BUDDY-BLOOM: MODELING SOCIAL CONNECTIONS THROUGH GRAPH-BASED DATA REPRESENTATION IN NEO4J

PROJECT REPORT

PRESENTED TO
DR. SUNEUY KIM

DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

IN FULFILLMENT

OF THE REQUIREMENTS FOR THE CLASS CS157C

BY

RUDRAKSH NAIK [rudrakshpushkar.naik@sjsu.edu]

CHANDRAKANTH CHALLA [chandrakanthreddy.challa@sjsu.edu]

SHANTANU KHANAPURE [shantanushivraj.khanapure@sjsu.edu]

DECEMBER 2025

INDEX

1. Property Graph Schema

Node Properties (:User)

userId (string): canonical UUID/ID used by the application and stored/returned by DB queries.

username (string, unique): human-friendly identifier; uniqueness enforced.

passwordHash (string): bcrypt hash for authentication.

name (string)

email (string)

bio (string)

version (int)

followersCount (int, denormalized): maintained on the node to avoid runtime counting.

followingCount (int, denormalized): maintained on the node.

Relationship Properties (:FOLLOWS)

since (datetime): timestamp set on creation (ON CREATE SET f.since = datetime()).

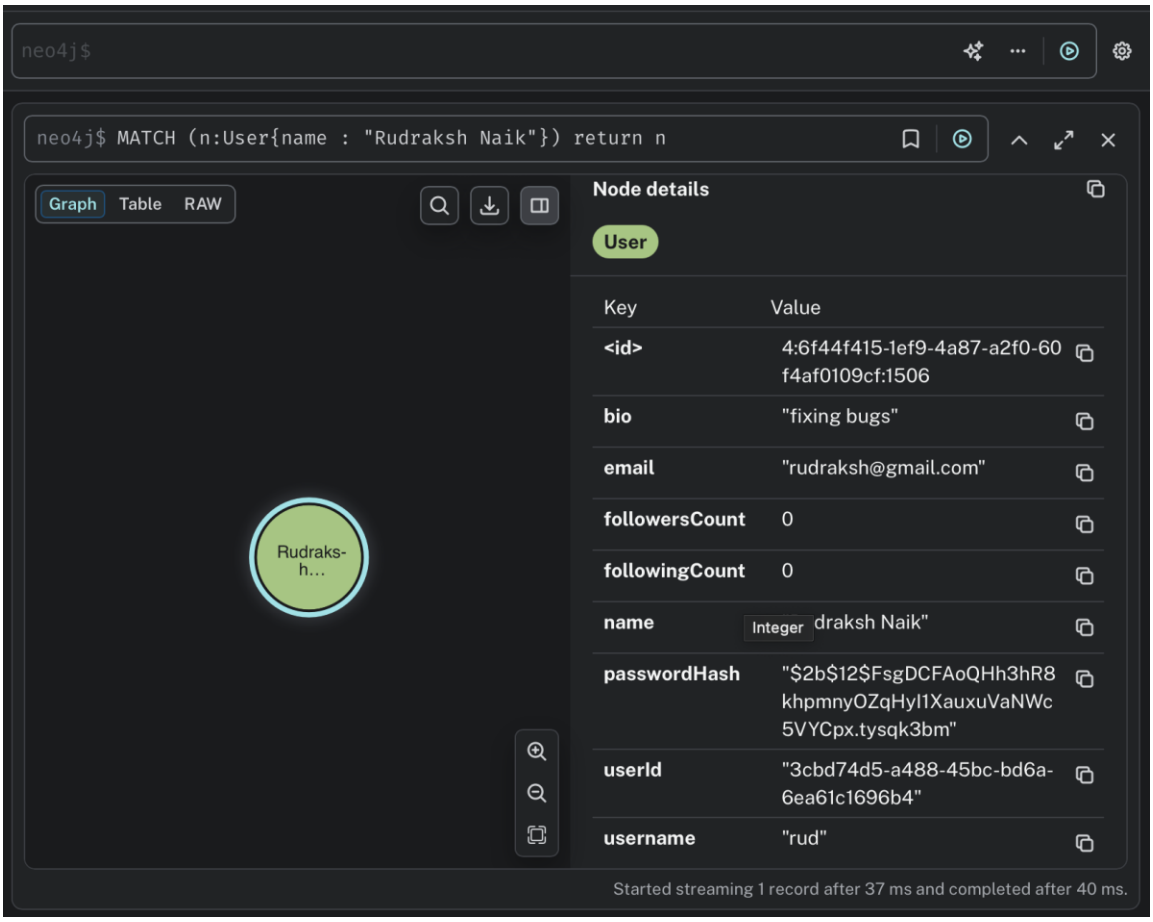
No additional relationship-level counters; node-level counts are used instead.

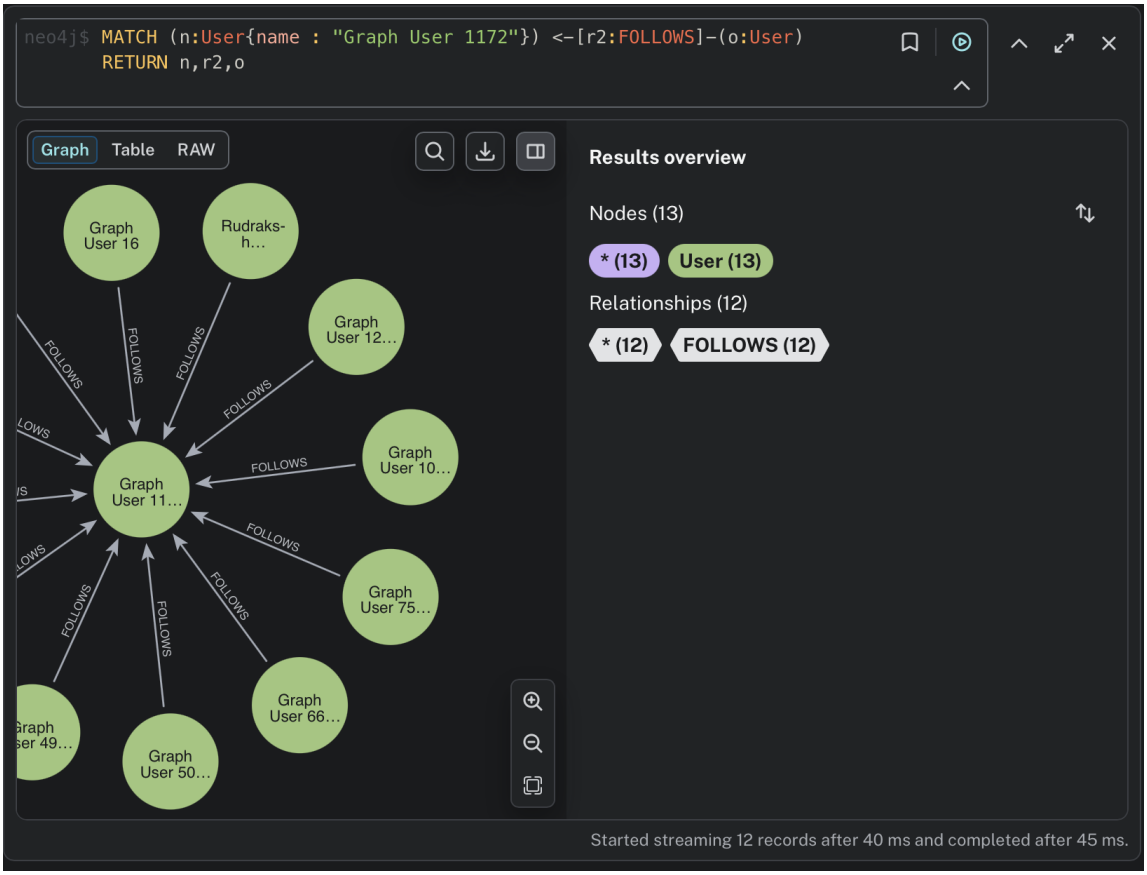
Constraints and Indexes

Unique constraint on :User(username) (created/ensured at startup in UserCRUD.init).

Denormalization & Rationale

followersCount and followingCount are stored on :User and incremented/decremented within the same Cypher follow/unfollow operations to avoid expensive MATCH count operations at query time and to keep reads fast.


neo4j$
neo4j$ MATCH (n:User{name : "Rudraksh Naik"}) return n
Graph
Table
RAW
Rudraks-
h...
Node details
User
Key
Value
<id>
4:6f44f415-1ef9-4a87-a2f0-60
f4af0109cf:1506
bio
"fixing bugs"
email
"rudraksh@gmail.com"
followersCount
0
followingCount
0
name
Integer
draksh Naik"
passwordHash
"$2b$12$FsgDCFAoQHh3hR8
khpmnyOZqHyI1XauxuVaNWc
5VYCpx.tysqk3bm"
userId
"3cbd74d5-a488-45bc-bd6a-
6ea61c1696b4"
username
"rud"
Started streaming 1 record after 37 ms and completed after 40 ms.


neo4j$ MATCH (n:User{name : "Graph User 1172"}) <-[r2:FOLLOWS]-(o:User)
RETURN n,r2,o
Graph
Table
RAW
Graph
User 16
Rudraks-
h...
Graph
User 12...
Graph
User 11...
Graph
User 10...
Graph
User 75...
Graph
User 66...
Graph
User 50...
FOLLOWS
Results overview
Nodes (13)
* (13)
User (13)
Relationships (12)
* (12)
FOLLOWS (12)
Started streaming 12 records after 40 ms and completed after 45 ms.

2. Dataset Information

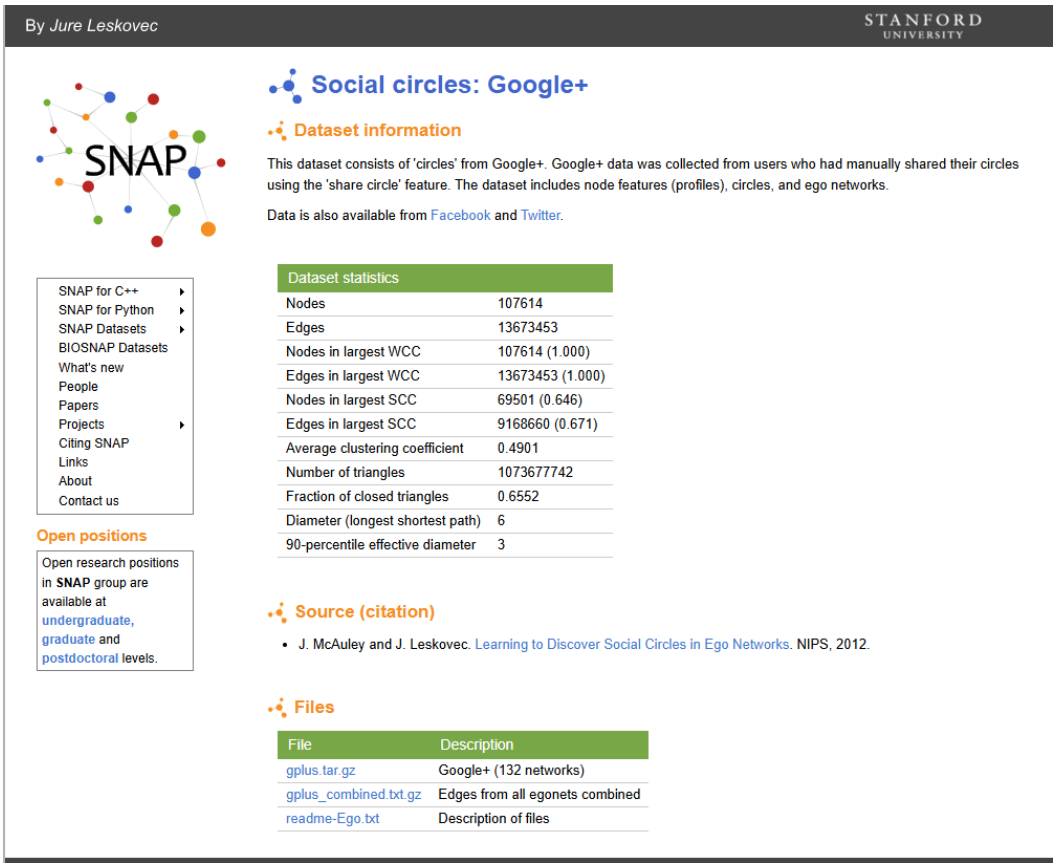
Social circles: Google+ https://snap.stanford.edu/data/ego-Gplus.html

Dataset Description:

The ego-Gplus dataset is a large-scale "ego-network" collection derived from the (now defunct) Google+ social network. It includes roughly **107,614 users (nodes)** and **13,673,453 directed links (edges)**, where a directed edge A → B indicates that user A listed B in one of their Google+ "circles." For many users, the dataset provides not only the network structure but also anonymized profile-feature data and the "circle" labels as defined by the users themselves (i.e. how they grouped their contacts). As a result, ego-Gplus captures both social connectivity patterns and personal grouping behavior — making it useful for research on social network structure, community detection, friend recommendation, link prediction, and how people organize their relationships.

Because this dataset stems only from users who opted to publicly share their circles, it represents **a biased, partial snapshot** of Google+ rather than a full social graph; the "circles" reflect individual users' subjective grouping rather than objective communities.

In summary: ego-Gplus is among the larger publicly available online-social ego networks, combining structural, relational, and (anonymized) profile information along with user-defined communities — a rich resource for studying social network patterns, community structure, and circle detection.



By *Jure Leskovec* STANFORD UNIVERSITY

SNAP

- SNAP for C++
- SNAP for Python
- SNAP Datasets
- BIOSNAP Datasets
- What's new
- People
- Papers
- Projects
- Citing SNAP
- Links
- About
- Contact us

Open positions

Open research positions in **SNAP** group are available at undergraduate, graduate and postdoctoral levels.

Social circles: Google+

Dataset information

This dataset consists of 'circles' from Google+. Google+ data was collected from users who had manually shared their circles using the 'share circle' feature. The dataset includes node features (profiles), circles, and ego networks.

Data is also available from Facebook and Twitter.

Dataset statistics	
Nodes	107614
Edges	13673453
Nodes in largest WCC	107614 (1.000)
Edges in largest WCC	13673453 (1.000)
Nodes in largest SCC	69501 (0.646)
Edges in largest SCC	9168660 (0.671)
Average clustering coefficient	0.4901
Number of triangles	1073677742
Fraction of closed triangles	0.6552
Diameter (longest shortest path)	6
90-percentile effective diameter	3

Source (citation)

- J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

Files

File	Description
gplus.tar.gz	Google+ (132 networks)
gplus_combined.txt.gz	Edges from all egonets combined
readme-Ego.txt	Description of files

Data Processing Steps:

- **Read the raw edge file (gplus_combined.txt)** line by line, where each line represents a directed edge ID_A ID_B (A → B).
- **Collect the first 1,500 unique user IDs** encountered in the file:
  - Maintain a unique_ids set.
  - Keep an ordered_ids list to preserve the order in which IDs were discovered.
  - Stop reading once the target count (1,500) is reached.
- **Store edges only if both nodes are within the collected ID set**, creating a clean subgraph (processed_edges).
- **Check minimum graph size requirements** (at least 1,000 users and 5,000 edges).

User Data Augmentation

- **Generate a single bcrypt hash** for a default password (password123) to use for all users.
- For each collected user ID, create synthetic user properties:
  - username → user_<raw_id>
  - name → Graph User <index>
  - email → <raw_id>@gplus.com
  - bio → a default profile bio
  - passwordHash → the bcrypt hash
- Store all user records in a list (user_data) and create an ID → username mapping.

CSV Export

- **Write users.csv** containing:
  - userId, username, name, email, passwordHash, bio
- **Translate graph edges into application-friendly relationships** using the ID→username mapping.
- **Write connections.csv** containing:
  - follower_username, followee_username
  - Each row represents a directed follow relationship.

End Result

- A clean, filtered 1,500-node subgraph of Google+.
- Synthetic user profiles ready for import into a database.
- A separate connections list representing all valid directed relationships between these users.

Cypher Statements Used:

```
UNWIND $rows AS row
MERGE (u:User {username: row.username})
SET u.userId = row.userId,
    u.name = row.name,
    u.email = row.email,
    u.passwordHash = row.passwordHash,
    u.bio = row.bio,
    u.followersCount = 0,
    u.followingCount = 0
"""
```

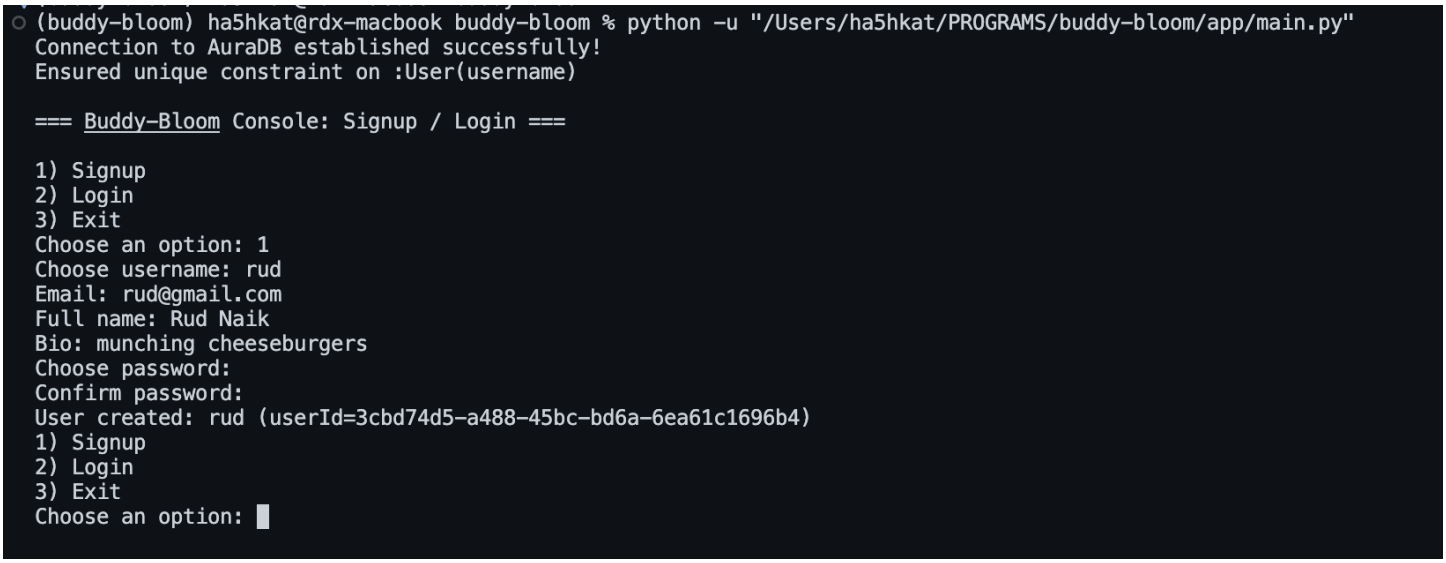
```
for i in range(0, len(connections), batch_size):
    batch = connections[i : i + batch_size]
    query = """
    UNWIND $rows AS row
    MATCH (follower:User {username: row.follower_username})
    MATCH (followee:User {username: row.followee_username})
    MERGE (follower)-[r:FOLLOWS]->(followee)
    ON CREATE SET
        r.since = datetime(),
        follower.followingCount = coalesce(follower.followingCount, 0) + 1,
        followee.followersCount = coalesce(followee.followersCount, 0) + 1
    """
```

3. Use Case Evidence

UC-1

User Registration: A new user can sign up by providing basic details (name, email, username, password). The system stores user data in Neo4j as nodes.

Screenshot:



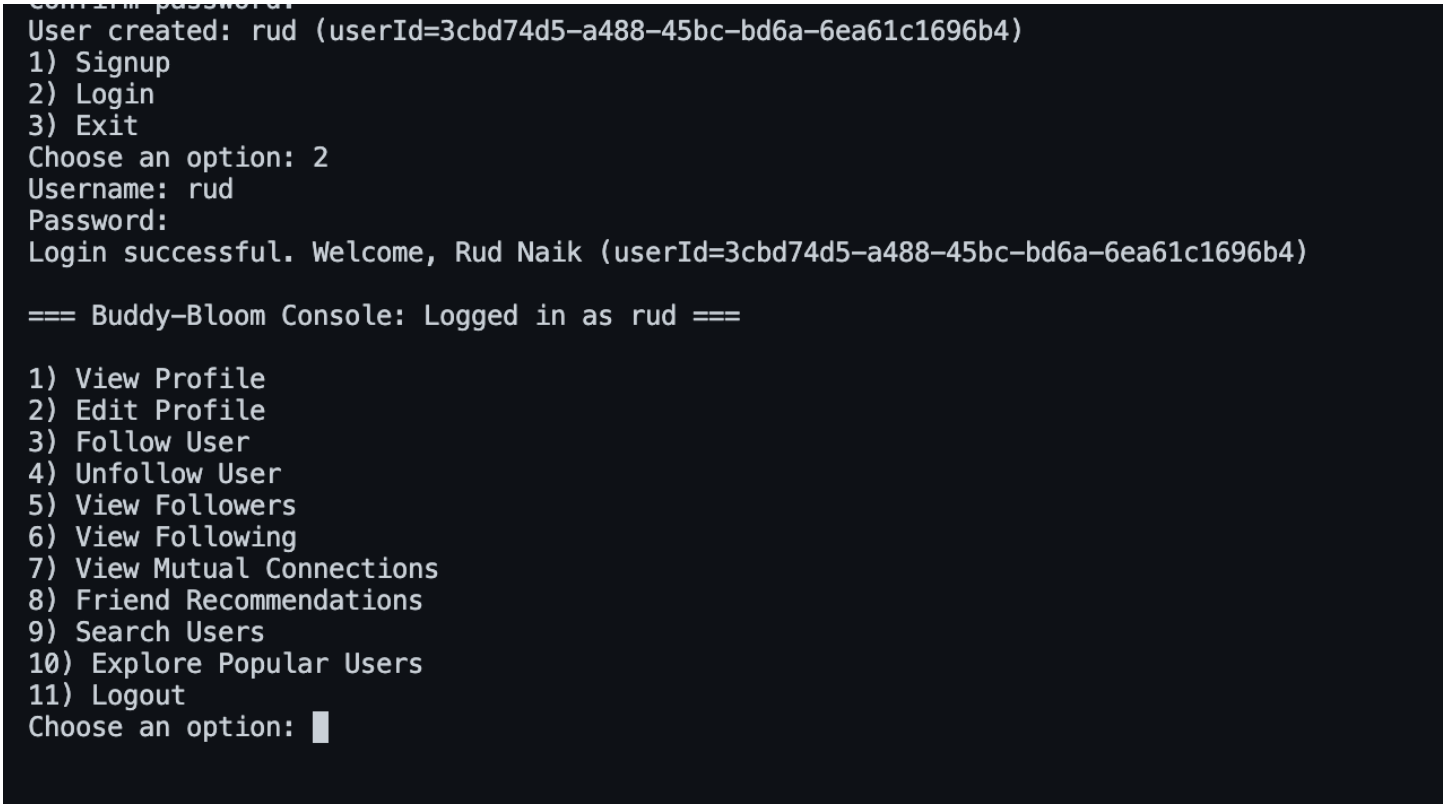
Cypher Query:

```
"""
MERGE (u:User {username: $username})
ON CREATE SET
    u.userId = $userId,
    u.passwordHash = $passwordHash,
    u.name = $name,
    u.email = $email,
    u.bio = $bio,
    u.followersCount = 0,
    u.followingCount = 0
RETURN u.userId AS userId, u.username AS username, u.passwordHash AS passwordHash,
       u.name AS name, u.email AS email, u.bio AS bio,
       u.followersCount AS followersCount, u.followingCount AS followingCount
"""
```

UC-2

Description:
User Login: A registered user can log in using their username and password. The system authenticates the credentials and grants access.

Screenshot:



Cypher Query:
The login flow in this project uses a DB lookup by username to fetch the stored password hash, then compares it to the provided password in application code (bcrypt check). The Cypher used to read the user by username is in database.py — get_user_by_username:

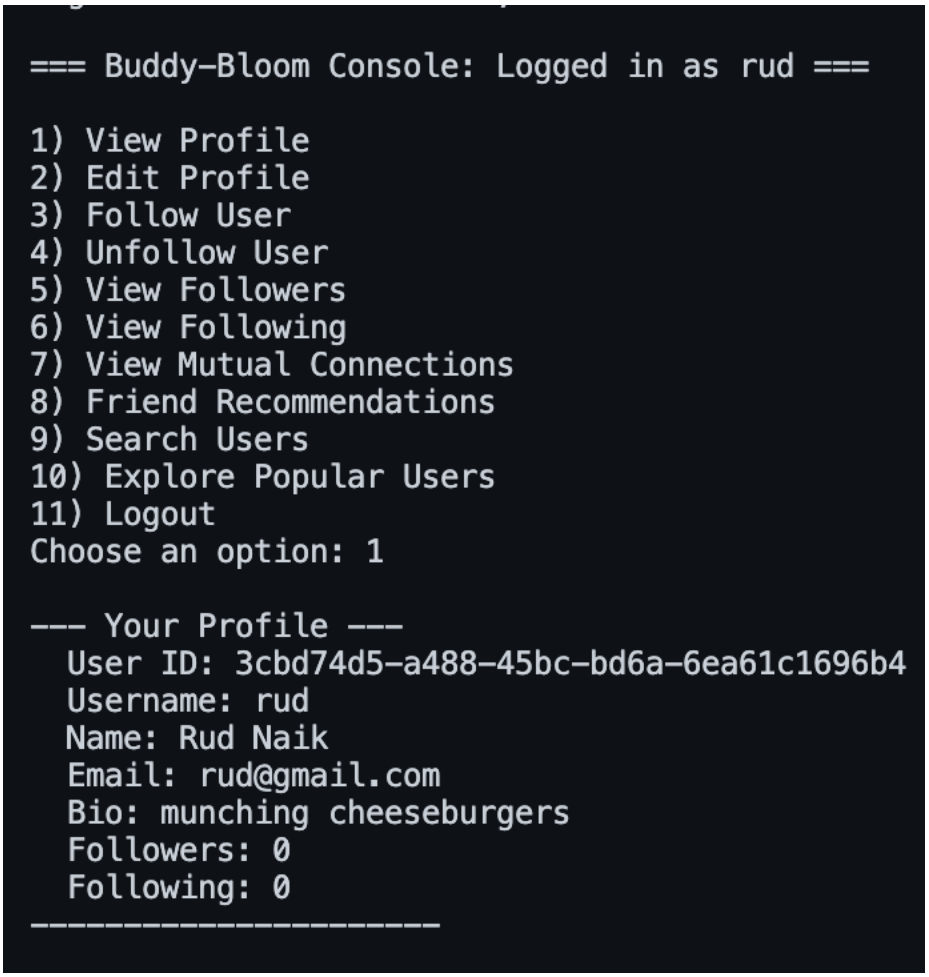
```
// ...existing code...

MATCH (u:User {username: $username})

RETURN u.userId AS userId, u.username AS username, u.passwordHash AS passwordHash,

    u.name AS name, u.email AS email, u.followersCount AS followersCount,

    u.followingCount AS followingCount, u.bio AS bio

// ...existing code...
```

UC-3

Description:

View Profile: A user can view their own profile and update basic information.

Screenshot:



Cypher Query:

```
"MATCH (u:User {userId: $userId}) RETURN u.userId AS userId,"
" u.username AS username, u.passwordHash AS passwordHash, u.name AS name, u.email AS email,"
" u.followersCount AS followersCount, u.followingCount AS followingCount, u.bio AS bio",
userId=user_id,
```

```
"MATCH (u:User {username: $username}) RETURN u.userId AS userId,"
" u.username AS username, u.passwordHash AS passwordHash, u.name AS name,"
" u.email AS email, u.followersCount AS followersCount, u.followingCount AS followingCount, u.bio AS bio",
username=username,
```

UC-4

Edit Profile: A user can update their name, bio, and other details.

Screenshot:

```
=== Buddy-Bloom Console: Logged in as rud ===

1) View Profile
2) Edit Profile
3) Follow User
4) Unfollow User
5) View Followers
6) View Following
7) View Mutual Connections
8) Friend Recommendations
9) Search Users
10) Explore Popular Users
11) Logout
Choose an option: 2

--- Edit Profile ---
Current Name: Rud Naik
Enter new Name (or press Enter to keep current): Rudraksh Naik
Current Email: rud@gmail.com
Enter new Email (or press Enter to keep current): rudraksh@gmail.com
Current Bio: munching cheeseburgers
Enter new Bio (or press Enter to keep current): fixing bugs

--- Change Password ---
Enter new Password (or press Enter to keep current):
Profile updated successfully!
```

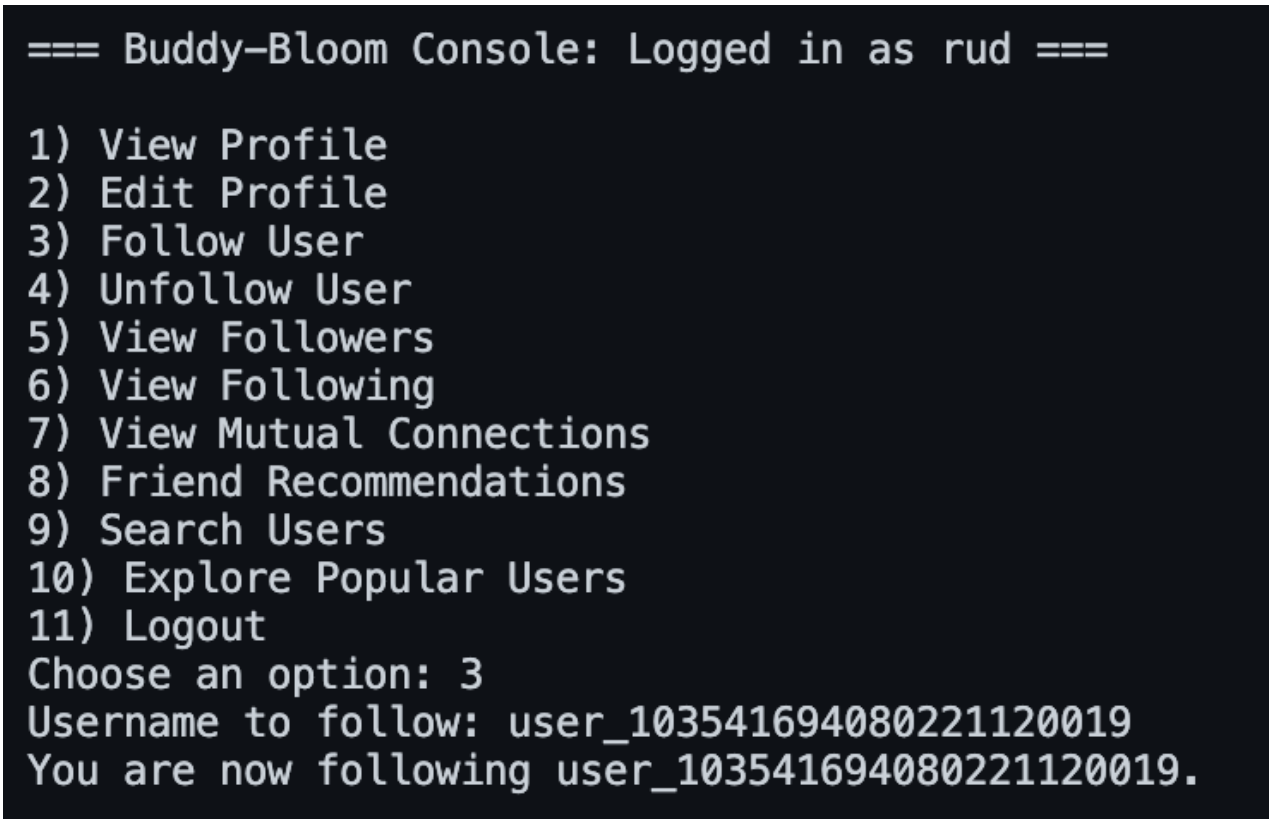
Cypher Query:

```
MATCH (u:User {{userId: $userId}})
SET {set_clause_str}
RETURN u.userId AS userId, u.username AS username, u.passwordHash AS passwordHash,
       u.name AS name, u.email AS email, u.bio AS bio,
       u.followersCount AS followersCount, u.followingCount AS followingCount
"""
```

UC-5

Follow Another User - A user can follow another user, creating a "FOLLOWS"

Screenshot:



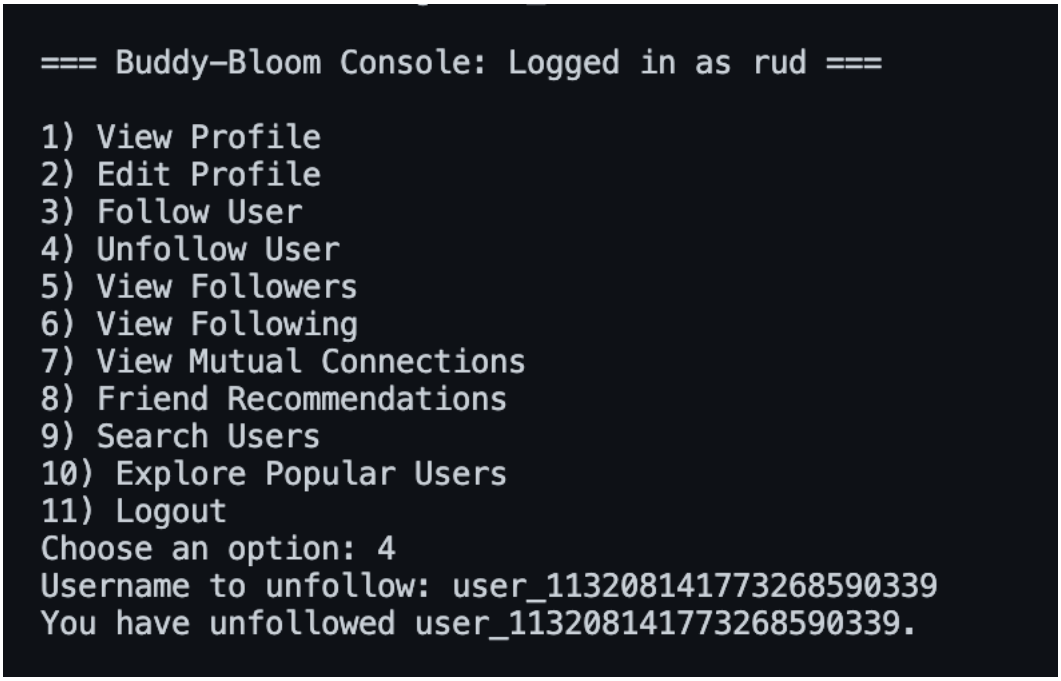
Cypher Query:

```
MATCH (follower:User {username: $follower_username})
MATCH (followee:User {username: $followee_username})
MERGE (follower)-[f:FOLLOWS]->(followee)
ON CREATE SET f.since = datetime()
ON CREATE SET
    follower.followingCount = coalesce(follower.followingCount, 0) + 1,
    followee.followersCount = coalesce(followee.followersCount, 0) + 1
RETURN f
"""
```

UC-6

: Unfollow a User - A user can unfollow another user, removing the "FOLLOWS" relationship.

Screenshot:



Cypher Query:

```
MATCH (follower:User {username: $follower_username})-[f:FOLLOWS]->(followee:User {username: $followee_username})
WITH follower, followee, f
DELETE f
SET
    follower.followingCount = coalesce(follower.followingCount, 1) - 1,
    followee.followersCount = coalesce(followee.followersCount, 1) - 1
RETURN follower, followee
```

UC-7

View Friends/Connections - A user can see a list of people they are following and who follow them.

Screenshot:

```
=== Buddy-Bloom Console: Logged in as rud ===

1) View Profile
2) Edit Profile
3) Follow User
4) Unfollow User
5) View Followers
6) View Following
7) View Mutual Connections
8) Friend Recommendations
9) Search Users
10) Explore Popular Users
11) Logout
Choose an option: 6

--- Following ---
 - user_103541694080221120019 (Graph User 1172) — followers:12 following:6
------------------
```

```
MATCH (u:User {username: $username})-[:FOLLOWS]->(followee:User)
RETURN followee.userId AS userId, followee.username AS username, followee.name AS name, followee.email AS email,
       followee.followersCount AS followersCount, followee.followingCount AS followingCount
ORDER BY followee.username SKIP $skip LIMIT $limit
```

```
=== Buddy-Bloom Console: Logged in as rud ===

1) View Profile
2) Edit Profile
3) Follow User
4) Unfollow User
5) View Followers
6) View Following
7) View Mutual Connections
8) Friend Recommendations
9) Search Users
10) Explore Popular Users
11) Logout
Choose an option: 5

--- Followers ---
 - user_103541694080221120019 (Graph User 1172) — followers:12 following:7
------------------
```

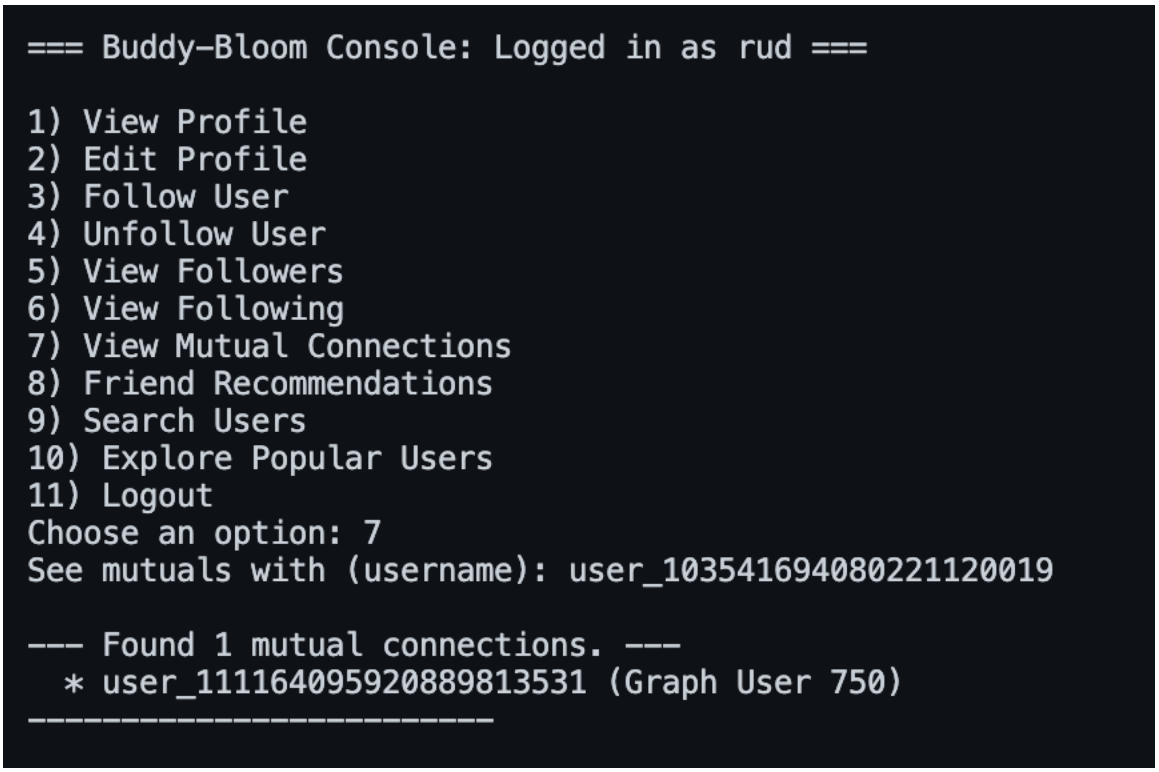
Cypher Query:

```
MATCH (f:User)-[:FOLLOWS]->(u:User {username: $username})
RETURN f.userId AS userId, f.username AS username, f.name AS name, f.email AS email,
       f.followersCount AS followersCount, f.followingCount AS followingCount
ORDER BY f.username SKIP $skip LIMIT $limit
```

UC-8

Mutual Connections - A user can see mutual friends (users followed by both parties).

Screenshot:



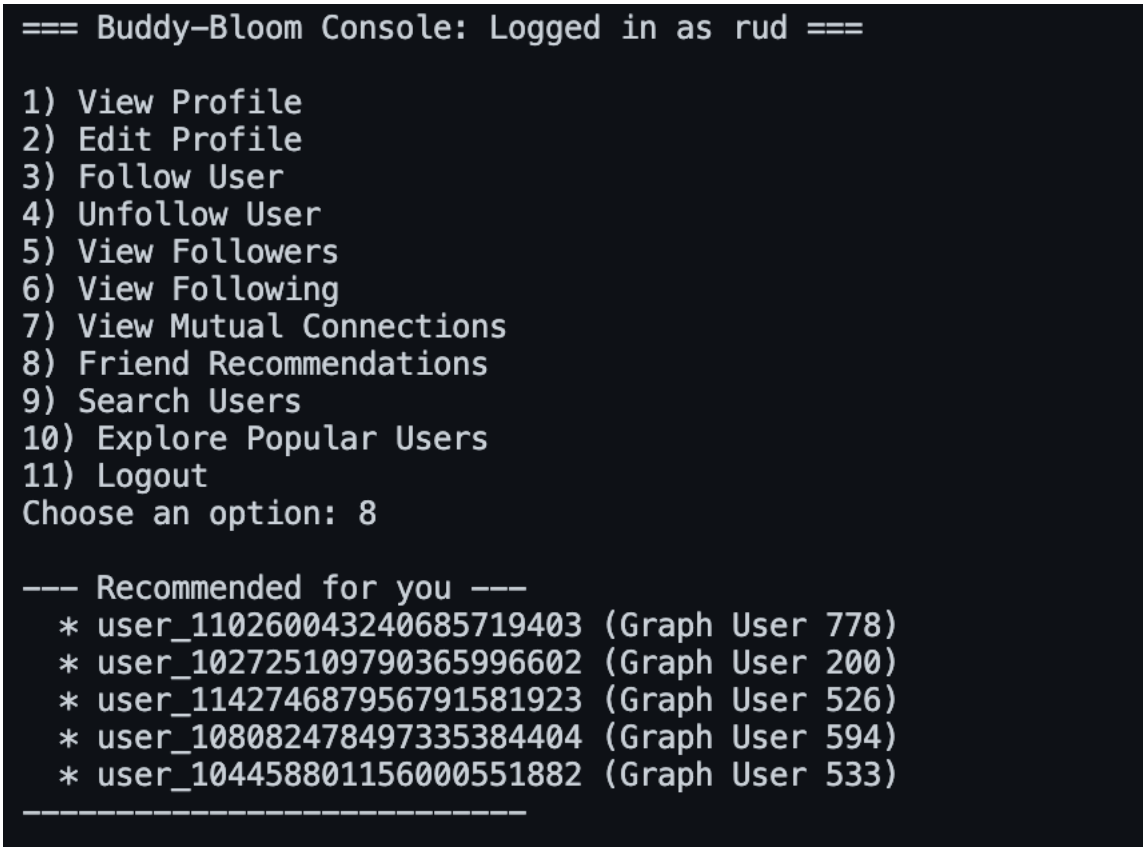
Cypher Query:

```
MATCH (u1:User {username: $u1})-[:FOLLOWS]->(mutual:User)<-[:FOLLOWS]-(u2:User {username: $u2})
RETURN mutual.userId AS userId, mutual.username AS username, mutual.name AS name,
       mutual.email AS email, mutual.bio AS bio,
       mutual.followersCount AS followersCount, mutual.followingCount AS followingCount
```

UC-9

Friend Recommendations - The system suggests new people to follow based on common connections using graph traversal queries.

Screenshot:



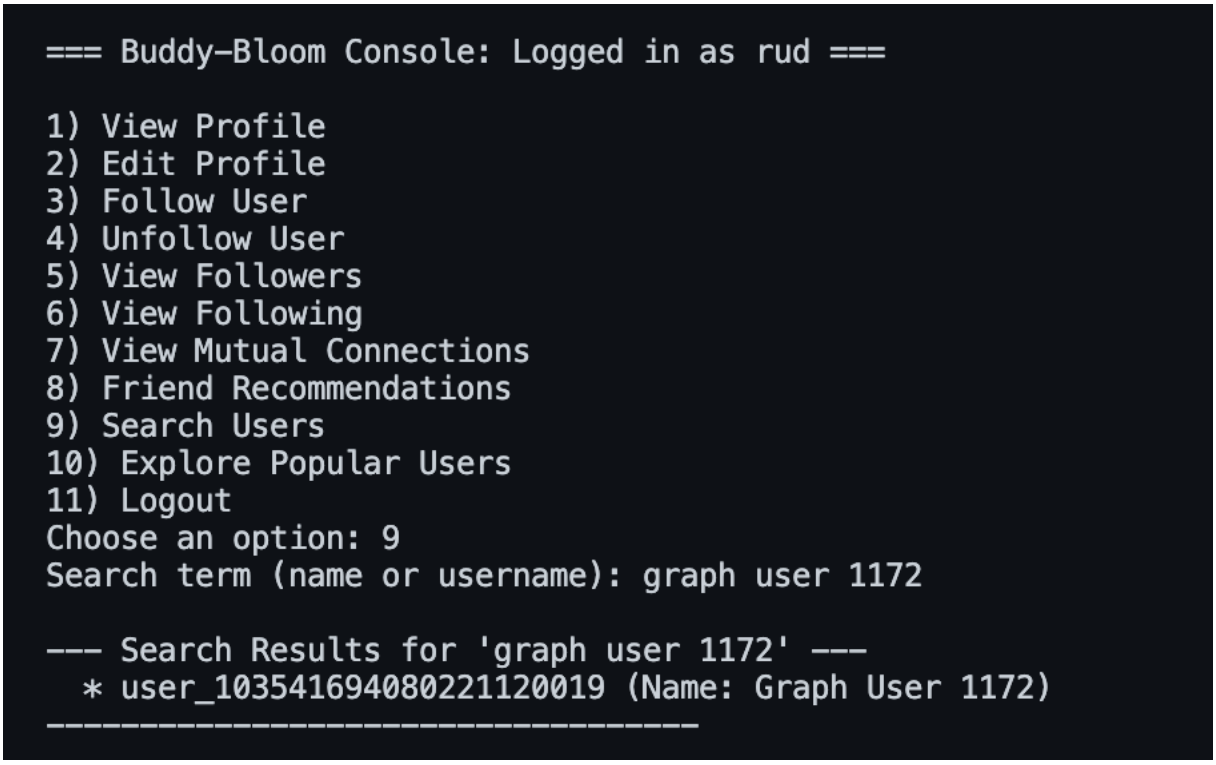
Cypher Query:

```
MATCH (u:User {username: $username})-[:FOLLOWS]->(friend)-[:FOLLOWS]->(fof:User)
WHERE NOT (u)-[:FOLLOWS]->(fof) AND u <> fof
RETURN fof.userId AS userId, fof.username AS username, fof.name AS name,
       fof.email AS email, fof.bio AS bio,
       fof.followersCount AS followersCount, fof.followingCount AS followingCount,
       count(friend) as strength
ORDER BY strength DESC
LIMIT 5
```

UC-10

Search Users - A user can search for other users by name or username. The system returns a list of matching users.

Screenshot:



Cypher Query:

```
MATCH (u:User)
WHERE toLower(u.username) CONTAINS toLower($term)
   OR toLower(u.name) CONTAINS toLower($term)
RETURN u.userId AS userId, u.username AS username, u.name AS name,
       u.email AS email, u.bio AS bio,
       u.followersCount AS followersCount, u.followingCount AS followingCount
LIMIT 20
"""
```

UC-11

Explore Popular Users - The system displays the most-followed users.

Screenshot:

```
=== Buddy-Bloom Console: Logged in as rud ===

1) View Profile
2) Edit Profile
3) Follow User
4) Unfollow User
5) View Followers
6) View Following
7) View Mutual Connections
8) Friend Recommendations
9) Search Users
10) Explore Popular Users
11) Logout
Choose an option: 10

--- Top 10 Popular Users ---
  #1 user_106189723444098348646 (37 followers)
  #2 user_112063946124358686266 (35 followers)
  #3 user_111091089527727420853 (34 followers)
  #4 user_109813896768294978296 (32 followers)
  #5 user_113116318008017777871 (29 followers)
  #6 user_107117483540235115863 (29 followers)
  #7 user_107753428759636856492 (29 followers)
  #8 user_118207880179234484610 (28 followers)
  #9 user_105841645449323318101 (28 followers)
  #10 user_100535338638690515335 (28 followers)
----------------------------
```

Cypher Query:

```
MATCH (u:User)
RETURN u.userId AS userId, u.username AS username, u.name AS name,
       u.email AS email, u.bio AS bio,
       u.followersCount AS followersCount, u.followingCount AS followingCount
ORDER BY u.followersCount DESC
LIMIT 10
```