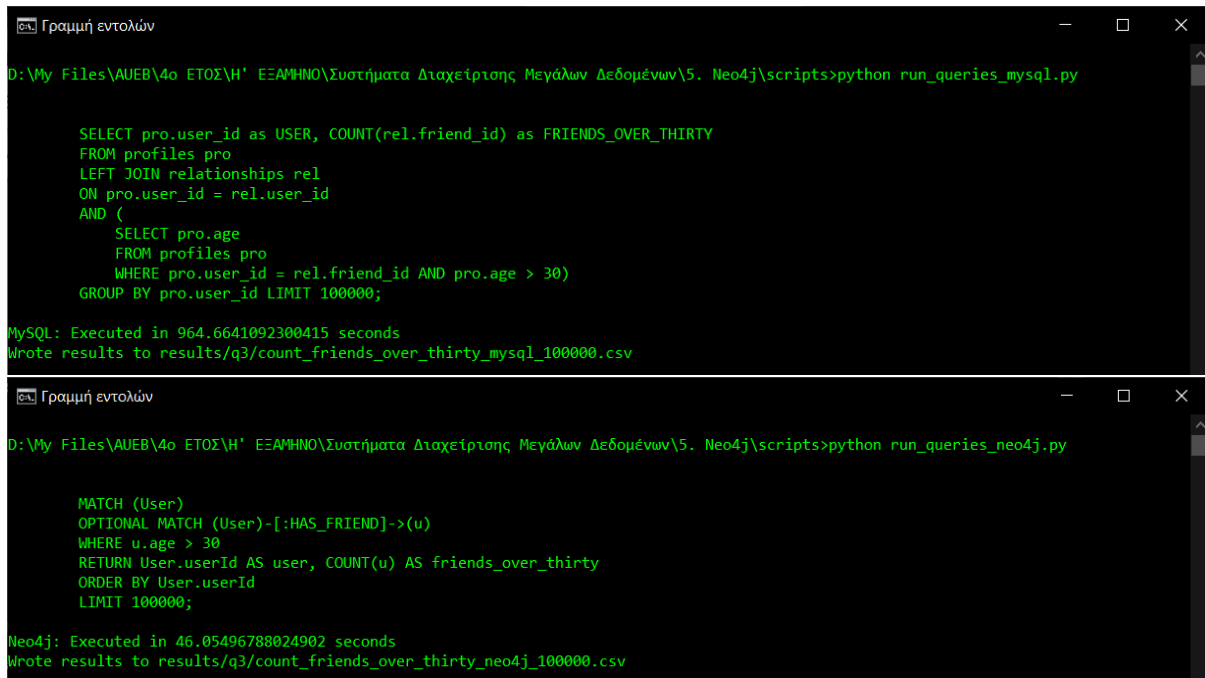
Γραμμή εντολών
D:\My Files\AUEB\4o ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py

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



The image shows two separate command-line windows. The top window is titled 'Γραμμή εντολών' (Command Line) and shows a Python script running a MySQL query. The query is a complex SQL statement that joins 'profiles' and 'relationships' tables to find users with more than 30 friends, ordered by the number of friends in descending order, limited to 100,000 results. The output shows the execution time in seconds and the file where results were written. The bottom window is also titled 'Γραμμή εντολών' and shows a Python script running a Cypher query. The query is a Cypher statement that matches users with more than 30 friends, ordered by the number of friends in descending order, limited to 100,000 results. The output shows the execution time in seconds and the file where results were written.

```
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;

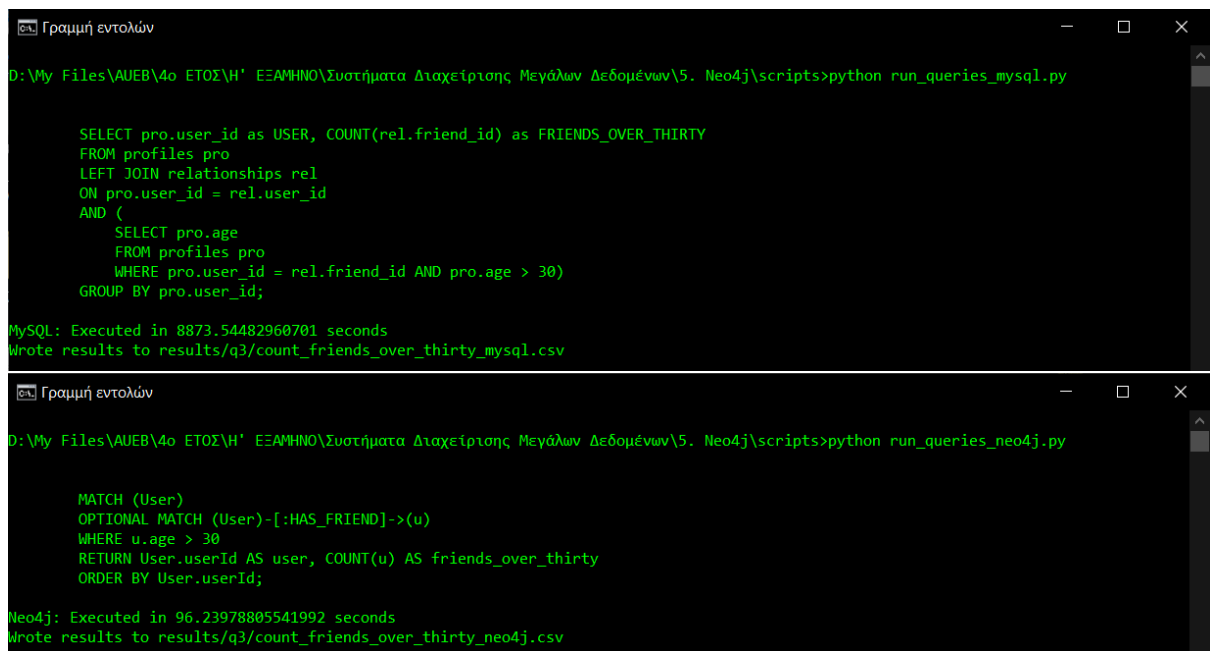
MySQL: Executed in 964.6641092300415 seconds
Wrote results to results/q3/count_friends_over_thirty_mysql_100000.csv
```

```
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



The image shows two screenshots of a Windows command prompt window titled "Γραμμή εντολών". The first screenshot shows a MySQL query being executed via a Python script. The query is a SQL SELECT statement that joins the 'profiles' and 'relationships' tables to count the number of friends over 30 for each user. The output shows the query was executed in 8873.54482960701 seconds and results were written to 'results/q3/count\_friends\_over\_thirty\_mysql.csv'. The second screenshot shows a Neo4j query being executed via a Python script. The query is a Cypher MATCH statement that finds users with more than 30 friends. The output shows the query was executed in 96.23978805541992 seconds and results were written to 'results/q3/count\_friends\_over\_thirty\_neo4j.csv'.

```
D:\My Files\AU\EB\4o 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

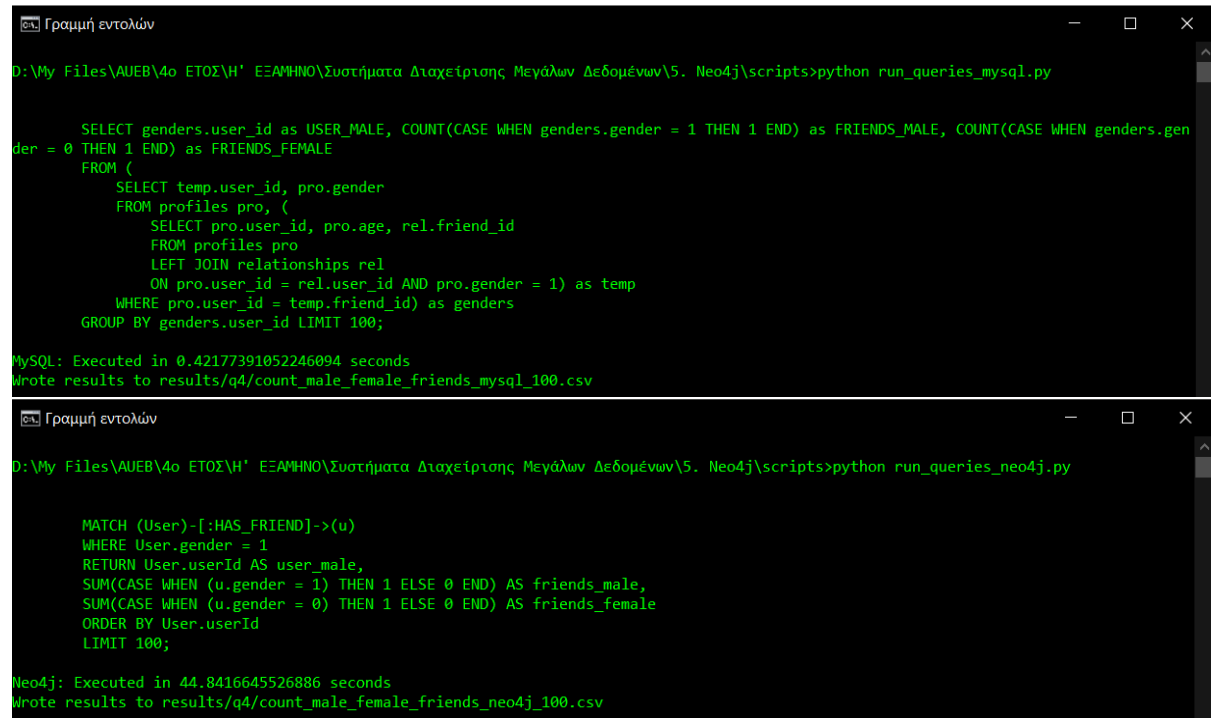
D:\My Files\AU\EB\4o ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py

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



The image shows two screenshots of a Windows command prompt window. The top screenshot shows the execution of a MySQL query. The bottom screenshot shows the execution of a Neo4j query.

**Top Screenshot (MySQL):**

```

D:\My Files\AUEB\4o ETOS\H' 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
  
```

**Bottom Screenshot (Neo4j):**

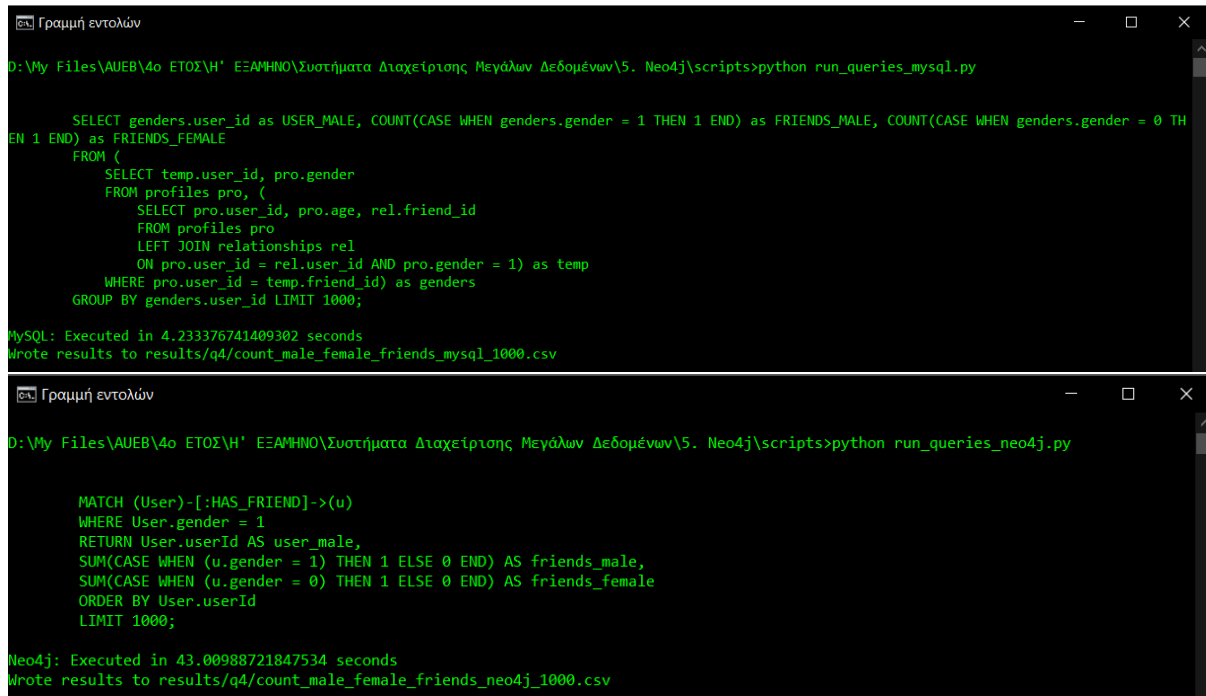
```

D:\My Files\AUEB\4o ETOS\H' 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



The image contains two screenshots of a Windows command prompt window. The top screenshot shows the execution of a Python script that runs a MySQL query. The query is a complex SQL statement that selects user IDs and counts friends by gender, limited to 1000 results. The output shows the execution time in seconds and the file where results were written. The bottom screenshot shows the execution of a Python script that runs a Neo4j query. The query is a Cypher statement that matches users with friends and returns their IDs and friend counts by gender, limited to 1000 results. The output shows the execution time in seconds and the file where results were written.

```

D:\My Files\AUEB\4o 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

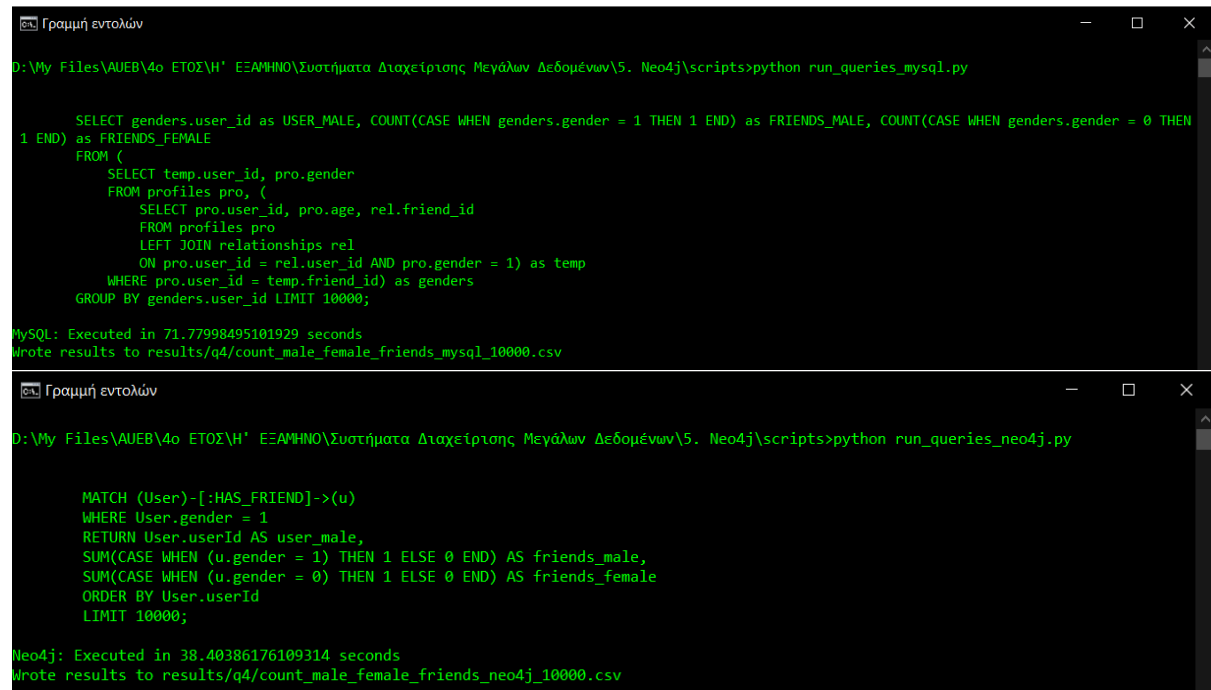
D:\My Files\AUEB\4o ΕΤΟΣ\Η' ΕΞΑΜΗΝΟ\Συστήματα Διαχείρισης Μεγάλων Δεδομένων\5. Neo4j\scripts>python run_queries_neo4j.py

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



The image displays two screenshots of a Windows command prompt window, titled "Γραμμή εντολών" (Command Line). The first screenshot shows the execution of a Python script that runs a MySQL query. The query is a complex SQL statement designed to count the number of male and female friends for each user, limited to 10,000 results. The output indicates that the query was executed in 71.77998495101929 seconds and the results were written to a CSV file named "results/q4/count\_male\_female\_friends\_mysql\_10000.csv". The second screenshot shows the execution of a similar Python script that runs a Neo4j query. The query is a Cypher statement that matches users with friends and counts the number of male and female friends, also limited to 10,000 results. The output indicates that the query was executed in 38.40386176109314 seconds and the results were written to a CSV file named "results/q4/count\_male\_female\_friends\_neo4j\_10000.csv".

```
D:\My Files\AUEB\4o ETOS\H' 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;

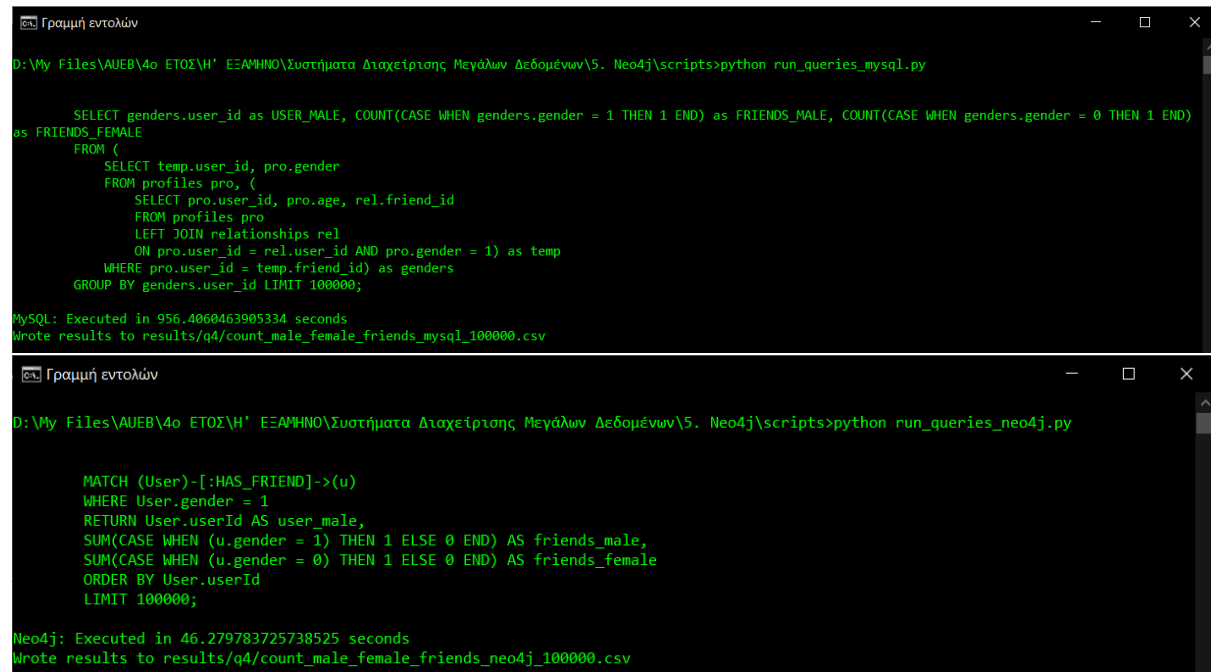
MySQL: Executed in 71.77998495101929 seconds
Wrote results to results/q4/count_male_female_friends_mysql_10000.csv

D:\My Files\AUEB\4o ETOS\H' 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



The image shows two screenshots of a Windows command prompt window. The top screenshot shows the execution of a Python script that runs a MySQL query. The query is a complex SQL statement that selects user IDs, counts friends by gender, and limits the results to 100,000. The output shows the query was executed in 956.4060463905334 seconds and the results were written to a CSV file. The bottom screenshot shows the execution of a Python script that runs a Neo4j query. The query is a Cypher statement that matches users with friends, filters by gender, and returns the results ordered by user ID, limited to 100,000. The output shows the query was executed in 46.279783725738525 seconds and the results were written to a CSV file.

```
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;

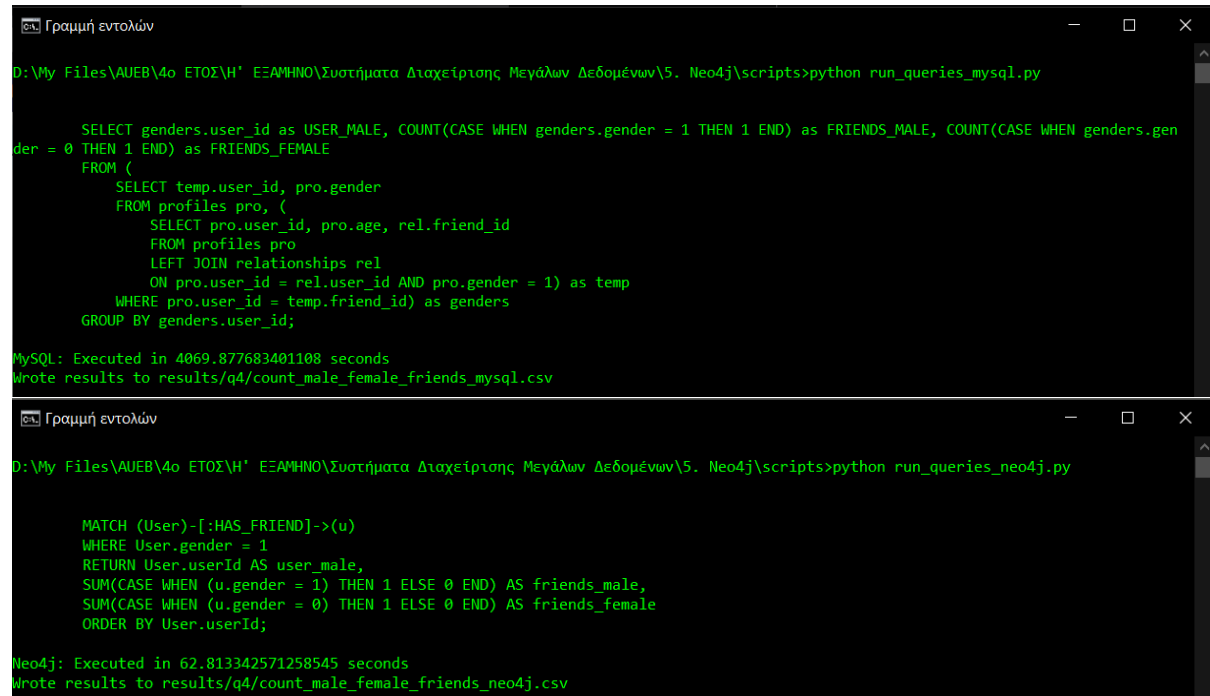
MySQL: Executed in 956.4060463905334 seconds
Wrote results to results/q4/count_male_female_friends_mysql_100000.csv

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



The image shows two screenshots of a Windows command prompt window. The top screenshot shows a MySQL query being executed. The query is a complex SQL statement that selects user IDs as USER\_MALE, counts friends for males as FRIENDS\_MALE, and counts friends for females as FRIENDS\_FEMALE. It uses a subquery to join profiles and relationships tables. The execution time is 4069.877683401108 seconds, and the results are written to results/q4/count\_male\_female\_friends\_mysql.csv. The bottom screenshot shows a Neo4j Cypher query being executed. The query matches users with a friend relationship, filters for males, and returns user IDs, counts of male friends, and counts of female friends. The execution time is 62.813342571258545 seconds, and the results are written to results/q4/count\_male\_female\_friends\_neo4j.csv.

```
0:\My Files\AUEB\4o ETOS\H' 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

0:\My Files\AUEB\4o ETOS\H' 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
```