Comparing graph-oriented queries in Cypher and GraphQL languages

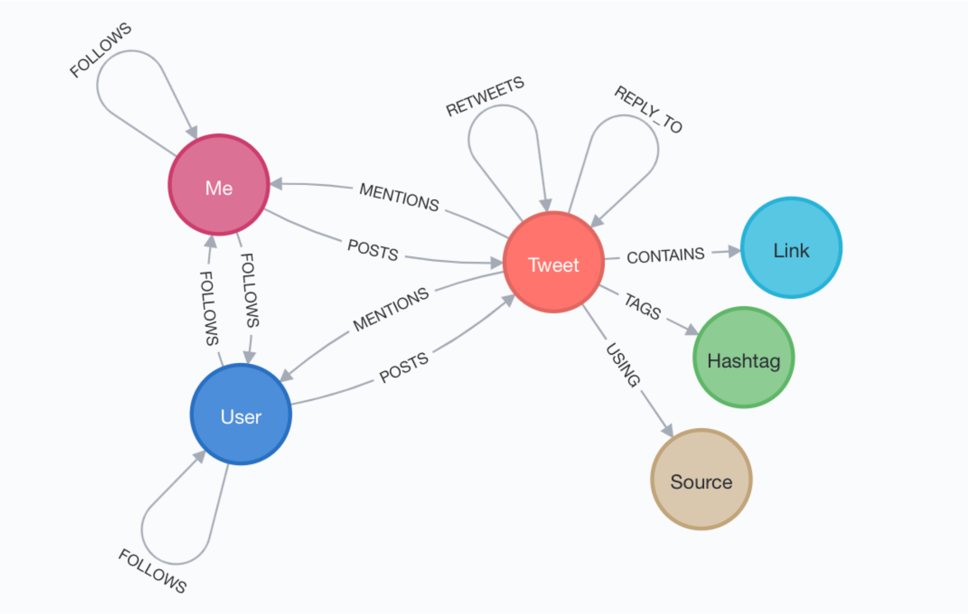
Aim: The aim is to explore and compare the strengths and limitations of Cypher and GraphQL using two Neo4j graph databases. By conducting this comparison, this report explores capabilities of each language and how they handle complex graph queries. This report also identifies limitations of expressing queries in GraphQL compared to Cypher.

Methodology: The methodology is writing relatively complex queries in Cypher, the native query language for Neo4j, and then attempting to express the same queries in GraphQL. Throughout the process, the report records the challenges faced when translating Cypher queries to GraphQL.
What I used: *Neo4j Desktop* and *Node.js GraphQL API* application using @neo4j/graphql.

Explore a social network with Neo4j's Twitter data

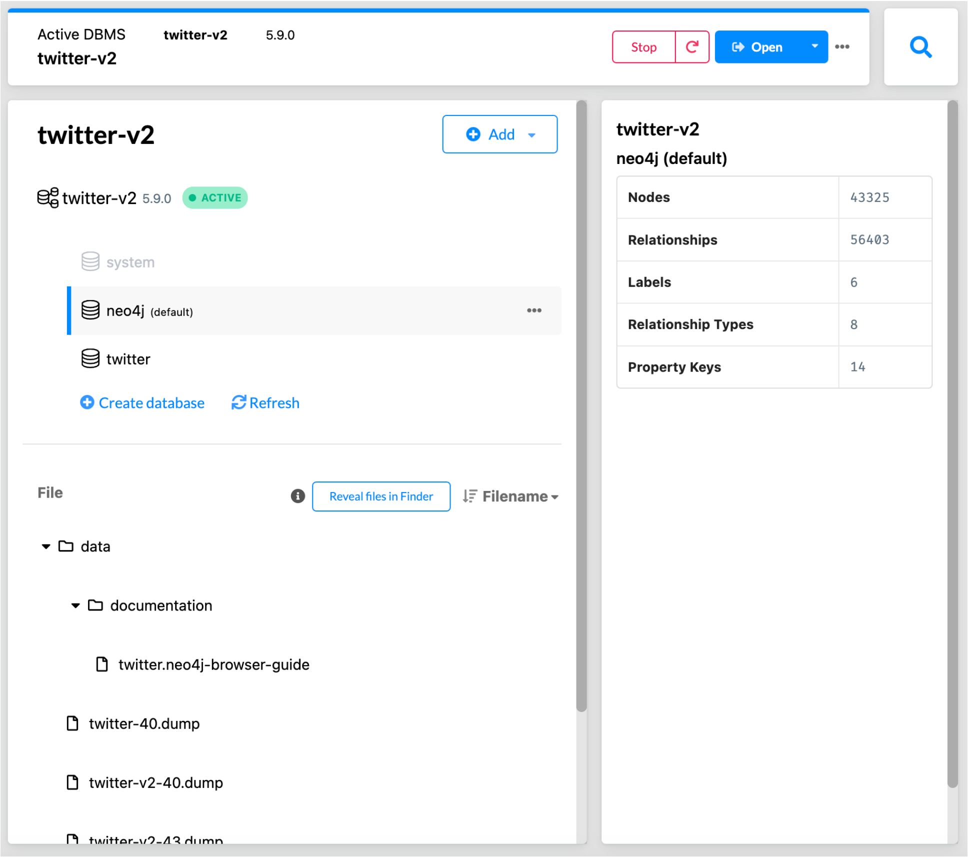
Resource used:
https://github.com/neo4j-graph-examples/twitter-v2/blob/main/documentation/twitter.adoc

Relationships graph:



Questions to explore: Who are your most influential followers? What tags you use frequently? How many people you follow also follow you back? People tweeting about you, but you don't follow? Links from intresting retweets. Other people tweeting with some of your top hashtags.

See the number of nodes and relationships in twitter database:



twitter-v2

neo4j (default)

Nodes	43325
Relationships	56403
Labels	6
Relationship Types	8
Property Keys	14

Queries explored:

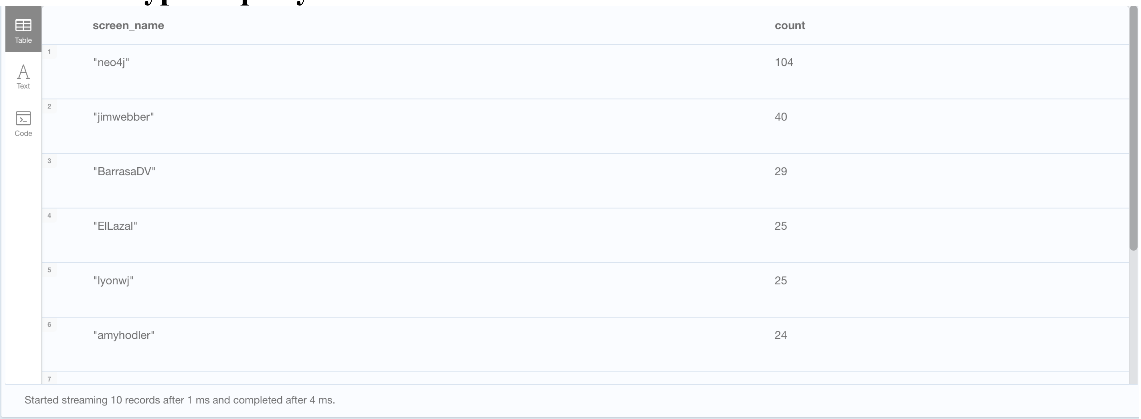
1. Your mentions

Description: Find the top 10 users being mentioned most times by the user “Me”.

Cypher query:

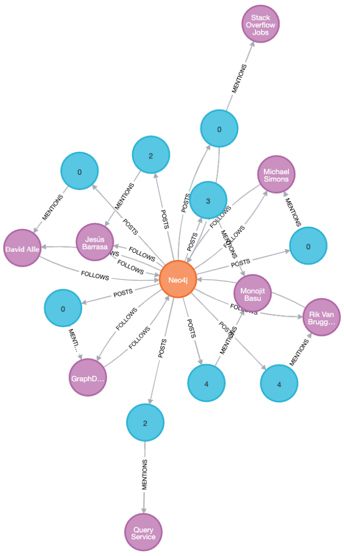
```
MATCH
  (u:User:Me)-[:POSTS]->(t:Tweet)-[:MENTIONS]->(m:User)
RETURN
  m.screen_name AS screen_name, COUNT(m.screen_name) AS count
ORDER BY
  count DESC
LIMIT 10
```

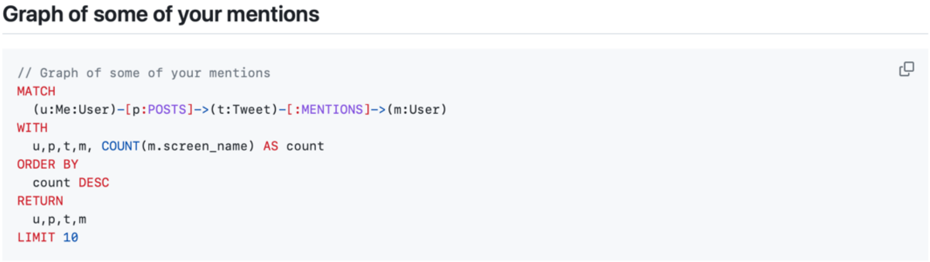
Result of cypher query:



	screen_name	count
1	"neo4j"	104
2	"jimwebber"	40
3	"BarrasaDV"	29
4	"ElLazal"	25
5	"lyonwj"	25
6	"amyhodler"	24
7		

Started streaming 10 records after 1 ms and completed after 4 ms.

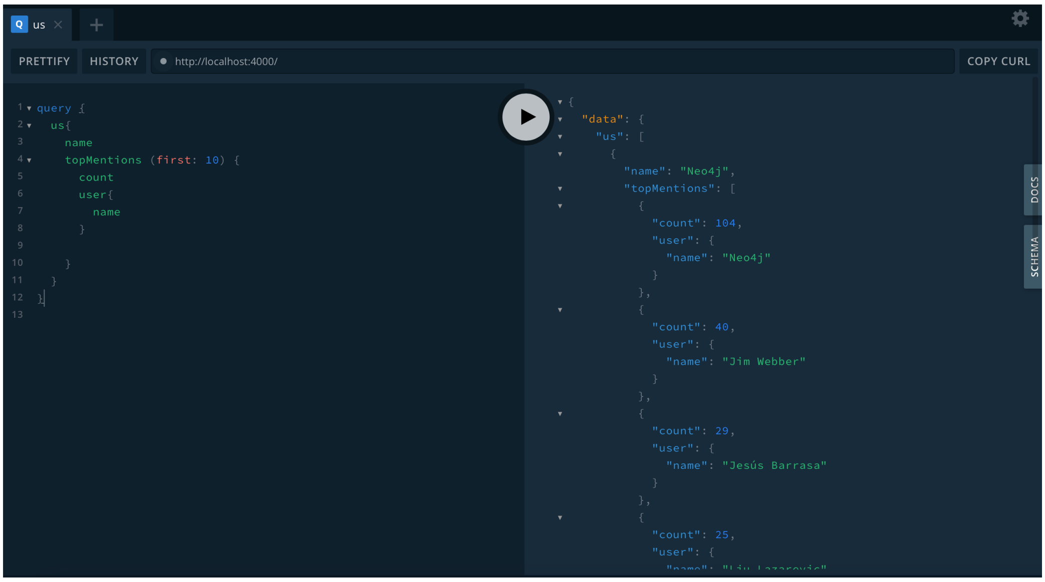




Graph of some of your mentions

```
// Graph of some of your mentions
MATCH
  (u:Me:User)-[p:POSTS]->(t:Tweet)-[:MENTIONS]->(m:User)
WITH
  u,p,t,m, COUNT(m.screen_name) AS count
ORDER BY
  count DESC
RETURN
  u,p,t,m
LIMIT 10
```

GraphQL Query and result:



```
query {
  us{
    name
    topMentions (first: 10) {
      count
      user{
        name
      }

    }
  }
}
```

```
{
  "data": {
    "us": [
      {
        "name": "Neo4j",
        "topMentions": [
          {
            "count": 104,
            "user": {
              "name": "Neo4j"
            }
          },
          {
            "count": 40,
            "user": {
              "name": "Jim Webber"
            }
          },
          {
            "count": 29,
            "user": {
              "name": "Jesús Barrasa"
            }
          },
          {
            "count": 25,
            "user": {
              "name": "Lju Lazarevic"
```

Although both languages can find the list of top 10 users being mentioned most times, the Cypher graph is more illustrative.

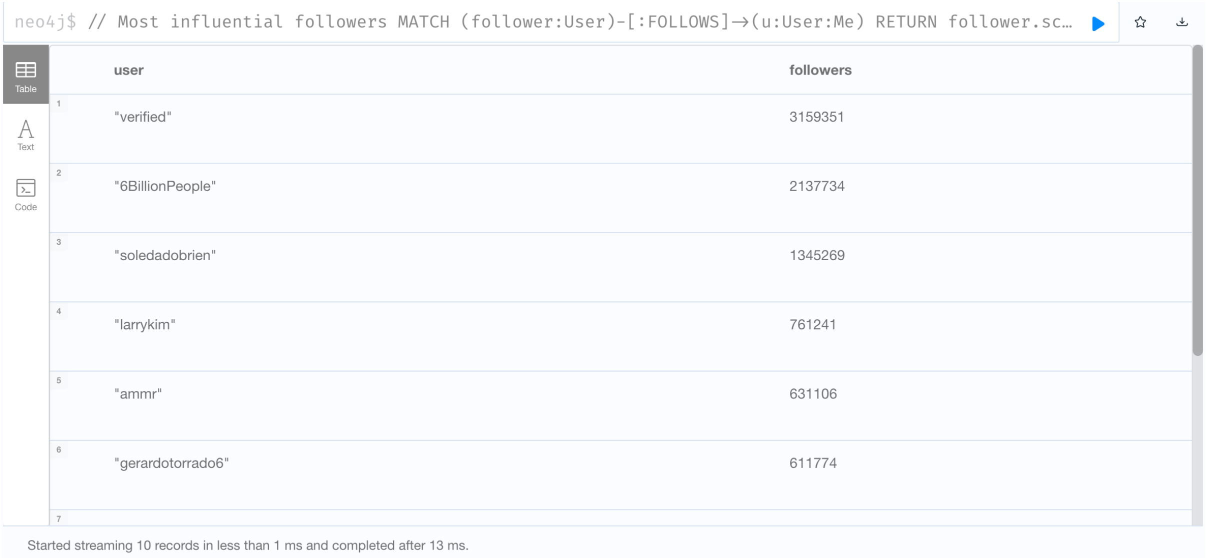
2. Most influential followers

Description: Find the top 10 most influential followers of “Me” by descending followers count.

Cypher query:

```
MATCH
  (follower:User)-[:FOLLOWS]->(u:User:Me)
RETURN
  follower.screen_name AS user, follower.followers AS followers
ORDER BY
  followers DESC
LIMIT 10
```

Result of cypher query:



```
neo4j$ // Most influential followers MATCH (follower:User)-[:FOLLOWS]→(u:User:Me) RETURN follower.sc…
```

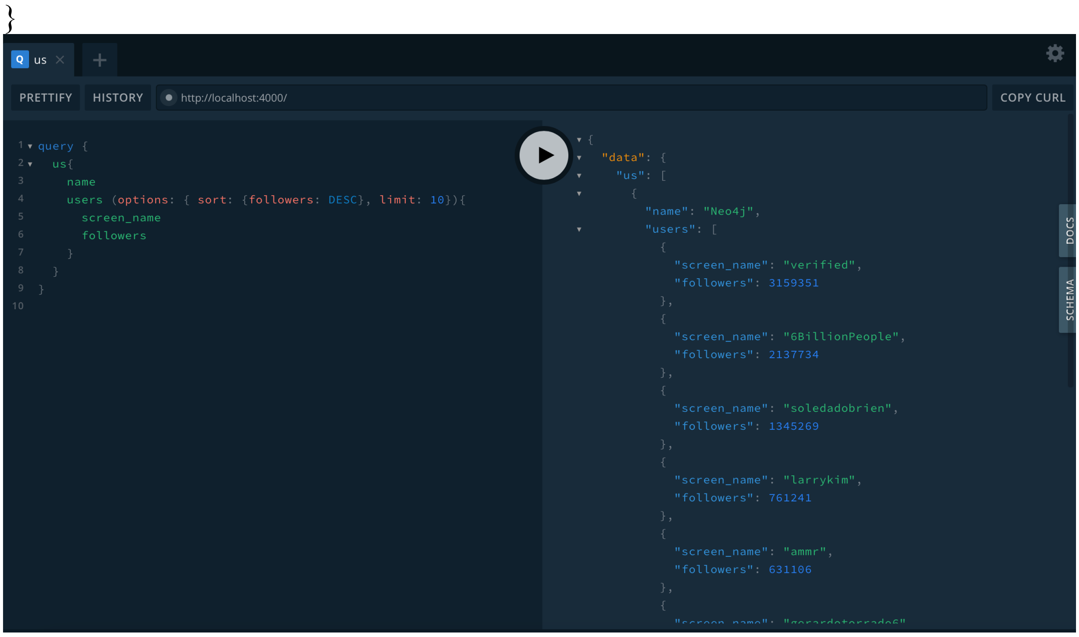
	user	followers
1	"verified"	3159351
2	"6BillionPeople"	2137734
3	"soledadobrien"	1345269
4	"larrykim"	761241
5	"ammr"	631106
6	"gerardotorrado6"	611774
7		

Started streaming 10 records in less than 1 ms and completed after 13 ms.

GraphQL Query and result:

```
query {
        us{
  name
  users (options: { sort: {followers: DESC}, limit: 10}){
    screen_name
    followers
  }
 }
```

}



```
query {
  us{
    name
    users (options: { sort: {followers: DESC}, limit: 10}){
      screen_name
      followers
    }
  }
}
```

```
{
  "data": {
    "us": [
      {
        "name": "Neo4j",
        "users": [
          {
            "screen_name": "verified",
            "followers": 3159351
          },
          {
            "screen_name": "6BillionPeople",
            "followers": 2137734
          },
          {
            "screen_name": "soledadobrien",
            "followers": 1345269
          },
          {
            "screen_name": "larrykim",
            "followers": 761241
          },
          {
            "screen_name": "ammr",
            "followers": 631106
          },
          {
            "screen_name": "gerardotorrado6"
```

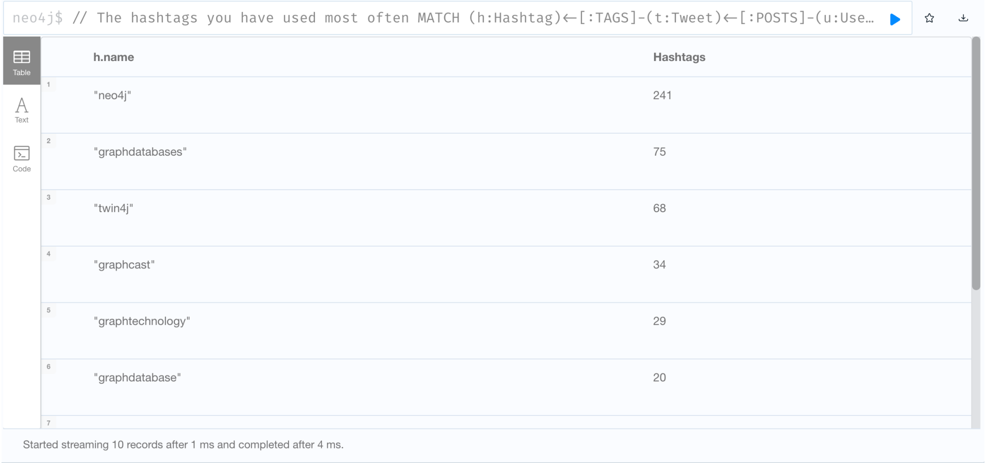
3. Most Tagged

Description: Find the 10 tags being used most frequently by the user "Me".

Cypher query:

```
MATCH
  (h:Hashtag)<-[:TAGS]-(t:Tweet)<-[:POSTS]-(u:User:Me)
WITH
  h, COUNT(h) AS Hashtags
ORDER BY
  Hashtags DESC
LIMIT 10
RETURN
  h.name, Hashtags
```

Result of cypher query:



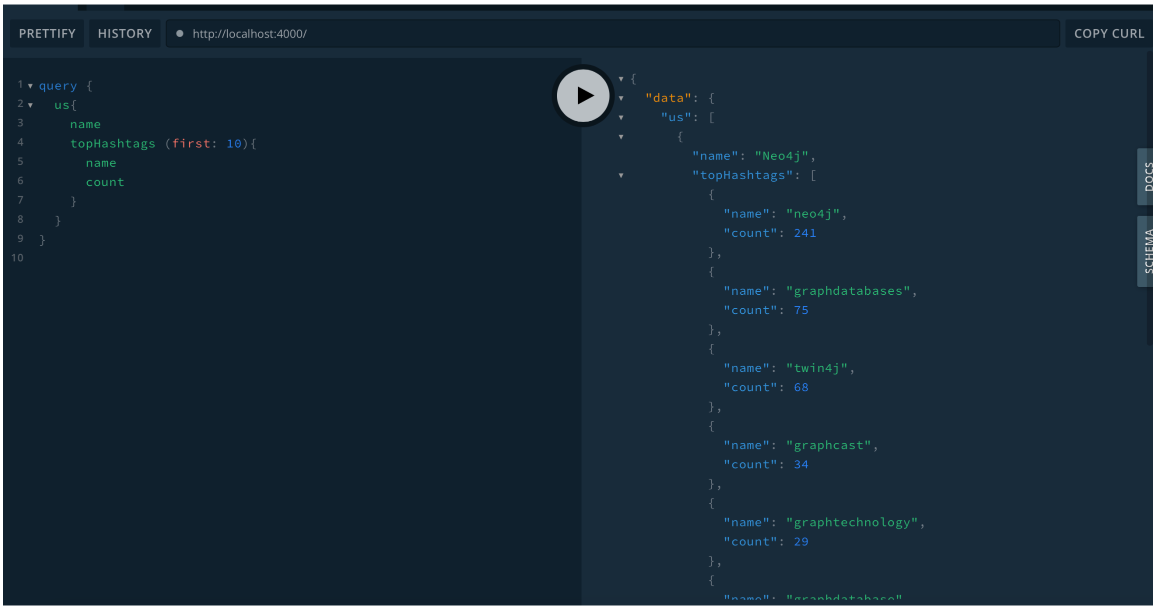
```
neo4j$ // The hashtags you have used most often MATCH (h:Hashtag)←[:TAGS]-(t:Tweet)←[:POSTS]-(u:Use…
```

	h.name	Hashtags
1	"neo4j"	241
2	"graphdatabases"	75
3	"twin4j"	68
4	"graphcast"	34
5	"graphtechnology"	29
6	"graphdatabase"	20
7		

Started streaming 10 records after 1 ms and completed after 4 ms.

GraphQL Query and result:

```
query {
          us{
    name
    topHashtags (first: 10){
      name
      count
    }
  }
}
```



```
query {
  us{
    name
    topHashtags (first: 10){
      name
      count
    }
  }
}
```

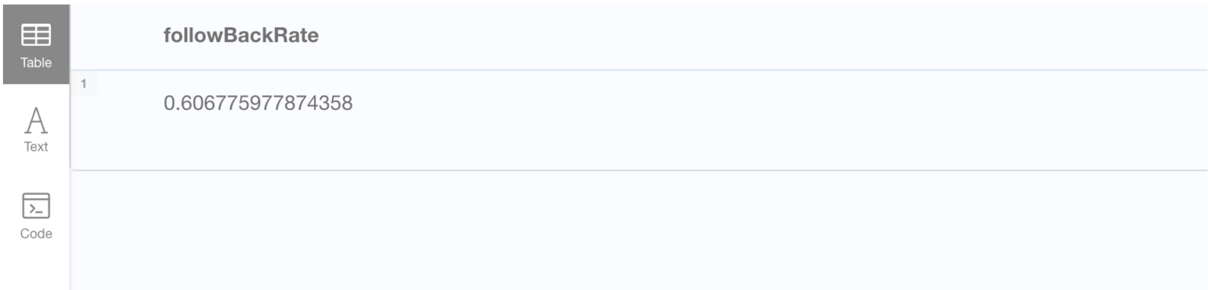
```
{
  "data": {
    "us": [
      {
        "name": "Neo4j",
        "topHashtags": [
          {
            "name": "neo4j",
            "count": 241
          },
          {
            "name": "graphdatabases",
            "count": 75
          },
          {
            "name": "twin4j",
            "count": 68
          },
          {
            "name": "graphcast",
            "count": 34
          },
          {
            "name": "graphtechnology",
            "count": 29
          },
          {
            "name": "graphdatabase"
```

4. Followback rate

Description: The rate of follow back for user “Me”.

Cypher query and result:

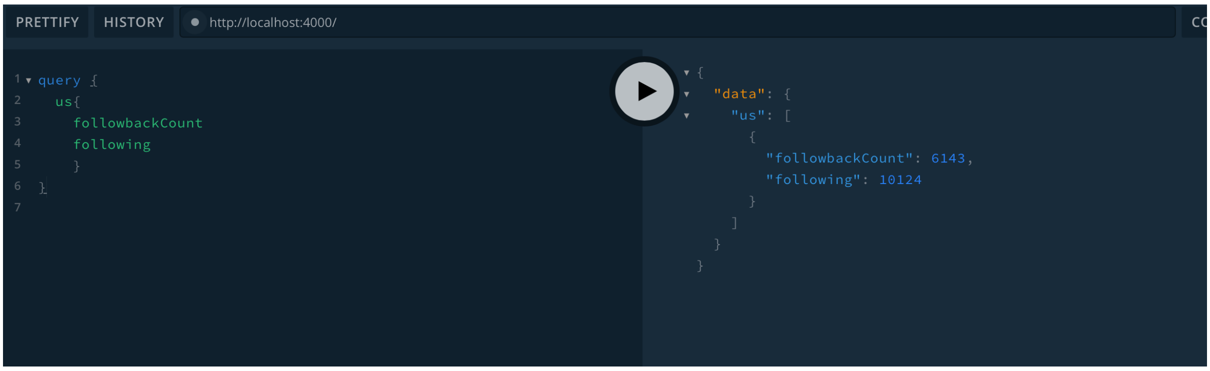
```
MATCH
  (me:User:Me)-[:FOLLOWS]->(f)
WITH
  me, f, count {(f)-[:FOLLOWS]->(me)} as doesFollowBack
RETURN
  SUM(doesFollowBack) / toFloat(COUNT(f))  AS followBackRate
```



	followBackRate
1	0.606775977874358

GraphQL Query and result:

```
query {
        us{
  followbackCount
  following
  }
}
```



```
query {
  us{
    followbackCount
    following
    }
}
```

```
{
  "data": {
    "us": [
      {
        "followbackCount": 6143,
        "following": 10124
      }
    ]
  }
}
```

There's no automatic aggregate function in GraphQL itself. To achieve this query in GraphQL, we can add a field called follow back rate, and in the resolve function calculate the result.

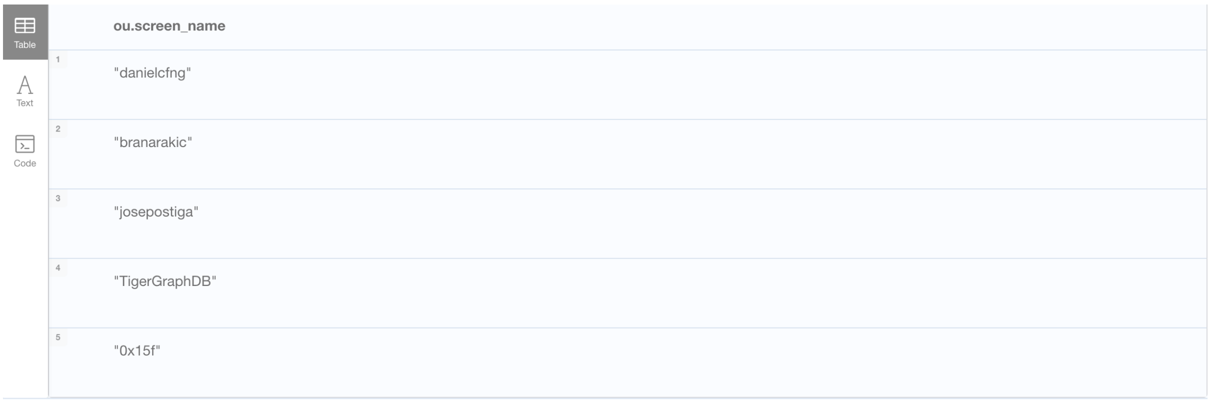
5. Follower Recommendation

Description: Find the names of users followed and tweeted about the user "Me", but the user doesn't follow.

Cypher query and result:

```
// Follower Recommendations - tweeting about you, but you don't follow
MATCH
  (ou:User)-[:POSTS]->(t:Tweet)-[mt:MENTIONS]->(me:User:Me)
WITH
  DISTINCT ou, me, count(t) as count
WHERE
  (ou)-[:FOLLOWS]->(me)
  AND NOT
    (me)-[:FOLLOWS]->(ou)
```

```
RETURN
  ou.screen_name
ORDER BY count DESC
LIMIT 5
```



	ou.screen_name
1	"danielcfng"
2	"branarakic"
3	"josepostiga"
4	"TigerGraphDB"
5	"0x15f"

GraphQL Query and result:

Failed attempt:

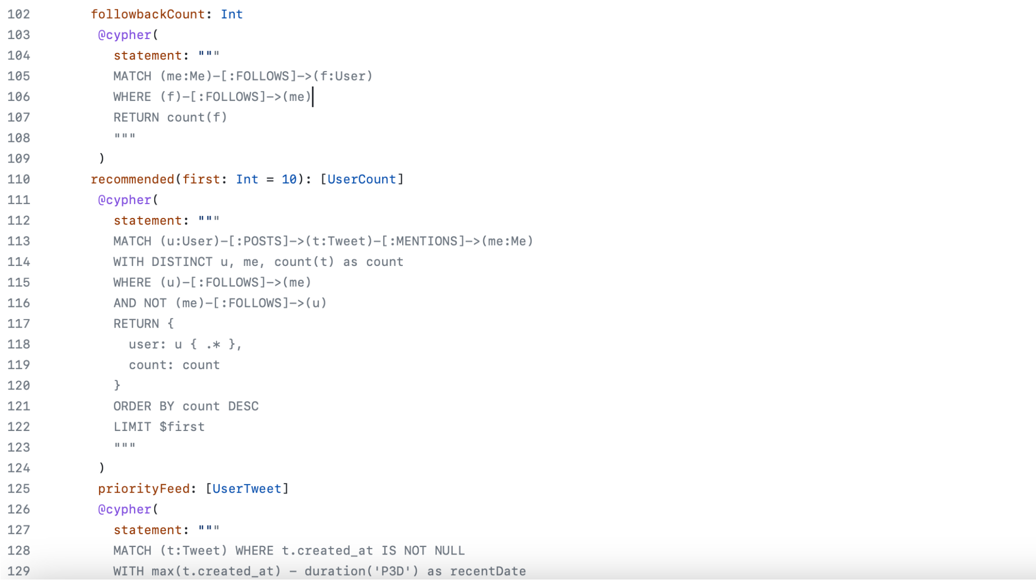
```
query {
  users(where: {following: 1, followers_NOT: 1}) {
    followers
    following
    tweets (where: {mentions: {name: "Neo4j"}}){
            posted_by {
        screen_name
      }
    }
  }
}
```

Successful attempt:

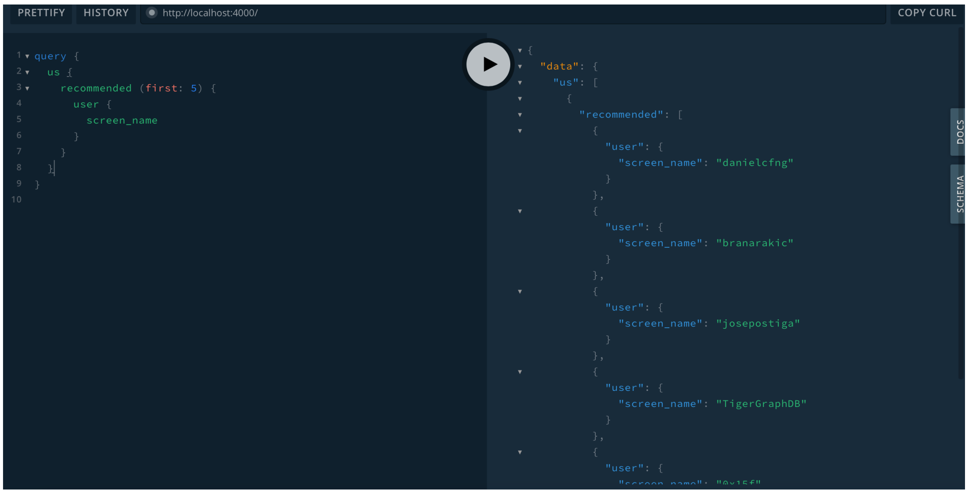
```
query {
  us {
    recommended (first: 5) {
      user {
        screen_name
      }
    }
  }
}
```

Related schema:

```
recommended(first: Int = 10): [UserCount]
  @cypher(
    statement: """
    MATCH (u:User)-[:POSTS]->(t:Tweet)-[:MENTIONS]->(me:Me)
    WITH DISTINCT u, me, count(t) as count
    WHERE (u)-[:FOLLOWS]->(me)
    AND NOT (me)-[:FOLLOWS]->(u)
    RETURN {
      user: u { .* },
      count: count
    }
    ORDER BY count DESC
    LIMIT $first
    """
  )
```



```
      followbackCount: Int
       @cypher(
         statement: """
         MATCH (me:Me)-[:FOLLOWS]->(f:User)
         WHERE (f)-[:FOLLOWS]->(me)
         RETURN count(f)
         """
       )
      recommended(first: Int = 10): [UserCount]
       @cypher(
         statement: """
         MATCH (u:User)-[:POSTS]->(t:Tweet)-[:MENTIONS]->(me:Me)
         WITH DISTINCT u, me, count(t) as count
         WHERE (u)-[:FOLLOWS]->(me)
         AND NOT (me)-[:FOLLOWS]->(u)
         RETURN {
           user: u { .* },
           count: count
         }
         ORDER BY count DESC
         LIMIT $first
         """
       )
       priorityFeed: [UserTweet]
       @cypher(
         statement: """
         MATCH (t:Tweet) WHERE t.created_at IS NOT NULL
         WITH max(t.created_at) - duration('P3D') as recentDate
```



```
query {
  us {
    recommended (first: 5) {
      user {
        screen_name
      }
    }
  }
}
```

```
{
  "data": {
    "us": [
      {
        "recommended": [
          {
            "user": {
              "screen_name": "danielcfng"
            }
          },
          {
            "user": {
              "screen_name": "branarakic"
            }
          },
          {
            "user": {
              "screen_name": "josepostiga"
            }
          },
          {
            "user": {
              "screen_name": "TigerGraphDB"
            }
          },
          {
            "user": {
              "screen_name": "0x15f"
```

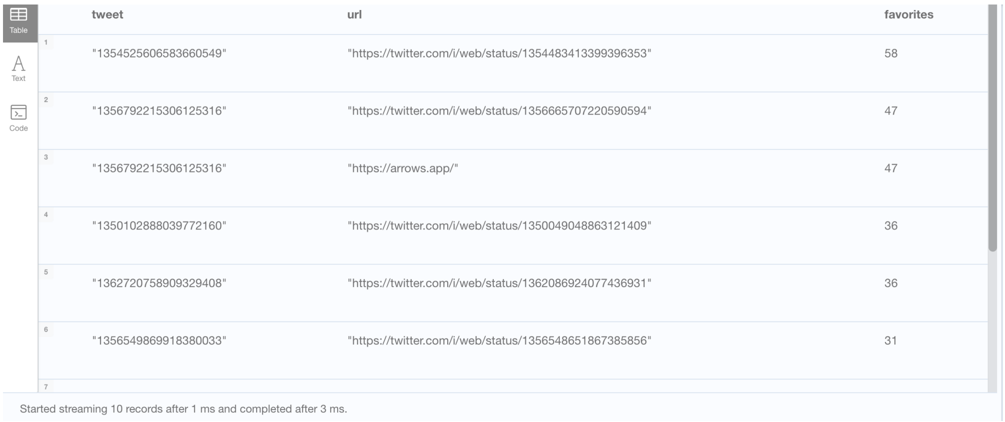
Reflection: The GraphQL schema in the screenshot represents a field called "recommended" with an argument "first" of type Int, which defaults to 10. The annotation "@cypher" indicates that the field is resolved using a Cypher query. The provided GraphQL schema field attempts to express the same logic as the given Cypher query. Without this Recommended field, this query would not be achievable in GraphQL.

6. Favourite Retweet Analysis

Description: Show a list of links that the user “Me” retweet, and how often are they favorited.

Cypher query and result:

```
MATCH
  (:User:Me)-[:POSTS]->
  (t:Tweet)-[:RETWEETS]->(rt)-[:CONTAINS]->(link:Link)
RETURN
  t.id_str AS tweet, link.url AS url, rt.favorites AS favorites
ORDER BY
  favorites DESC
LIMIT 10
```

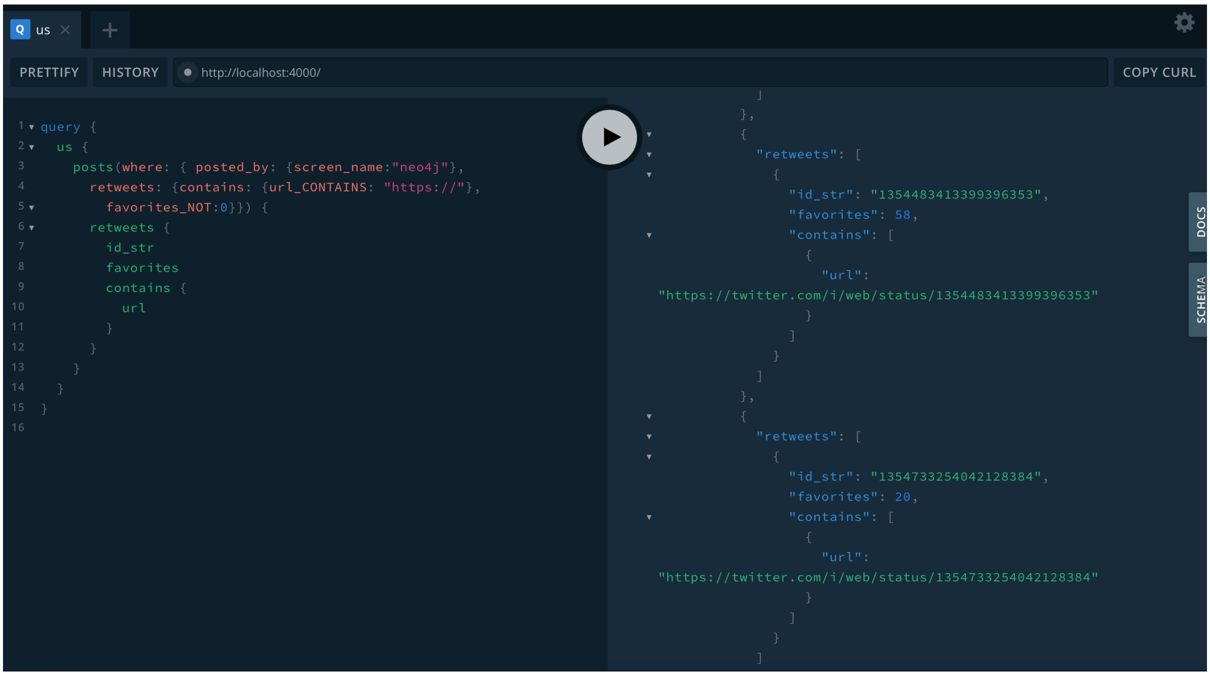


	tweet	url	favorites
1	"1354525606583660549"	"https://twitter.com/i/web/status/1354483413399396353"	58
2	"1356792215306125316"	"https://twitter.com/i/web/status/1356665707220590594"	47
3	"1356792215306125316"	"https://arrows.app/"	47
4	"1350102888039772160"	"https://twitter.com/i/web/status/1350049048863121409"	36
5	"1362720758909329408"	"https://twitter.com/i/web/status/1362086924077436931"	36
6	"1356549869918380033"	"https://twitter.com/i/web/status/1356548651867385856"	31
7			

Started streaming 10 records after 1 ms and completed after 3 ms.

GraphQL Query and result:

```
query {
  us {
    posts(where: { posted_by: {screen_name:"neo4j"}, retweets: {contains: {url_CONTAINS:
"https://"}, favorites_NOT:0}}) {
      retweets {
        id_str
        favorites
        contains {
          url
        }
      }
    }}}
```



```
query {
  us {
    posts(where: { posted_by: {screen_name:"neo4j"},
      retweets: {contains: {url_CONTAINS: "https://"},
        favorites_NOT:0}}) {
      retweets {
        id_str
        favorites
        contains {
          url
        }
      }
    }
  }
}
```

```
            ]
          },
          {
            "retweets": [
              {
                "id_str": "1354483413399396353",
                "favorites": 58,
                "contains": [
                  {
                    "url":
"https://twitter.com/i/web/status/1354483413399396353"
                  }
                ]
              }
            ]
          },
          {
            "retweets": [
              {
                "id_str": "1354733254042128384",
                "favorites": 20,
                "contains": [
                  {
                    "url":
"https://twitter.com/i/web/status/1354733254042128384"
                  }
                ]
              }
            ]
```

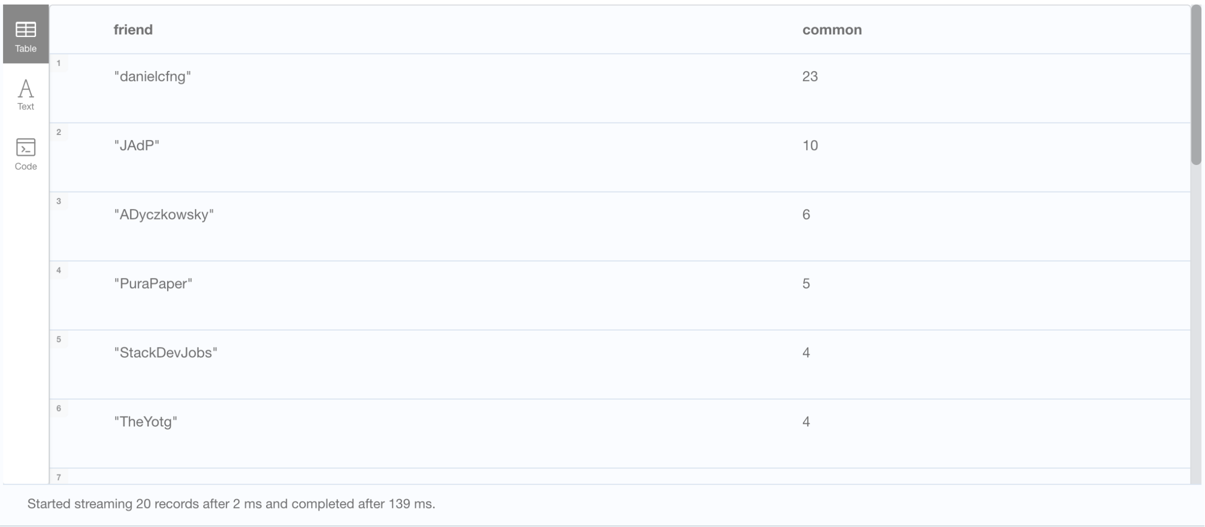
Reflection: Although the GrapQL query can obtain the list of “my” retweets with urls and favourites count, the list cannot be sorted and limited, as no options argument in **posts** to sort and limit based on the favourites field of retweets.

7. Common hashtags

Description: Find users who tweet with user Me’s top hashtags.

Cypher query and result:

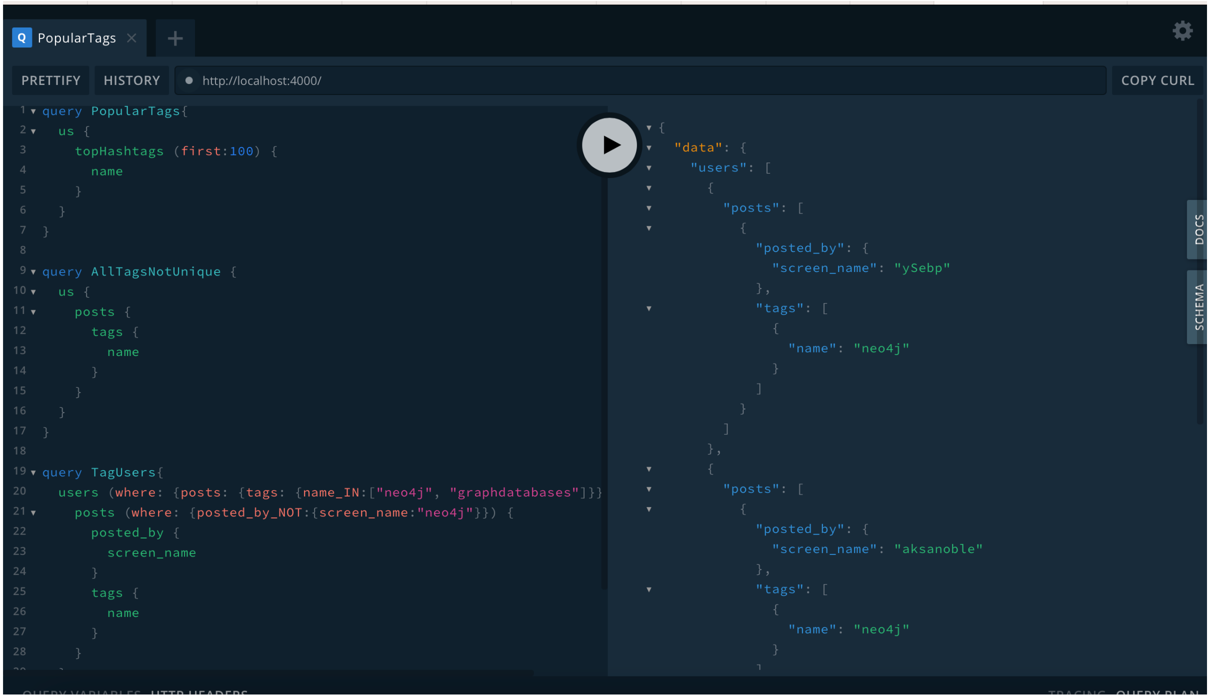
```
// Users tweeting with your top hashtags
MATCH
  (me:User:Me)-[:POSTS]->(tweet:Tweet)-[:TAGS]->(ht)
MATCH
  (ht)<-[:TAGS]-(tweet2:Tweet)<-[:POSTS]-(sugg:User)
WHERE
  sugg <> me
  AND NOT
  (tweet2)-[:RETWEETS]->(tweet)
WITH
  sugg, collect(distinct(ht)) as tags
RETURN
  sugg.screen_name as friend, size(tags) as common
ORDER BY
  common DESC
LIMIT 20
```



	friend	common
1	"danielcfng"	23
2	"JAdP"	10
3	"ADyczkowsky"	6
4	"PuraPaper"	5
5	"StackDevJobs"	4
6	"TheYotg"	4
7		

Started streaming 20 records after 2 ms and completed after 139 ms.

GraphQL Query and result:



```
query PopularTags{
  us {
    topHashtags (first:100) {
      name
    }
  }
}

query AllTagsNotUnique {
  us {
    posts {
      tags {
        name
      }
    }
  }
}

query TagUsers{
  users (where: {posts: {tags: {name_IN:["neo4j", "graphdatabases"]}}}
    posts (where: {posted_by_NOT:{screen_name:"neo4j"}}) {
      posted_by {
        screen_name
      }
      tags {
        name
      }
    }
```

```
{
  "data": {
    "users": [
      {
        "posts": [
          {
            "posted_by": {
              "screen_name": "ySebp"
            },
            "tags": [
              {
                "name": "neo4j"
              }
            ]
          }
        ]
      },
      {
        "posts": [
          {
            "posted_by": {
              "screen_name": "aksanoble"
            },
            "tags": [
              {
                "name": "neo4j"
              }
```

query PopularTags{
  us {
    topHashtags (first:100) {
      name
    }
  }
}

query AllTagsNotUnique {
  us {
    posts {

```
      tags {
        name
      }
    }
  }
}
```

```
query TagUsers{
  users (where: {posts: {tags: {name_IN:["neo4j", "graphdatabases"… (insert result from above query)]}}}}){
    posts (where: {posted_by_NOT:{screen_name:"neo4j"}}) {
                                    posted_by {
        screen_name
      }
      tags {
        name
      }
    }
  }
}
```

Reflection: This query is complex and hard to achieve in GraphQL. By trying the above separate queries, we can get a list of posts containing tags used by “me”, but the condition “NOT (tweet2)-[:RETWEETS]->(tweet)” is neglected, and it is hard to do Count and Distinct in GraphQL as well.

8. Mutual followers

Description: finding the user that share the highest number of mutual followers with "Me”

```
MATCH (follower2:User)-[:FOLLOWS]->(follower:User)-[:FOLLOWS]->(u:User:Me)
WHERE (follower2)-[:FOLLOWS]->(u:User:Me) AND u <> follower2
RETURN follower2.screen_name AS User, COUNT(*) AS mutualFollowers
ORDER BY mutualFollowers DESC
LIMIT 1
```

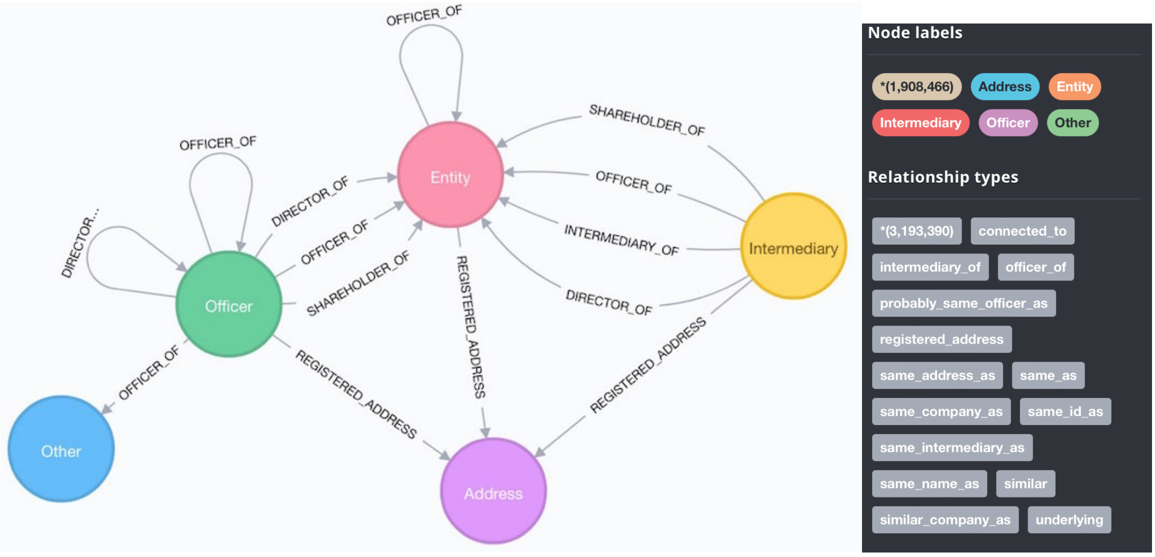
Post referred to:
https://medium.com/javarevisited/implementing-facebook-social-graph-using-spring-and-neo4j-81c1b67351b7

Reflection: For cases where we want to find the mutual objects between two other objects, for instance, find mutual friends between two users, Cypher queries would be better. To achieve this in GraphQL, we would need a resolver function for this query to handle the logic of retrieving the mutual. Within the resolver, we would construct and execute the necessary queries to find the mutual objects based on the provided input. Similarly, Cypher queries perform better when finding mutual followers not followed by “me”, thus providing friend recommendations.

Offshore Leaks Database by ICIJ Graph Example

Resource used:
https://github.com/neo4j-graph-examples/icij-offshoreleaks/tree/main

Relationships graph:



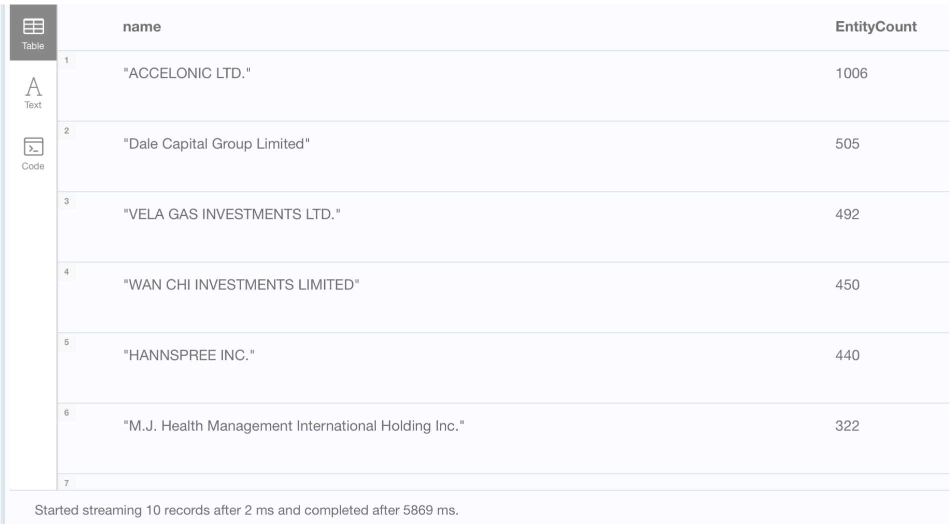
The Offshore Leaks data exposes a set of connections between people and offshore entities. Graph databases are the best way to explore the relationships between these people and entities.

Queries explored (will add more):

1. **Companies with most officers**

Cypher query and result:

```
MATCH (o:Officer)-[:officer_of]->(e:Entity)

WITH e, COUNT(e) AS EntityCount

ORDER BY EntityCount DESC

RETURN e.name AS name, EntityCount

LIMIT 10
```



	name	EntityCount
1	"ACCELONIC LTD."	1006
2	"Dale Capital Group Limited"	505
3	"VELA GAS INVESTMENTS LTD."	492
4	"WAN CHI INVESTMENTS LIMITED"	450
5	"HANNSPREE INC."	440
6	"M.J. Health Management International Holding Inc."	322
7		

Started streaming 10 records after 2 ms and completed after 5869 ms.

GraphQL Query and result:

```
query {
  entities {
    officers {
      name
    }
  }
}
```

Since this query involves aggregates, ordering and limits, it is harder to achieve in GraphQL as compared to Cypher.

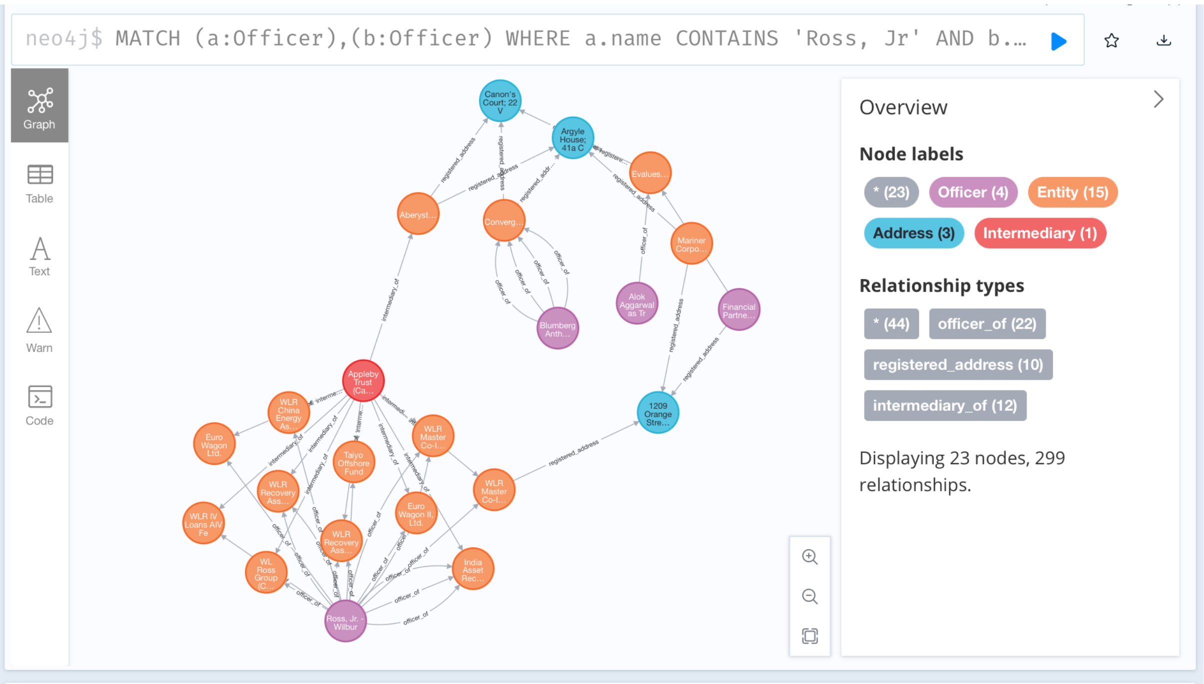
2. **Discover the shortest paths between two entity officers through a set of Entity or Address nodes.**

Cypher query and result:

```
MATCH (a:Officer),(b:Officer)

WHERE a.name CONTAINS 'Ross, Jr'

  AND b.name CONTAINS 'Grant'
```

```
MATCH p=allShortestPaths((a)-[:officer_of|intermediary_of|registered_address*..10]-(b))

RETURN p

LIMIT 50
```

The resulting graph allows us to explore how these people are connected.



```
neo4j$ MATCH (a:Officer),(b:Officer) WHERE a.name CONTAINS 'Ross, Jr' AND b.…
```

Overview

Node labels

* (23) Officer (4) Entity (15)

Address (3) Intermediary (1)

Relationship types

* (44) officer_of (22)

registered_address (10)

intermediary_of (12)

Displaying 23 nodes, 299 relationships.

GraphQL Queries:

```
query {
  officers(where: { name_CONTAINS: "Ross, Jr" }) {
    node_id
    name
    connected_to {
      type
      name
    }
  }
}
```

```
query {
  officers(where: { name_CONTAINS: "Grant" }) {
    node_id
    name
    connected_to {
      type
      name
    }
  }
}
```

Reflection: The Cypher query searches for two officers, one with the name containing "Ross, Jr" and the other with the name containing "Grant." It then finds the shortest paths between them, considering relationships of types "officer_of," "intermediary_of," and "registered_address" with a maximum depth of 10. Finally, it returns the paths with a limit of 50.

The GraphQL query first fetches officers with names containing "Ross, Jr" and then retrieves their relationships. Then it fetched fetches officers with names containing "Grant" and then retrieves their relationships. However, it is hard to discover shortest path with GraphQL query in this case.

3. **Find the officers who are connected to the same company and have the same nationality.**

Cypher query and result:

```
MATCH (o1:Officer)-[:officer_of]->(e:Entity)<-[:officer_of]-(o2:Officer)

WHERE o1.nationality = o2.nationality

RETURN o1.name AS officer1, o2.name AS officer2, e.name AS companyName,
o1.nationality AS nationality

LIMIT 10
```