CS G516 – Advanced Database Systems

Mini Project

BITS Pilani
Hyderabad Campus

Submitted by –

Abhishek	2021H1030070H
Chirag Jain	2021H1030072H
Subhabrata Kanjilal	2021H1030098H
Pranjal Gupta	2021H1030102H

Under the supervision of –

Dr. Subhrakanta Panda,

Assistant Professor , Dept. of CSIS ,
BITS Pilani , Hyderabad Campus

innovate achieve lead

Introduction

1. Limitations to Relational Databases

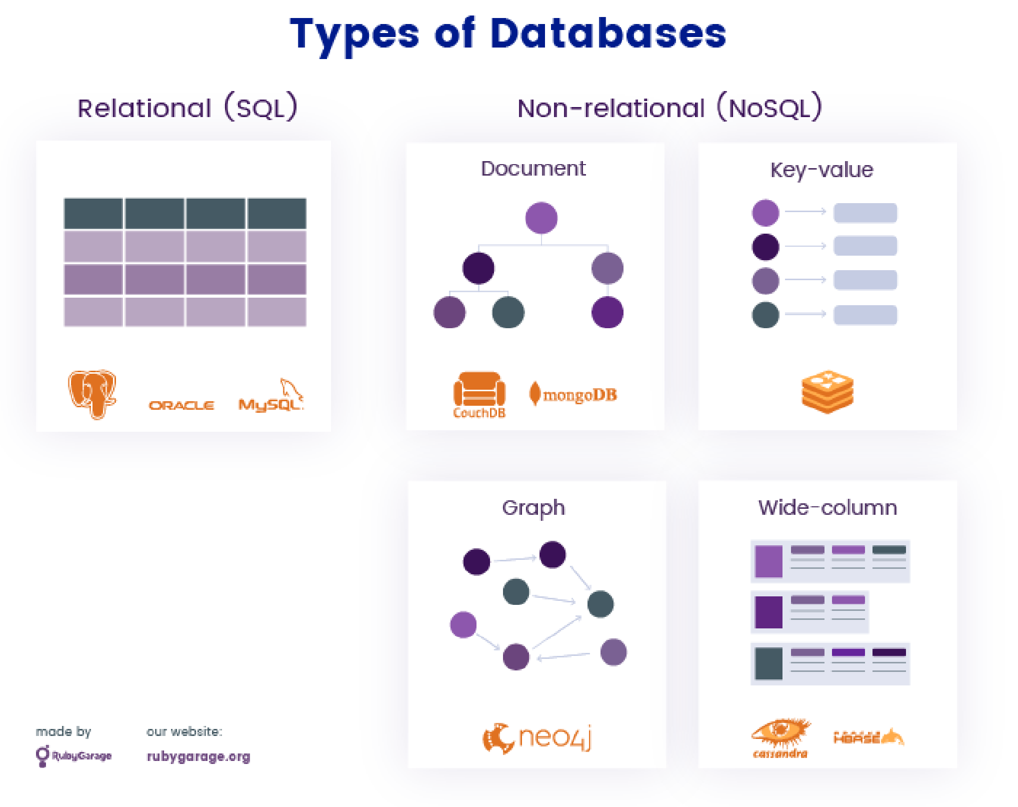
Relational Databases have the following limitations:

- **Volume of data –** Relational databases cannot be optimized to handle very large amounts of data, especially, ever-growing data.
- **Latency –** the performance of relational databases storage system suffers when they deal with huge numbers of read/write operations, hence, latency increases drastically.
- **Inability to represent relationships –** Relations data stores cannot describe relationships other than standard one-to-one, many-to-one and many-to-many.
- **Data –** Lack flexibility dealing with the data that can't be described using database schemes, e.g., Binary or semi-structured.
- **Scalability –** Horizontal scaling is inefficient.

Hence, to overcome these limitations, a number of different non-relational SQL(NoSQL) databases has been suggested as per the diagram below.

Our main focus in this project is to store relationships between the already connected data. Hence, for this purpose graph databases prove to be most efficient as they can handle large amounts of connected data.

The vertices in graph database represents entities, and edges represents the relationship between the two vertices. Edges can have value, called weight to show the strength of the relationship.

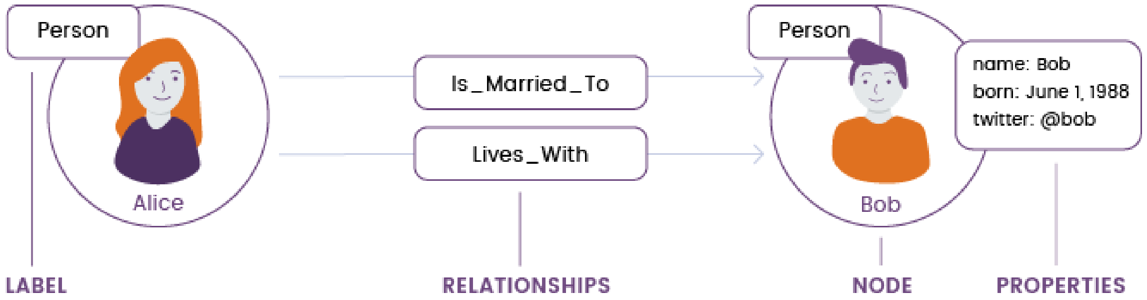


2.Neo4j

Neo4j has the following components:

- **Nodes –** these are equivalent to vertices which has labels(it's identity/role) and properties(it's attributes).
- **Relationships –** these are equivalent to edges which represents relationships among two more nodes. It is observed that two nodes can have multiple relationships and relationships can have one or more properties.
- **Labels –** these are used to group nodes that can be assigned multiple labels. Labels are indexed so as to speed up the search query of finding nodes.
- **Properties –** these are attributes associated with both nodes and relationships. Neo4j allows storing data as key-value pairs.

Here is an example of Neo4j showing a simple graph data model :



3.Neo4j VS Other Databases:

Data storage	Graph storage structure	Fixed, predefined tables with rows and columns	Connected data not supported at the database level
Data modeling	Flexible data model	Database model must be developed from a logical model	Not suitable for enterprise architectures
Query performance	Great performance regardless of number and depth of connections	Data processing speed slows with growing number of joins	Relationships must be created at the application level
Query language	**Cypher**: native graph query language	**SQL**: complexity grows as the number of joins increases	Different languages are used but none is tailored to express relationships
Transaction support	Retains ACID transactions	ACID transaction support	BASE transactions prove unreliable for data relationships

Processing at scale	Inherently scalable for pattern-based queries	Scales through replication, but it’s costly	Scalable, but data integrity isn’t trustworthy

4. Advantages of Neo4j Database:

The following are advantages of using neo4j databases:

- **Performance –** Performance of Neo4j is not disturbed by ever growing size of the data.
- **Flexibility –** Neo4j provides flexibility as we can adjust to changes in structure and schema of the graph model and can also upgrade the data structure without damaging the existing functionality.
- **Agility -** Neo4j structured database is easy-to-upgrade and our data can evolve with our application.

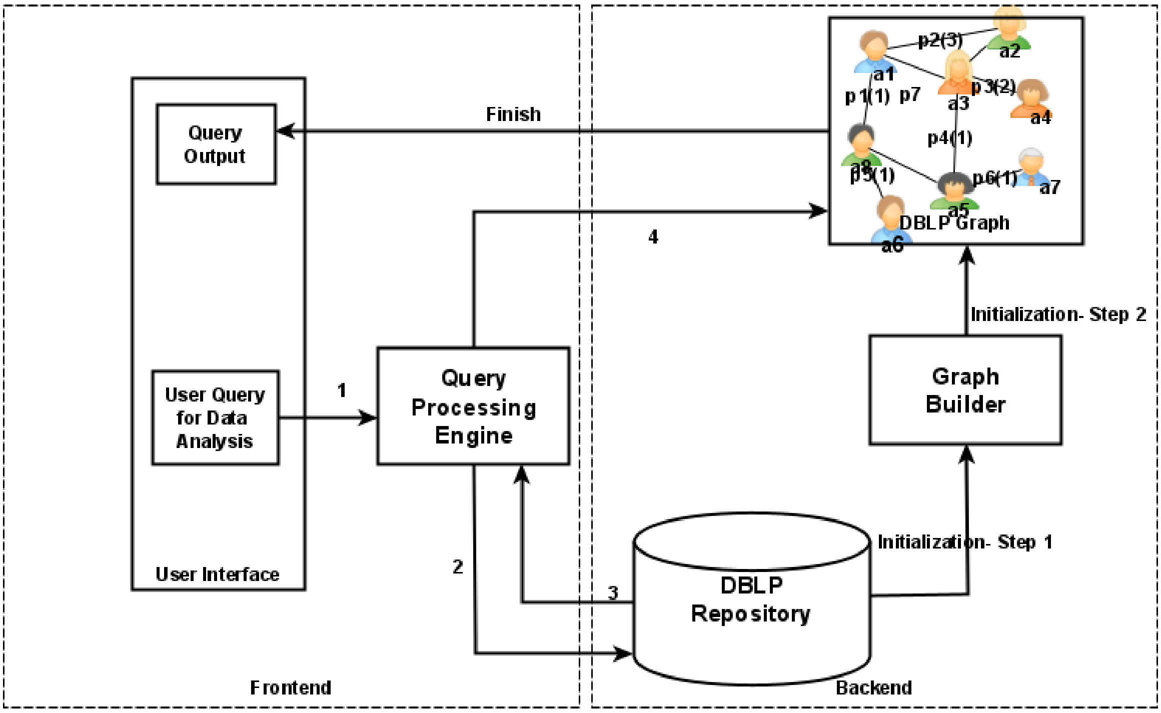
Project Overview

1. Problem Statement

In the present era, scientific publications are increasing rapidly, making the network of collaborations, topics, papers, etc., more complex than ever. Not surprisingly, the term Big Scholarly Data has been recently coined to refer to the rapidly growing scholarly source of information (e.g., extensive collections of scholarly data with million authors, papers, citations, figures, tables, as well as massive scale related data such as scholarly networks). The analysis of such data and network is helpful for researchers to identify colleagues working on similar topics, make a profile of a researcher for understanding its research interests based on its academic records and scores, and identify experts on a specific research area.

The DBLP server provides bibliographic information of Scholarly Data on major **computer science journals and proceedings**. It is a high-quality digital library with complete coverage of computer science literature. DBLP data if modelled in a graph format would allow several outcomes such as finding experts in the community, community detection, community mining, keyword extraction, etc. Researchers in academia are categorized into communities and characterized by topics, interests, geographical influence, etc. In the DBLP, the community is a significant object of interest. Generally, a community is a subset of nodes within the graph such that connections between the nodes are denser than connections with the rest of the graph. Relation among the entities can be represented in a graph format using Neo4j. Specifically, Neo4j is a graph database. It models attributes, labels, and directed multi-graphs. Neo4j makes use of the declarative Cypher language for querying the graph-store.

Recent studies state a growing interest in studying and understanding the network of scholars/researchers to find research experts, trending topics, influencing scholars, etc., by searching scholarly data in the graph format. In view of providing a tool to researchers for querying the DBLP bibliography in a graph format, students are required to build an application through a Python shell interface or a Web GUI. Any graph-based queries on Neo4j can be performed using the Cypher query language with the below specified model.

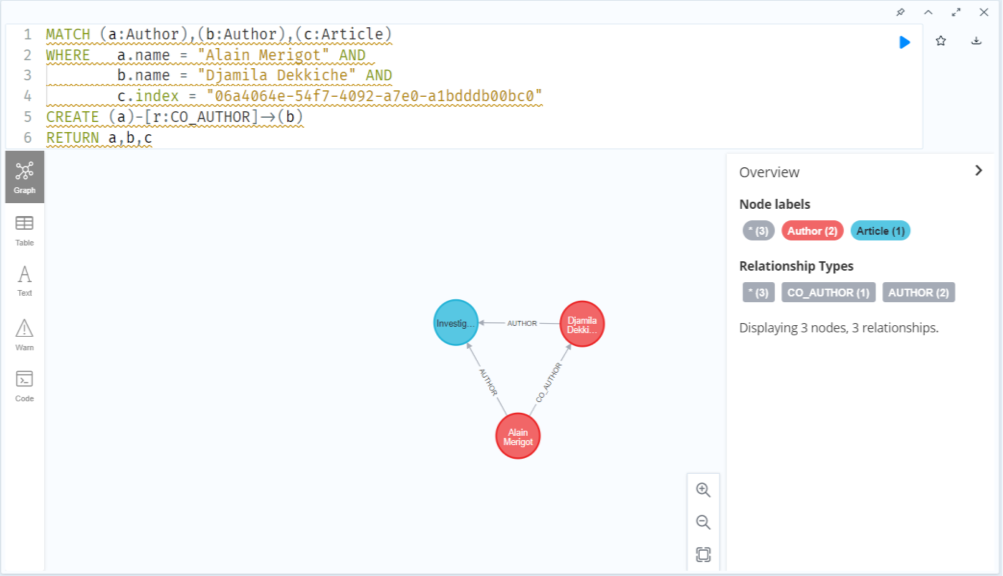


2. Cypher Query with Neo4j

We ran few queries with Neo4j and got the following outputs:

1. Create Co Author Relationship

If 2 or more authors were working on same article, we add co-author relationship between them.

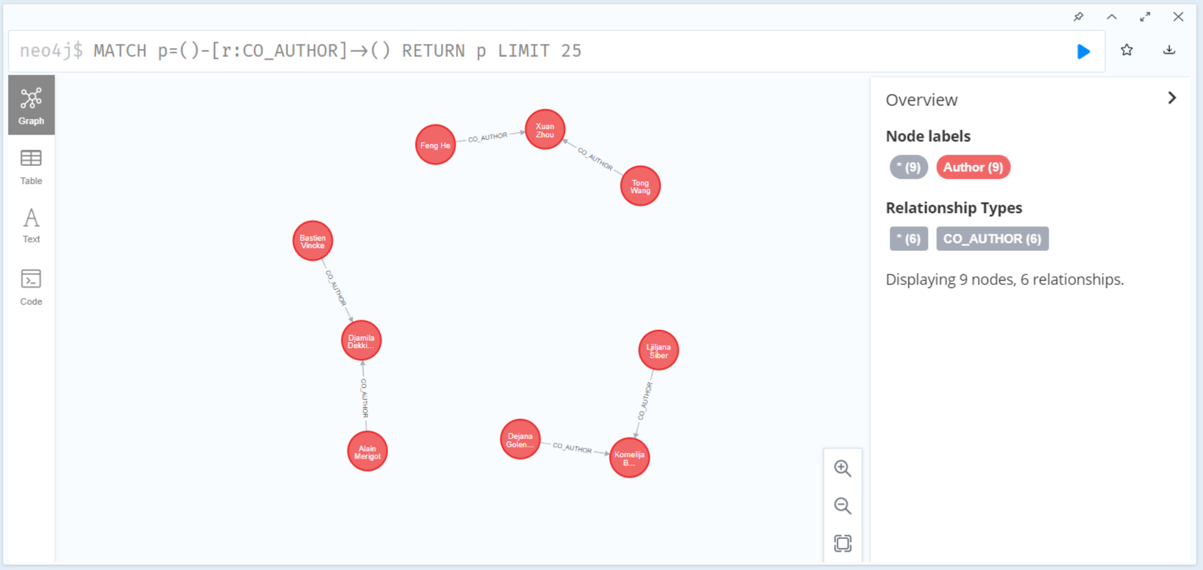


```
MATCH (a:Author),(b:Author),(c:Article)
WHERE   a.name = "Alain Merigot" AND
        b.name = "Djamila Dekkiche" AND
        c.index = "06a4064e-54f7-4092-a7e0-a1bdddb00bc0"
CREATE (a)-[r:CO_AUTHOR]→(b)
RETURN a,b,c
```

Displaying 3 nodes, 3 relationships.

2. Create Co Author Relationship (Process Update)

Earlier we created, co-author relationship with specific authors and articles. Now, we ran our query for a small part of the database to generate the co-author relationships.



```
neo4j$ MATCH p=()-[r:CO_AUTHOR]→() RETURN p LIMIT 25
```

Overview

Node labels

* (9) Author (9)

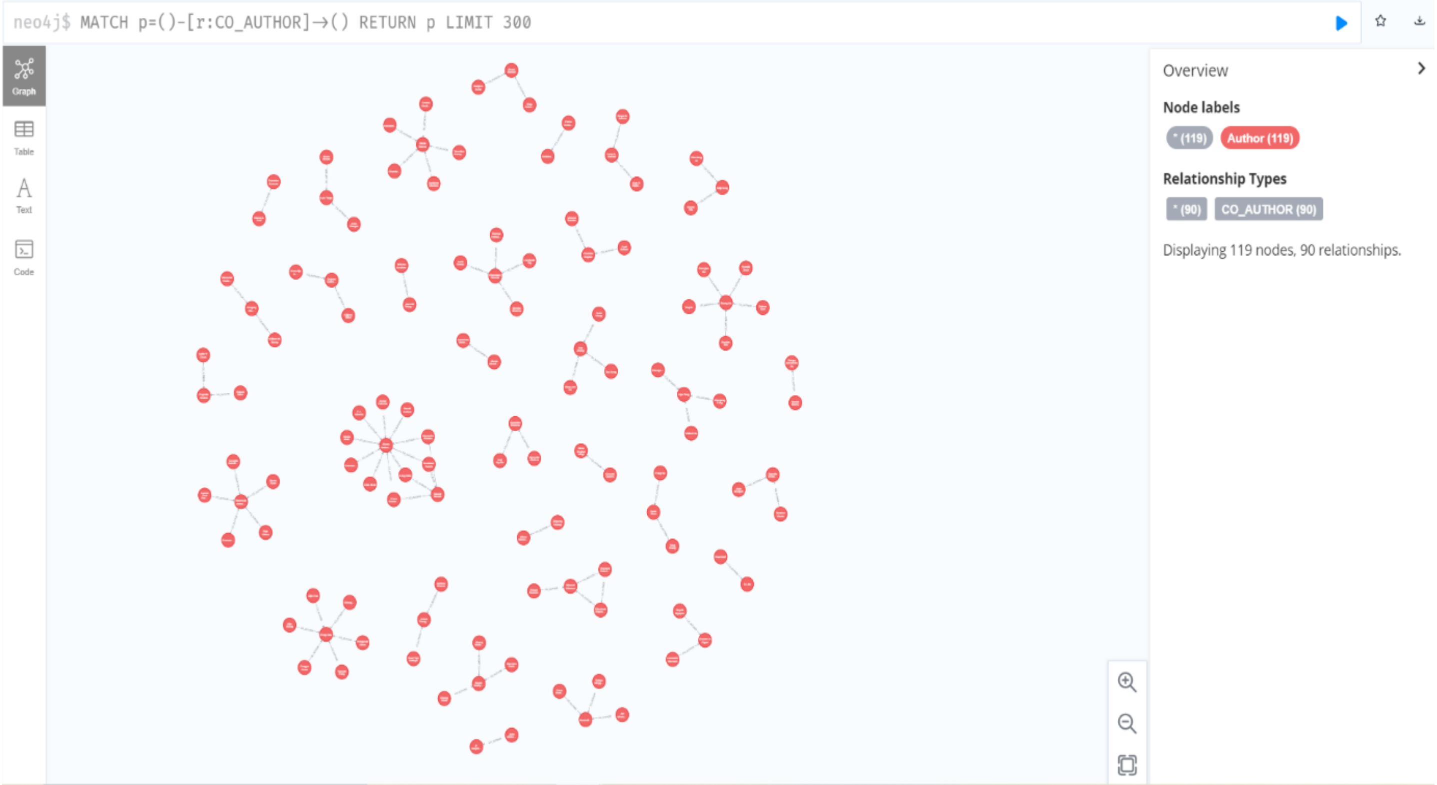
Relationship Types

* (6) CO_AUTHOR (6)

Displaying 9 nodes, 6 relationships.

3. Create Co Author Relationship (Full Graph)

We now ran our query for the full graph



```
neo4j$ MATCH p=()-[r:CO_AUTHOR]→() RETURN p LIMIT 300
```

Overview

Node labels

* (119) Author (119)

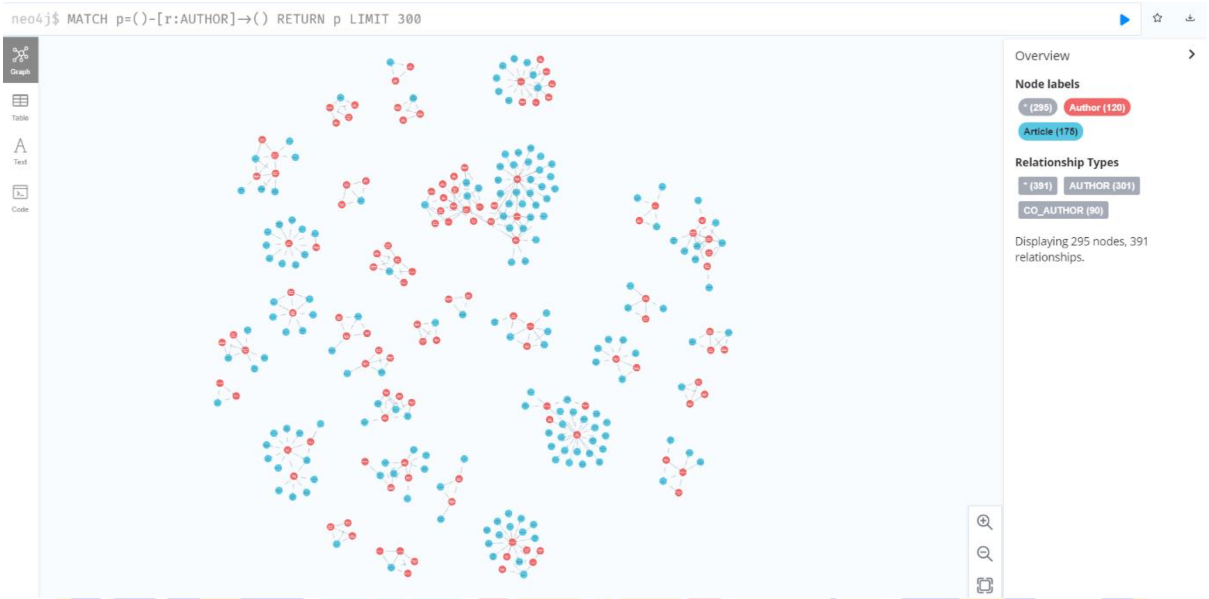
Relationship Types

* (90) CO_AUTHOR (90)

Displaying 119 nodes, 90 relationships.

4. Author Co-Author Relationship (Full - Graph)

In this query we figured which co-author is related to which lead author and is most likely to work in the near future with him/her.



neo4j$ MATCH p=()-[r:AUTHOR]→() RETURN p LIMIT 300

Overview

Node labels

* (295) Author (120)

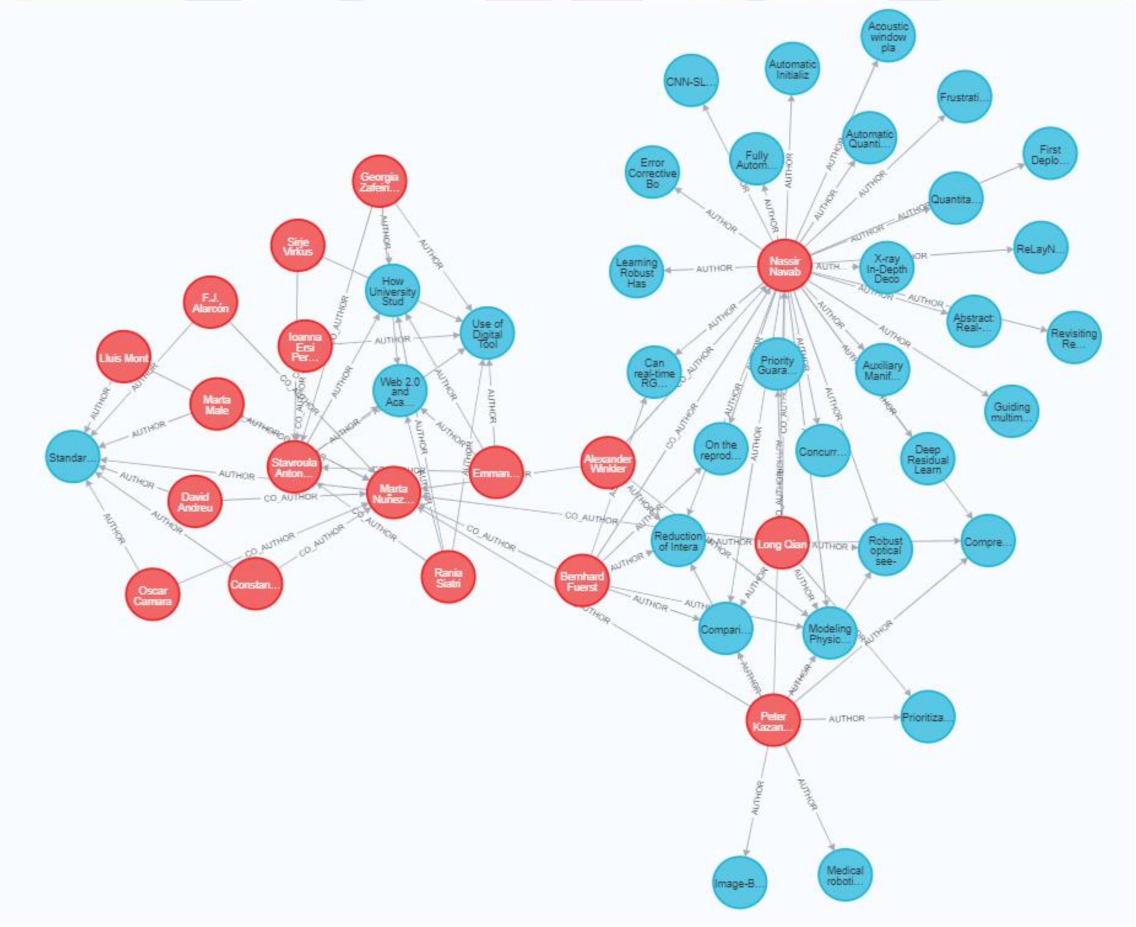
Article (175)

Relationship Types

* (391) AUTHOR (301)

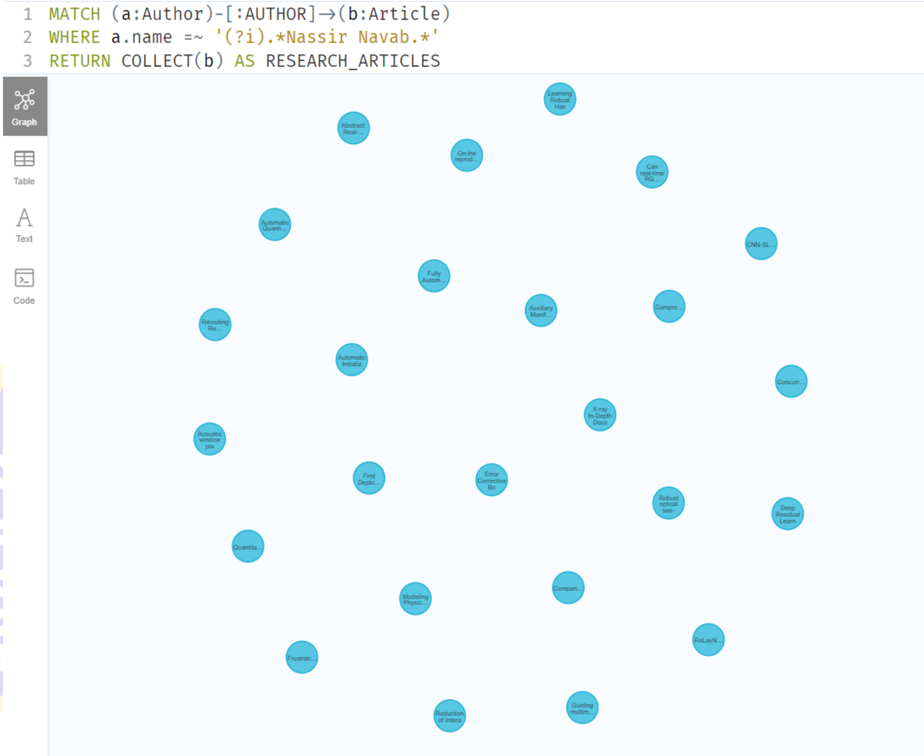
CO_AUTHOR (90)

Displaying 295 nodes, 391 relationships.



5. Author Name to Research Article Search

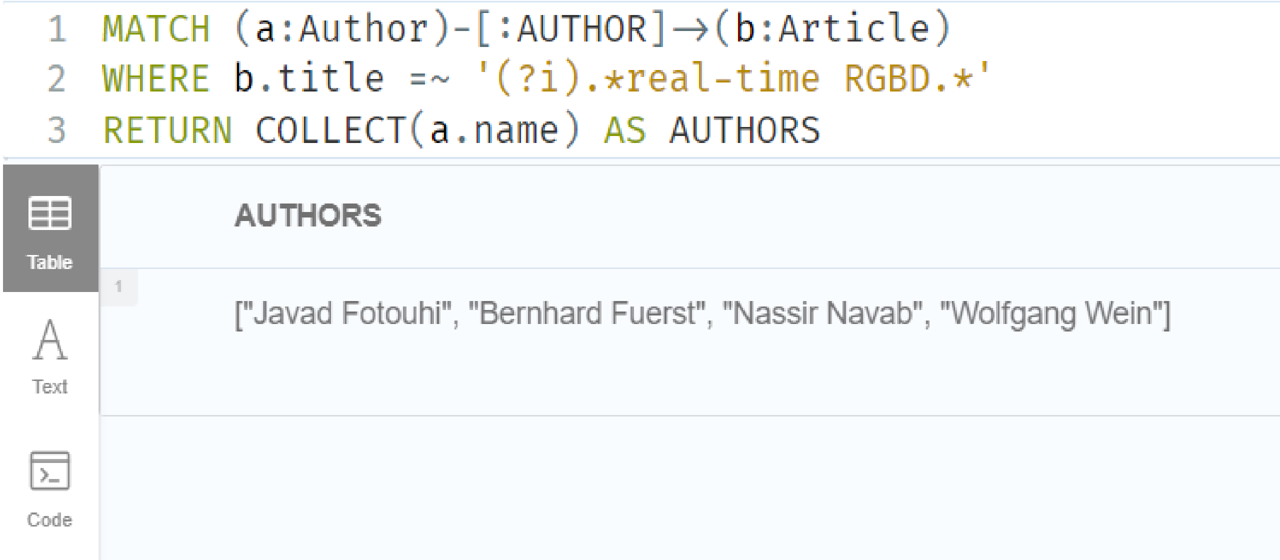
In this query, we search the research articles written a by specific author given as input.



```
MATCH (a:Author)-[:AUTHOR]→(b:Article)
WHERE a.name =~ '(?i).*Nassir Navab.*'
RETURN COLLECT(b) AS RESEARCH_ARTICLES
```

6. Keyword to Author Search

We use regular expressions search for particular authors.

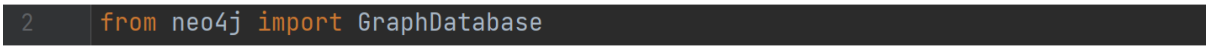


```
MATCH (a:Author)-[:AUTHOR]→(b:Article)
WHERE b.title =~ '(?i).*real-time RGBD.*'
RETURN COLLECT(a.name) AS AUTHORS
```

	AUTHORS
1	["Javad Fotouhi", "Bernhard Fuerst", "Nassir Navab", "Wolfgang Wein"]

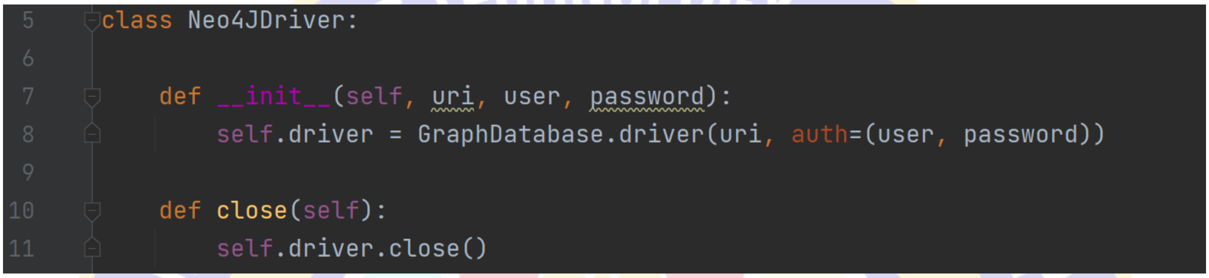
3. Connecting Python with Neo4j

1. Import Statement



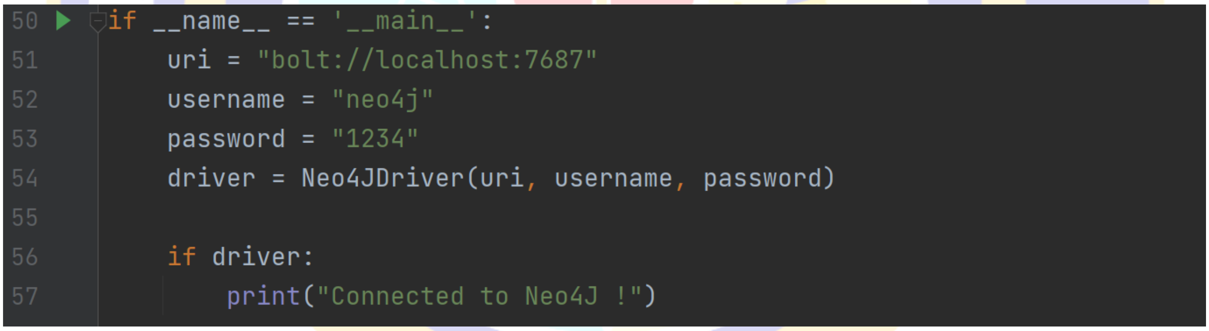
```python
from neo4j import GraphDatabase
```

2. Driver Class



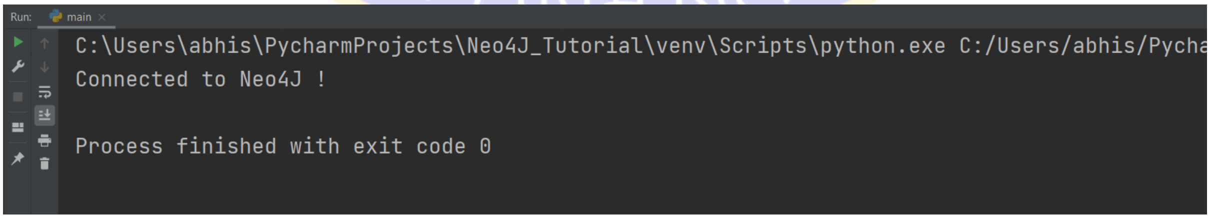
```python
class Neo4JDriver:

    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri, auth=(user, password))

    def close(self):
        self.driver.close()
```

3. Driver Connection



```python
if __name__ == '__main__':
    uri = "bolt://localhost:7687"
    username = "neo4j"
    password = "1234"
    driver = Neo4JDriver(uri, username, password)

    if driver:
        print("Connected to Neo4J !")
```

4. Database connectivity Test

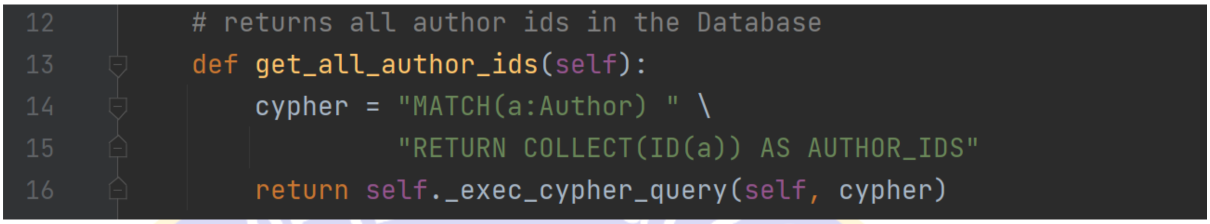


```
C:\Users\abhis\PycharmProjects\Neo4J_Tutorial\venv\Scripts\python.exe C:/Users/abhis/Pycha
Connected to Neo4J !

Process finished with exit code 0
```

4. Python – Query Processing Engine

1. Return a list of all Author IDs in the Database

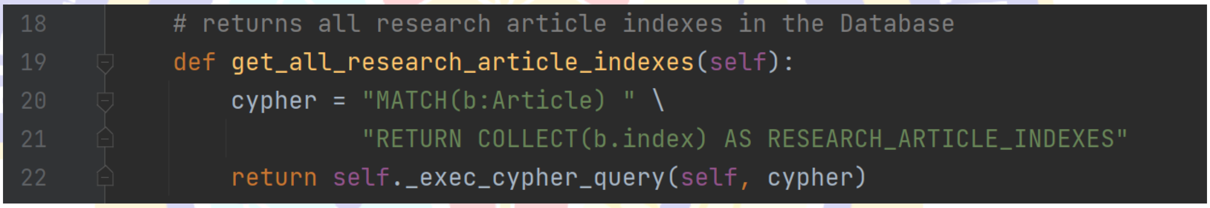
IDs are unique to each author node and are useful in fast processing of the cypher query.



```
    # returns all author ids in the Database
    def get_all_author_ids(self):
        cypher = "MATCH(a:Author) " \
                 "RETURN COLLECT(ID(a)) AS AUTHOR_IDS"
        return self._exec_cypher_query(self, cypher)
```

2. Return a list of all Research article indexes in the Database

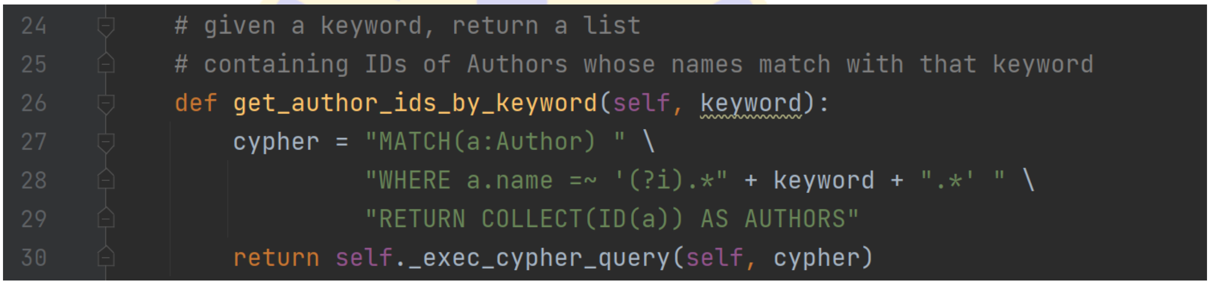
Here, we are returning research-article indexes instead of research article titles in order to uniquely identify each research article in the Database repository so that the Query operations are comparatively faster.



```
    # returns all research article indexes in the Database
    def get_all_research_article_indexes(self):
        cypher = "MATCH(b:Article) " \
                 "RETURN COLLECT(b.index) AS RESEARCH_ARTICLE_INDEXES"
        return self._exec_cypher_query(self, cypher)
```

3. Given a keyword, return the list of IDs of Authors whose names match with the given keyword.

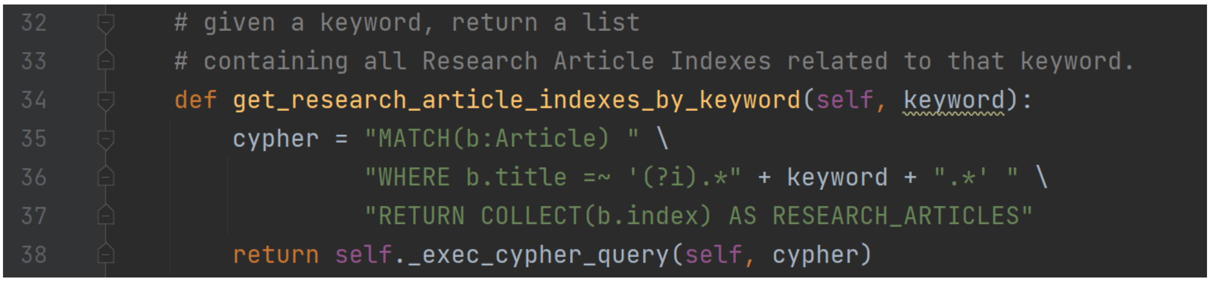
Here, we will use Regular Expression to filter out all author names related to that keyword and then return all the matching authors as their IDs in a list.



```
    # given a keyword, return a list
    # containing IDs of Authors whose names match with that keyword
    def get_author_ids_by_keyword(self, keyword):
        cypher = "MATCH(a:Author) " \
                 "WHERE a.name =~ '(?i).*" + keyword + ".*' " \
                 "RETURN COLLECT(ID(a)) AS AUTHORS"
        return self._exec_cypher_query(self, cypher)
```

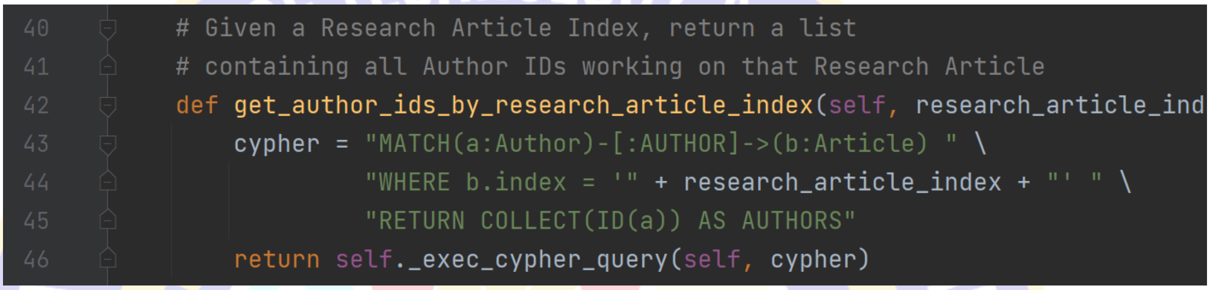
4. Given a keyword, return the list of all research article indexes related to that keyword.

Here also, we will use Regular Expression to filter out all research article names related to that keyword and return all matching research articles as their indexes.



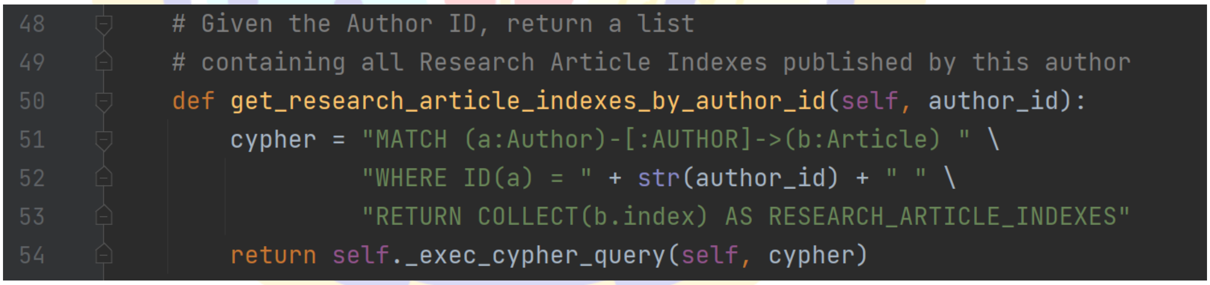
```python
# given a keyword, return a list
# containing all Research Article Indexes related to that keyword.
def get_research_article_indexes_by_keyword(self, keyword):
    cypher = "MATCH(b:Article) " \
             "WHERE b.title =~ '(?i).*" + keyword + ".*' " \
             "RETURN COLLECT(b.index) AS RESEARCH_ARTICLES"
    return self._exec_cypher_query(self, cypher)
```

5. Given a Research Article Index, return a list containing IDs of Authors working on that Research Article



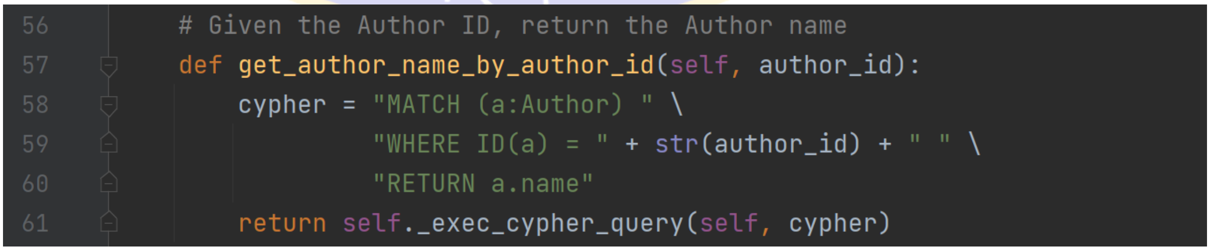
```python
# Given a Research Article Index, return a list
# containing all Author IDs working on that Research Article
def get_author_ids_by_research_article_index(self, research_article_ind
    cypher = "MATCH(a:Author)-[:AUTHOR]->(b:Article) " \
             "WHERE b.index = '" + research_article_index + "' " \
             "RETURN COLLECT(ID(a)) AS AUTHORS"
    return self._exec_cypher_query(self, cypher)
```

6. Given an Author ID, return a list containing all Research Article Indexes published by this Author



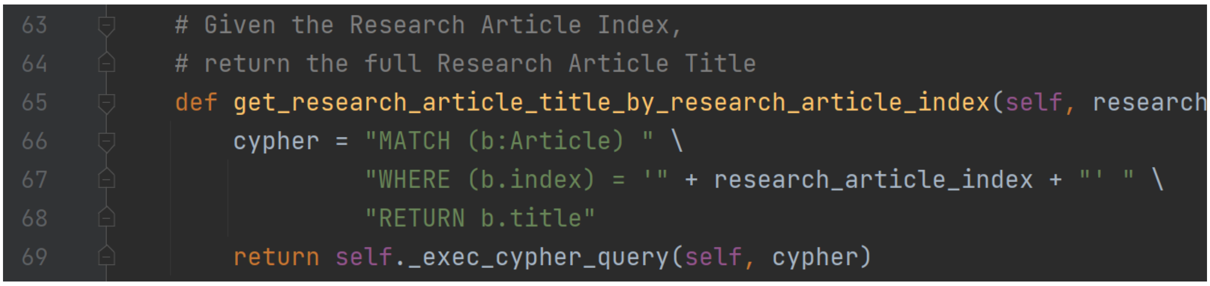
```python
# Given the Author ID, return a list
# containing all Research Article Indexes published by this author
def get_research_article_indexes_by_author_id(self, author_id):
    cypher = "MATCH (a:Author)-[:AUTHOR]->(b:Article) " \
             "WHERE ID(a) = " + str(author_id) + " " \
             "RETURN COLLECT(b.index) AS RESEARCH_ARTICLE_INDEXES"
    return self._exec_cypher_query(self, cypher)
```

7. Given the Author ID, return the Author name.



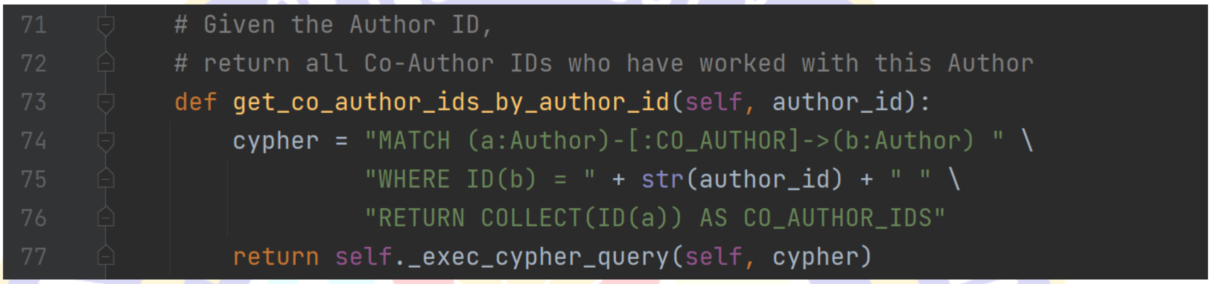
```python
# Given the Author ID, return the Author name
def get_author_name_by_author_id(self, author_id):
    cypher = "MATCH (a:Author) " \
             "WHERE ID(a) = " + str(author_id) + " " \
             "RETURN a.name"
    return self._exec_cypher_query(self, cypher)
```

8. Given a Research Article Index, return the Research Article Title.



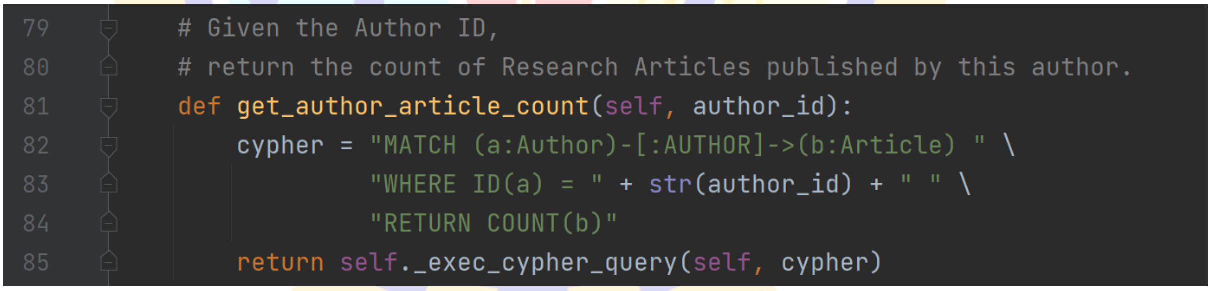
```python
    # Given the Research Article Index,
    # return the full Research Article Title
    def get_research_article_title_by_research_article_index(self, research
        cypher = "MATCH (b:Article) " \
                 "WHERE (b.index) = '" + research_article_index + "' " \
                 "RETURN b.title"
        return self._exec_cypher_query(self, cypher)
```

9. Given an Author ID, return a list containing the IDs of Co-Authors who have collaborated with this Author



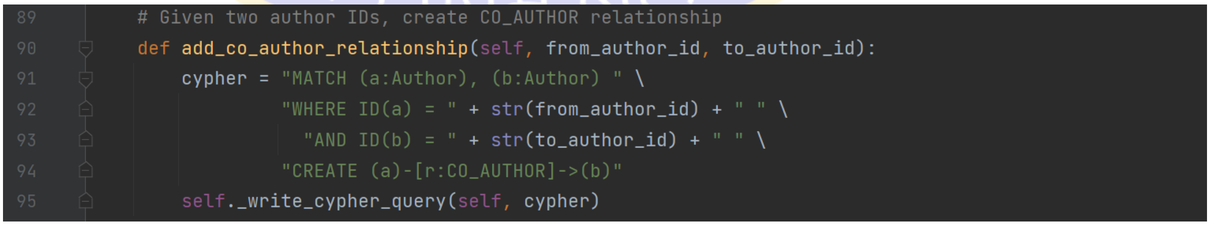
```python
    # Given the Author ID,
    # return all Co-Author IDs who have worked with this Author
    def get_co_author_ids_by_author_id(self, author_id):
        cypher = "MATCH (a:Author)-[:CO_AUTHOR]->(b:Author) " \
                 "WHERE ID(b) = " + str(author_id) + " " \
                 "RETURN COLLECT(ID(a)) AS CO_AUTHOR_IDS"
        return self._exec_cypher_query(self, cypher)
```

10. Given an Author ID, return the count of all Research Articles published by this author



```python
    # Given the Author ID,
    # return the count of Research Articles published by this author.
    def get_author_article_count(self, author_id):
        cypher = "MATCH (a:Author)-[:AUTHOR]->(b:Article) " \
                 "WHERE ID(a) = " + str(author_id) + " " \
                 "RETURN COUNT(b)"
        return self._exec_cypher_query(self, cypher)
```

11. Add Co-Author relationship between two given Authors

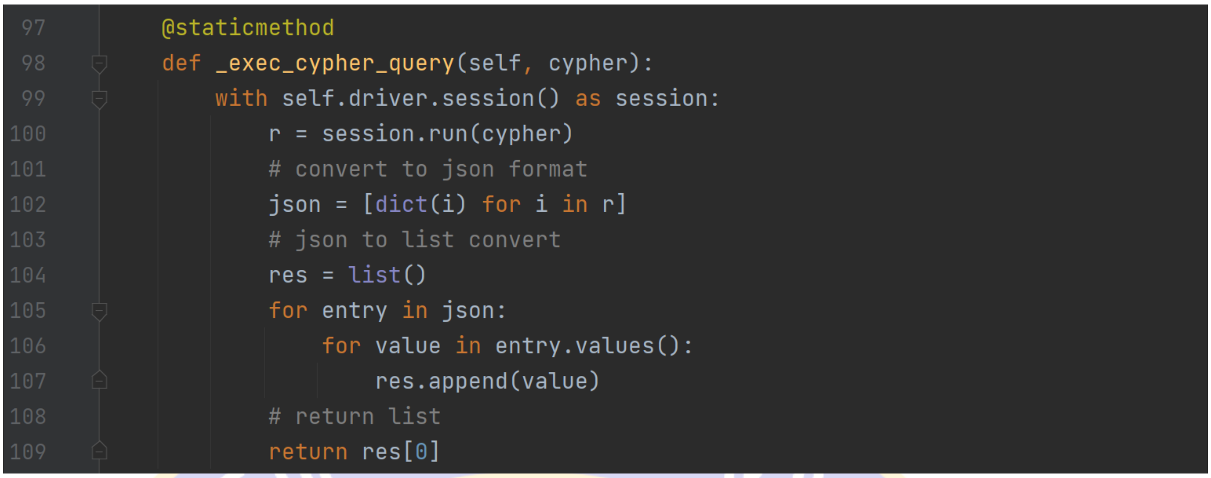


```python
    # Given two author IDs, create CO_AUTHOR relationship
    def add_co_author_relationship(self, from_author_id, to_author_id):
        cypher = "MATCH (a:Author), (b:Author) " \
                 "WHERE ID(a) = " + str(from_author_id) + " " \
                  "AND ID(b) = " + str(to_author_id) + " " \
                 "CREATE (a)-[r:CO_AUTHOR]->(b)"
        self._write_cypher_query(self, cypher)
```

12. Function that executes the given Cypher query

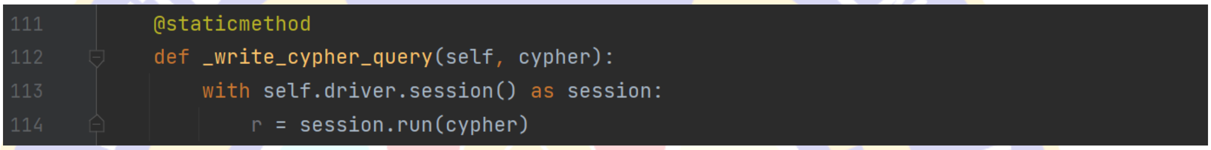
Given a cypher query, the cypher query is run using the driver class instance and a result-set is returned.

For a read-only cypher query, the result-set is then converted to a JSON format.



```python
    @staticmethod
    def _exec_cypher_query(self, cypher):
        with self.driver.session() as session:
            r = session.run(cypher)
            # convert to json format
            json = [dict(i) for i in r]
            # json to list convert
            res = list()
            for entry in json:
                for value in entry.values():
                    res.append(value)
            # return list
            return res[0]
```

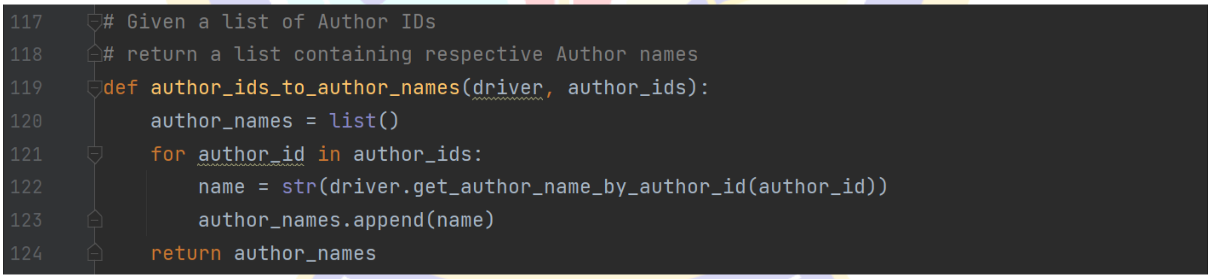
For a write query, such as to create the CO_AUTHOR relationship, the following method is needed.



```python
    @staticmethod
    def _write_cypher_query(self, cypher):
        with self.driver.session() as session:
            r = session.run(cypher)
```

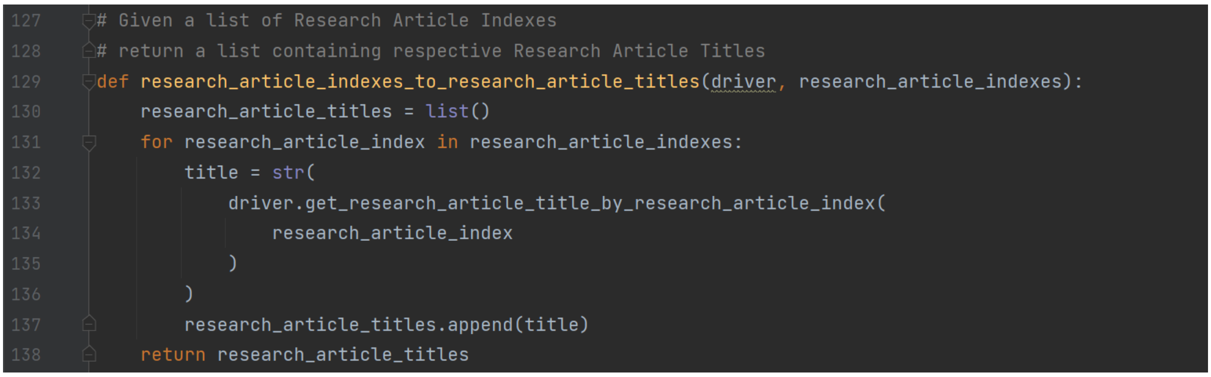
5.Python – Utility Methods

1. Given a list containing Author IDs, return a list containing the respective Author Names.



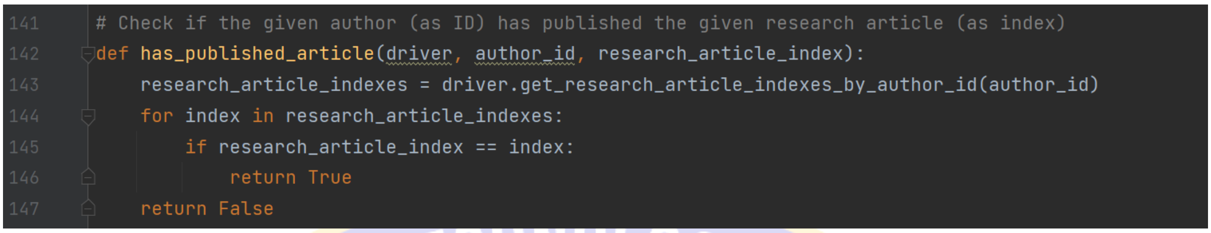
```python
# Given a list of Author IDs
# return a list containing respective Author names
def author_ids_to_author_names(driver, author_ids):
    author_names = list()
    for author_id in author_ids:
        name = str(driver.get_author_name_by_author_id(author_id))
        author_names.append(name)
    return author_names
```

2. Given a list containing Research Article indexes, return a list containing the respective Research Article Titles.



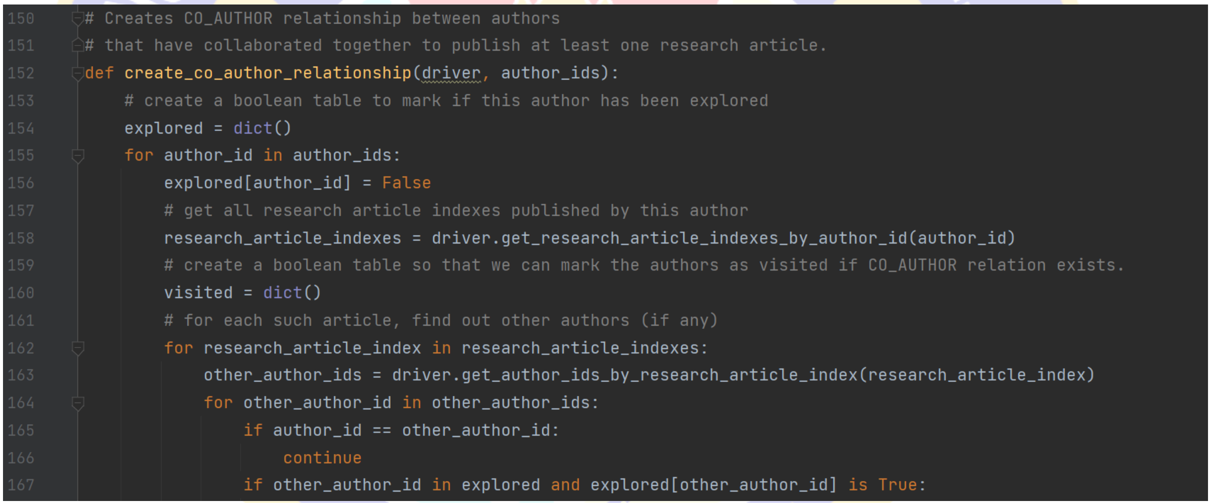
```python
# Given a list of Research Article Indexes
# return a list containing respective Research Article Titles
def research_article_indexes_to_research_article_titles(driver, research_article_indexes):
    research_article_titles = list()
    for research_article_index in research_article_indexes:
        title = str(
            driver.get_research_article_title_by_research_article_index(
                research_article_index
            )
        )
        research_article_titles.append(title)
    return research_article_titles
```

3. Given an Author ID and a Research Article Index, check whether this author has published the given research article.

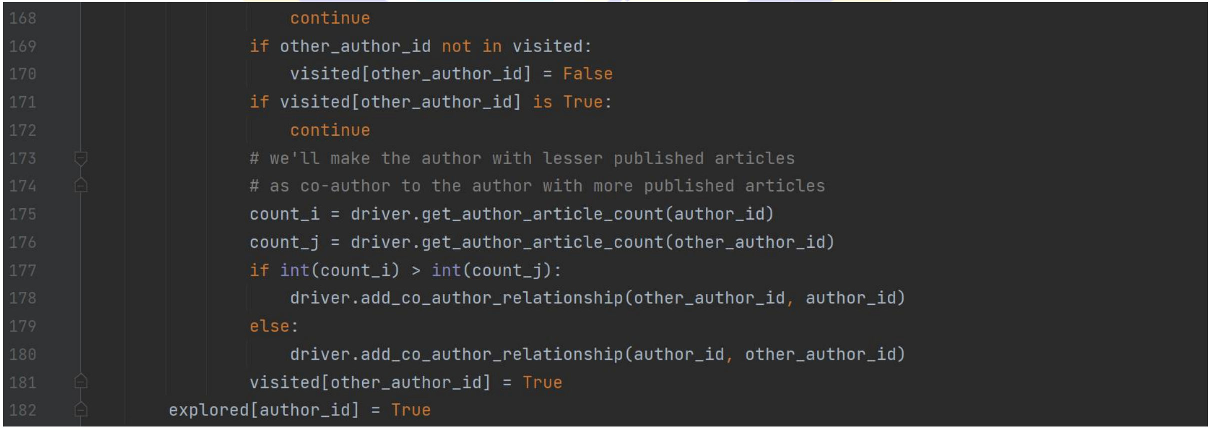


```python
# Check if the given author (as ID) has published the given research article (as index)
def has_published_article(driver, author_id, research_article_index):
    research_article_indexes = driver.get_research_article_indexes_by_author_id(author_id)
    for index in research_article_indexes:
        if research_article_index == index:
            return True
    return False
```

4. Function to create CO-AUTHOR relationship among Authors that collaborated together to publish at least one Research Article together.



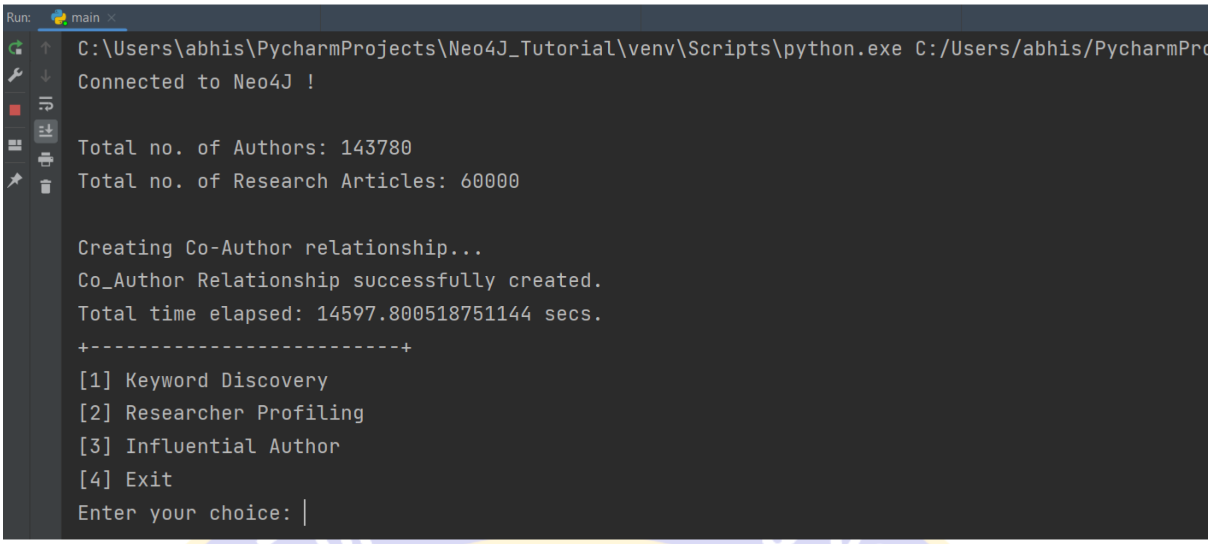
```python
# Creates CO_AUTHOR relationship between authors
# that have collaborated together to publish at least one research article.
def create_co_author_relationship(driver, author_ids):
    # create a boolean table to mark if this author has been explored
    explored = dict()
    for author_id in author_ids:
        explored[author_id] = False
        # get all research article indexes published by this author
        research_article_indexes = driver.get_research_article_indexes_by_author_id(author_id)
        # create a boolean table so that we can mark the authors as visited if CO_AUTHOR relation exists.
        visited = dict()
        # for each such article, find out other authors (if any)
        for research_article_index in research_article_indexes:
            other_author_ids = driver.get_author_ids_by_research_article_index(research_article_index)
            for other_author_id in other_author_ids:
                if author_id == other_author_id:
                    continue
                if other_author_id in explored and explored[other_author_id] is True:
```



```python
                    continue
                if other_author_id not in visited:
                    visited[other_author_id] = False
                if visited[other_author_id] is True:
                    continue
                # we'll make the author with lesser published articles
                # as co-author to the author with more published articles
                count_i = driver.get_author_article_count(author_id)
                count_j = driver.get_author_article_count(other_author_id)
                if int(count_i) > int(count_j):
                    driver.add_co_author_relationship(other_author_id, author_id)
                else:
                    driver.add_co_author_relationship(author_id, other_author_id)
                visited[other_author_id] = True
        explored[author_id] = True
```

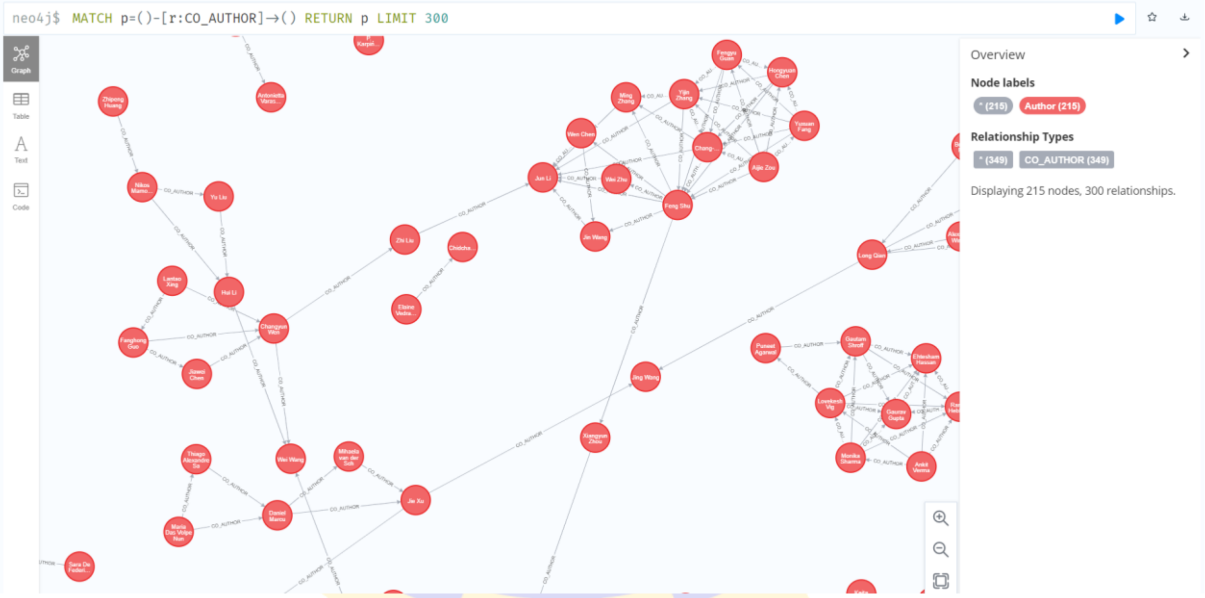
Output:

As we can see in the output snapshot below, it took almost 14600 secs i.e., around **4 hours** to create the CO_AUTHOR relationship between 143780 Authors and 60000 Articles.



```
C:\Users\abhis\PycharmProjects\Neo4J_Tutorial\venv\Scripts\python.exe C:/Users/abhis/PycharmPr
Connected to Neo4J !

Total no. of Authors: 143780
Total no. of Research Articles: 60000

Creating Co-Author relationship...
Co_Author Relationship successfully created.
Total time elapsed: 14597.800518751144 secs.
+--------------------------+
[1] Keyword Discovery
[2] Researcher Profiling
[3] Influential Author
[4] Exit
Enter your choice: |
```

Output in Neo4J Console:



```
neo4j$ MATCH p=()-[r:CO_AUTHOR]→() RETURN p LIMIT 300
```

Overview

Node labels

*(215) Author (215)

Relationship Types

*(349) CO_AUTHOR (349)

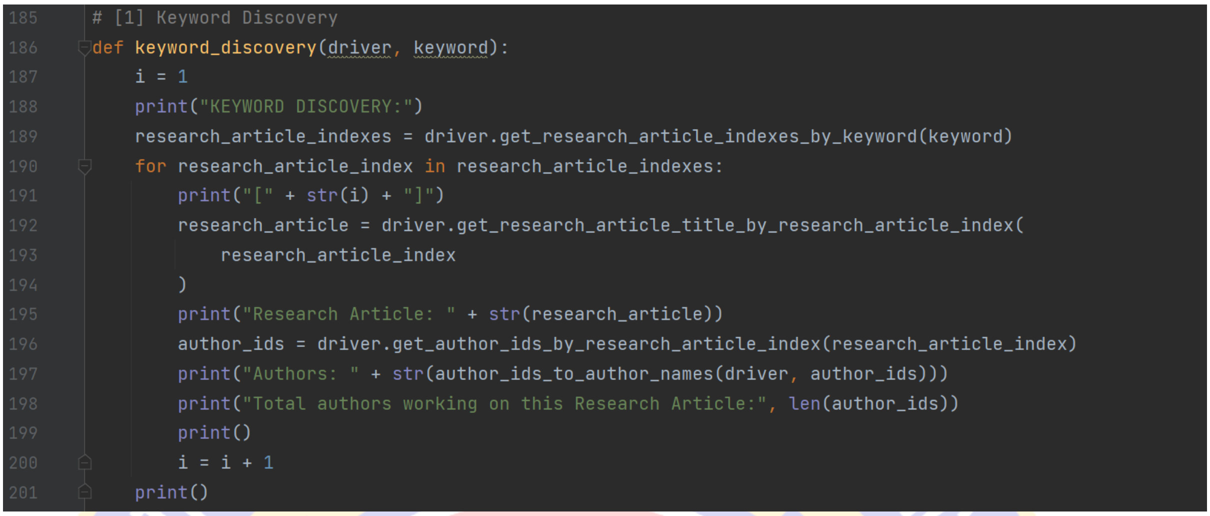
Displaying 215 nodes, 300 relationships.

6.Python – Implementing Functionalities

1. Keyword Discovery:

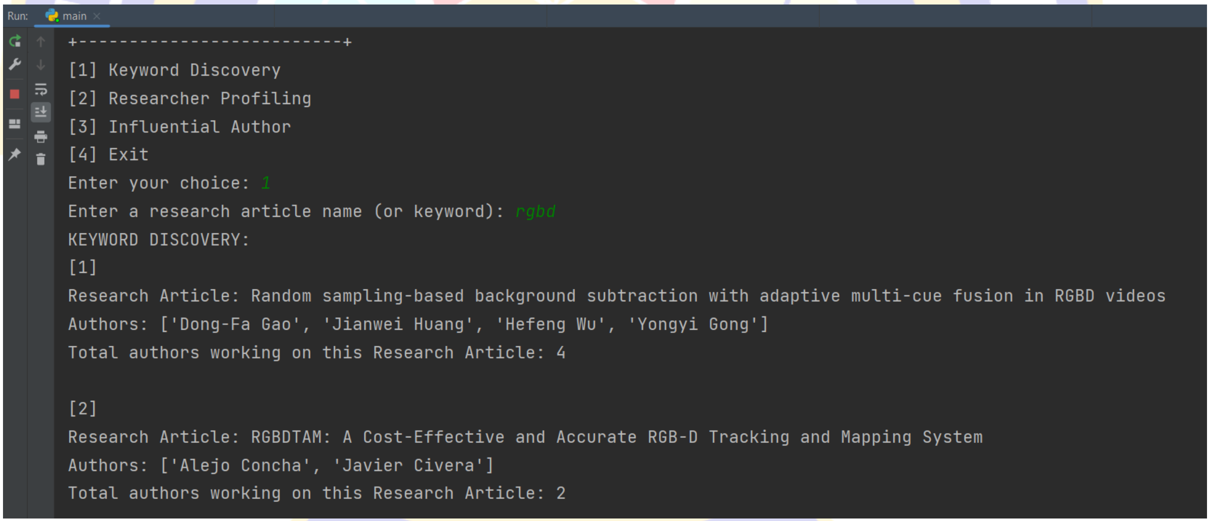
Problem: User enters any one research topic displayed on the user interface. Tool returns a list of authors working on that topic (see Table 1). The ***relevance*** estimates the prolificacy of the author within the whole DBLP community that has been working on that topic, while the ***score*** estimates the weight of that keyword among all the author's publication records. This query is useful to perform expert finding on a given research topic and similar research fields.

Code:

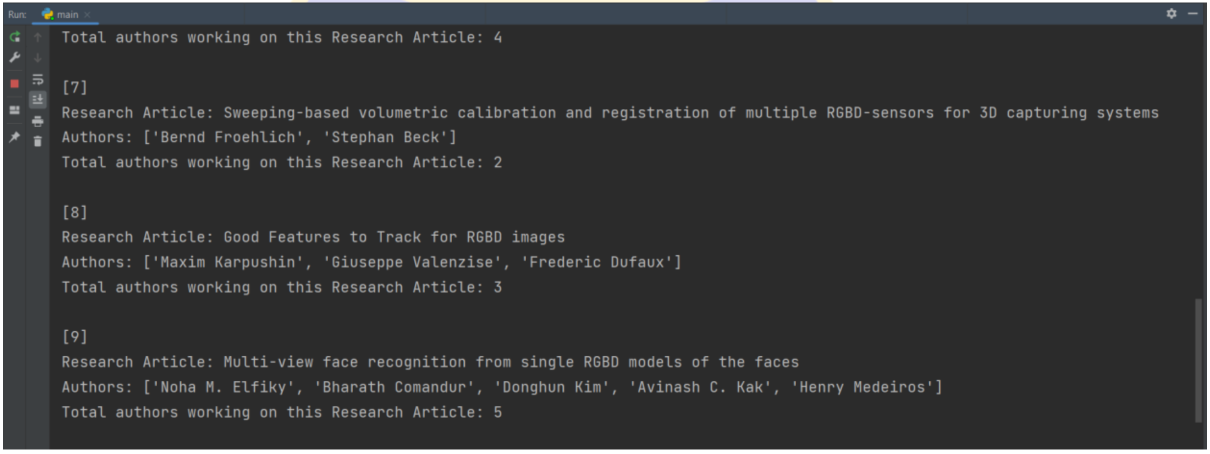


```
# [1] Keyword Discovery
def keyword_discovery(driver, keyword):
    i = 1
    print("KEYWORD DISCOVERY:")
    research_article_indexes = driver.get_research_article_indexes_by_keyword(keyword)
    for research_article_index in research_article_indexes:
        print("[" + str(i) + "]")
        research_article = driver.get_research_article_title_by_research_article_index(
            research_article_index
        )
        print("Research Article: " + str(research_article))
        author_ids = driver.get_author_ids_by_research_article_index(research_article_index)
        print("Authors: " + str(author_ids_to_author_names(driver, author_ids)))
        print("Total authors working on this Research Article:", len(author_ids))
        print()
        i = i + 1
    print()
```

Output:



```
+--------------------------+
[1] Keyword Discovery
[2] Researcher Profiling
[3] Influential Author
[4] Exit
Enter your choice: 1
Enter a research article name (or keyword): rgbd
KEYWORD DISCOVERY:
[1]
Research Article: Random sampling-based background subtraction with adaptive multi-cue fusion in RGBD videos
Authors: ['Dong-Fa Gao', 'Jianwei Huang', 'Hefeng Wu', 'Yongyi Gong']
Total authors working on this Research Article: 4

[2]
Research Article: RGBDTAM: A Cost-Effective and Accurate RGB-D Tracking and Mapping System
Authors: ['Alejo Concha', 'Javier Civera']
Total authors working on this Research Article: 2
```



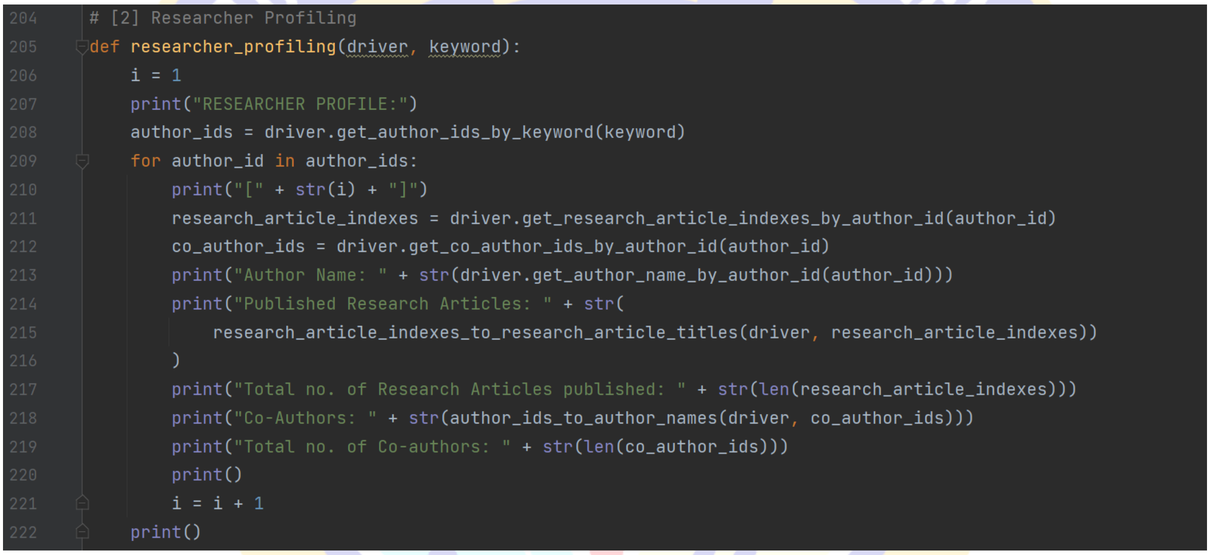
```
Total authors working on this Research Article: 4

[7]
Research Article: Sweeping-based volumetric calibration and registration of multiple RGBD-sensors for 3D capturing systems
Authors: ['Bernd Froehlich', 'Stephan Beck']
Total authors working on this Research Article: 2

[8]
Research Article: Good Features to Track for RGBD images
Authors: ['Maxim Karpushin', 'Giuseppe Valenzise', 'Frederic Dufaux']
Total authors working on this Research Article: 3

[9]
Research Article: Multi-view face recognition from single RGBD models of the faces
Authors: ['Noha M. Elfiky', 'Bharath Comandur', 'Donghun Kim', 'Avinash C. Kak', 'Henry Medeiros']
Total authors working on this Research Article: 5
```

2. Research Profiling:

Problem: User enters name of a researcher displayed on the user interface. Tool extracts all the topics on which she/he has been working along her/his career.

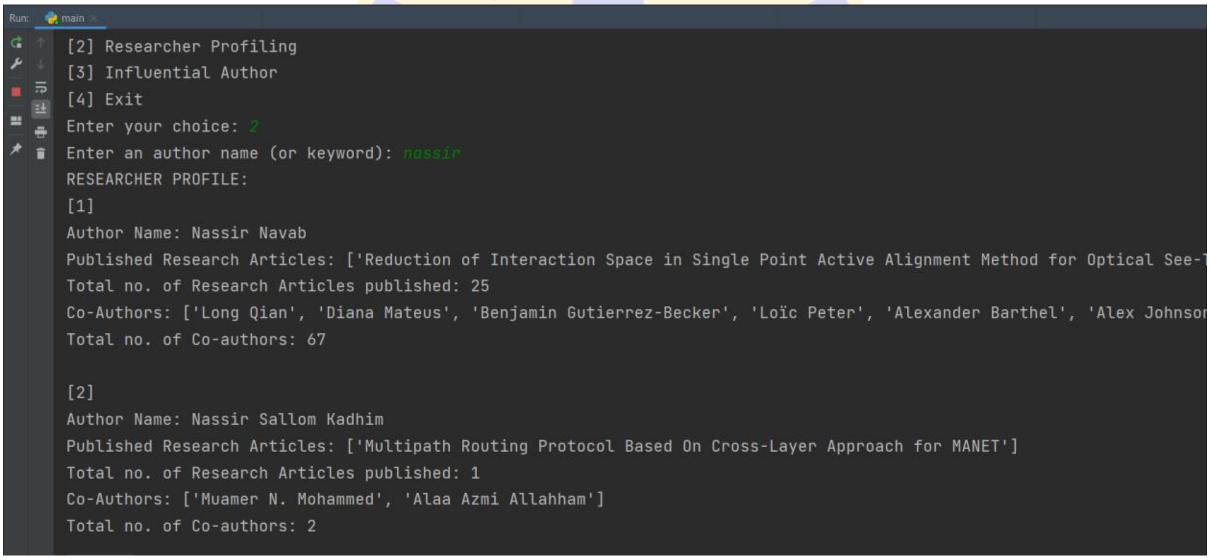
This query is useful to profile researchers, and to discover other researchers working on similar or related topics. To this end, a list of keyword similarities is returned for each topic with the similarity value.

Code:



```python
# [2] Researcher Profiling
def researcher_profiling(driver, keyword):
    i = 1
    print("RESEARCHER PROFILE:")
    author_ids = driver.get_author_ids_by_keyword(keyword)
    for author_id in author_ids:
        print("[" + str(i) + "]")
        research_article_indexes = driver.get_research_article_indexes_by_author_id(author_id)
        co_author_ids = driver.get_co_author_ids_by_author_id(author_id)
        print("Author Name: " + str(driver.get_author_name_by_author_id(author_id)))
        print("Published Research Articles: " + str(
            research_article_indexes_to_research_article_titles(driver, research_article_indexes))
        )
        print("Total no. of Research Articles published: " + str(len(research_article_indexes)))
        print("Co-Authors: " + str(author_ids_to_author_names(driver, co_author_ids)))
        print("Total no. of Co-authors: " + str(len(co_author_ids)))
        print()
        i = i + 1
    print()
```

Output:

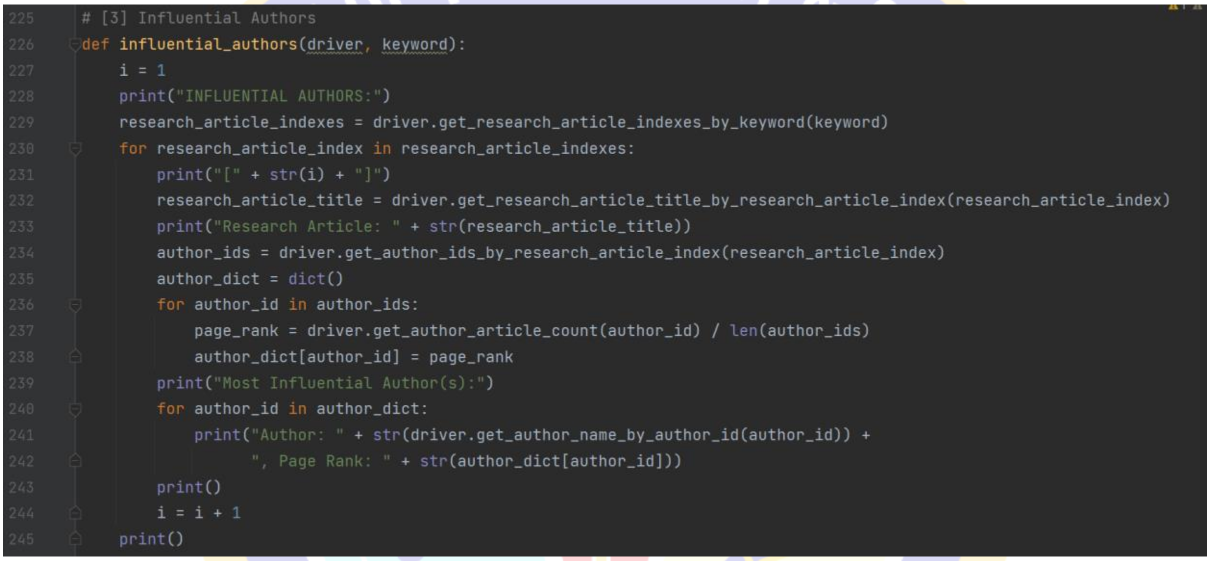


```
[2] Researcher Profiling
[3] Influential Author
[4] Exit
Enter your choice: 2
Enter an author name (or keyword): nassir
RESEARCHER PROFILE:
[1]
Author Name: Nassir Navab
Published Research Articles: ['Reduction of Interaction Space in Single Point Active Alignment Method for Optical See-
Total no. of Research Articles published: 25
Co-Authors: ['Long Qian', 'Diana Mateus', 'Benjamin Gutierrez-Becker', 'Loïc Peter', 'Alexander Barthel', 'Alex Johnso
Total no. of Co-authors: 67

[2]
Author Name: Nassir Sallom Kadhim
Published Research Articles: ['Multipath Routing Protocol Based On Cross-Layer Approach for MANET']
Total no. of Research Articles published: 1
Co-Authors: ['Muamer N. Mohammed', 'Alaa Azmi Allahham']
Total no. of Co-authors: 2
```

3. Influencing Author:

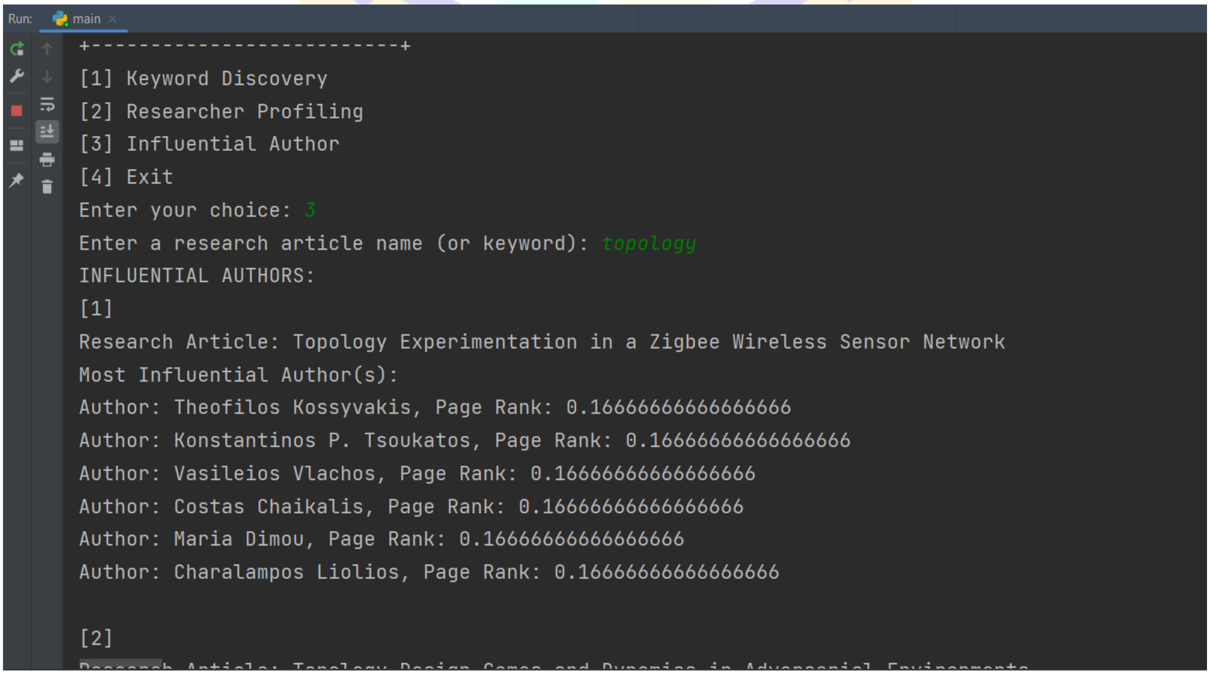
Problem: User enters research topic displayed on the user interface. Tool extracts the more influential list of authors working on that topic. The page rank algorithm computes a score that indicates the transitive influence of an author. The higher the score, authors are the more influential.

Code:

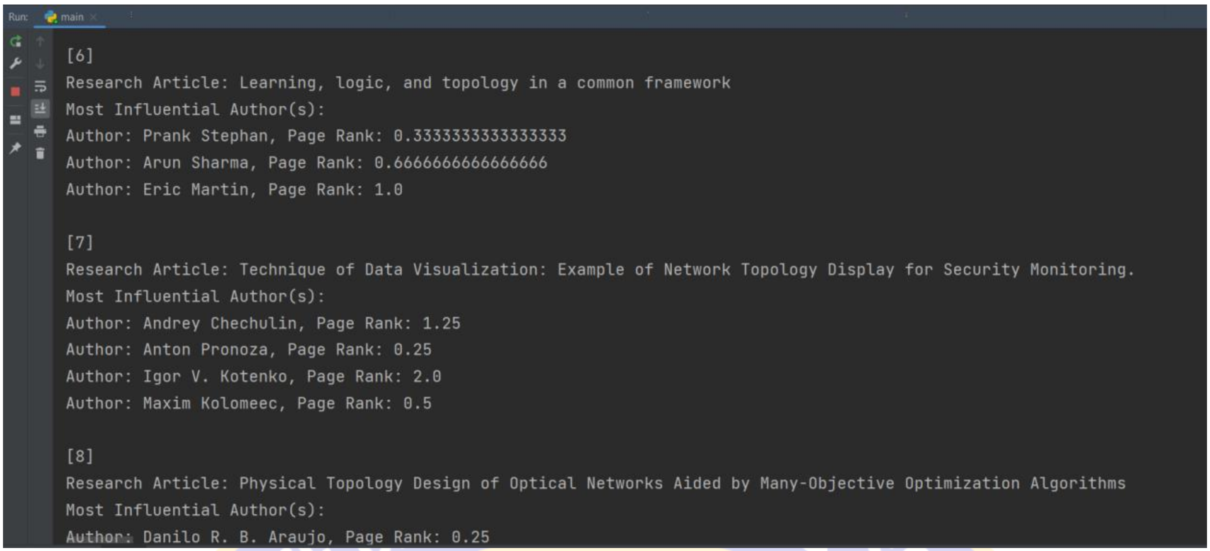


```python
# [3] Influential Authors
def influential_authors(driver, keyword):
    i = 1
    print("INFLUENTIAL AUTHORS:")
    research_article_indexes = driver.get_research_article_indexes_by_keyword(keyword)
    for research_article_index in research_article_indexes:
        print("[" + str(i) + "]")
        research_article_title = driver.get_research_article_title_by_research_article_index(research_article_index)
        print("Research Article: " + str(research_article_title))
        author_ids = driver.get_author_ids_by_research_article_index(research_article_index)
        author_dict = dict()
        for author_id in author_ids:
            page_rank = driver.get_author_article_count(author_id) / len(author_ids)
            author_dict[author_id] = page_rank
        print("Most Influential Author(s):")
        for author_id in author_dict:
            print("Author: " + str(driver.get_author_name_by_author_id(author_id)) +
                  ", Page Rank: " + str(author_dict[author_id]))
        print()
        i = i + 1
    print()
```

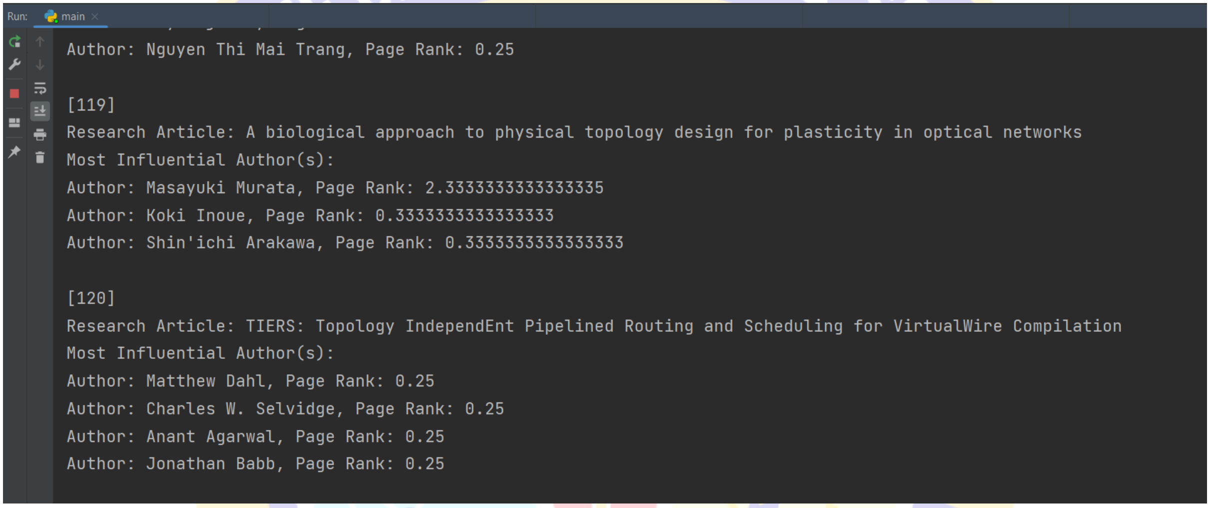
Output:



```
+-------------------------+
[1] Keyword Discovery
[2] Researcher Profiling
[3] Influential Author
[4] Exit
Enter your choice: 3
Enter a research article name (or keyword): topology
INFLUENTIAL AUTHORS:
[1]
Research Article: Topology Experimentation in a Zigbee Wireless Sensor Network
Most Influential Author(s):
Author: Theofilos Kossyvakis, Page Rank: 0.16666666666666666
Author: Konstantinos P. Tsoukatos, Page Rank: 0.16666666666666666
Author: Vasileios Vlachos, Page Rank: 0.16666666666666666
Author: Costas Chaikalis, Page Rank: 0.16666666666666666
Author: Maria Dimou, Page Rank: 0.16666666666666666
Author: Charalampos Liolios, Page Rank: 0.16666666666666666


[2]
Research Article: Topology Design Games and Dynamics in Adversarial Environments
```



```
[6]
Research Article: Learning, logic, and topology in a common framework
Most Influential Author(s):
Author: Prank Stephan, Page Rank: 0.3333333333333333
Author: Arun Sharma, Page Rank: 0.6666666666666666
Author: Eric Martin, Page Rank: 1.0

[7]
Research Article: Technique of Data Visualization: Example of Network Topology Display for Security Monitoring.
Most Influential Author(s):
Author: Andrey Chechulin, Page Rank: 1.25
Author: Anton Pronoza, Page Rank: 0.25
Author: Igor V. Kotenko, Page Rank: 2.0
Author: Maxim Kolomeec, Page Rank: 0.5

[8]
Research Article: Physical Topology Design of Optical Networks Aided by Many-Objective Optimization Algorithms
Most Influential Author(s):
Author: Danilo R. B. Araujo, Page Rank: 0.25
```



```
Author: Nguyen Thi Mai Trang, Page Rank: 0.25

[119]
Research Article: A biological approach to physical topology design for plasticity in optical networks
Most Influential Author(s):
Author: Masayuki Murata, Page Rank: 2.3333333333333335
Author: Koki Inoue, Page Rank: 0.3333333333333333
Author: Shin'ichi Arakawa, Page Rank: 0.3333333333333333

[120]
Research Article: TIERS: Topology IndependEnt Pipelined Routing and Scheduling for VirtualWire Compilation
Most Influential Author(s):
Author: Matthew Dahl, Page Rank: 0.25
Author: Charles W. Selvidge, Page Rank: 0.25
Author: Anant Agarwal, Page Rank: 0.25
Author: Jonathan Babb, Page Rank: 0.25
```

7.References

[1] YouTube: Microsoft Research "Introduction to Neo4j and Graph Databases" https://www.youtube.com/watch?v=oRtVdXvtD3o

[2] YouTube: Neo4j "Introduction to Graph Databases Series" https://www.youtube.com/watch?v=REVkXVxvMQE&list=PL9Hl4pk2FsvWM9GWaguRhlCQ-pa-ERd4U

[3] Neo4J Documentation: https://neo4j.com/docs/

[4] Cypher Query Language: https://neo4j.com/developer/cypher/