SDM Project 1 - Property Graphs

Jintao Ma, Hieu Nguyen Minh (BDMA)

April 10, 2024

A.1. Modeling

We model our graph as shown Figure 1a. We have a central node `Paper` with attributes `title, key, journal, conf, link, cite, keywords`. There are five other nodes related to `Paper`:

- `Author`: attributes `name, key`
- `Proceeding`: attributes `title, key, year, booktitle, link`
- `Book`: attribute `title`
- `Journal`: attributes `name, volume, year`
- `Keyword`: attribute `word`

Based on the description, we have the following relationship represented as edges:

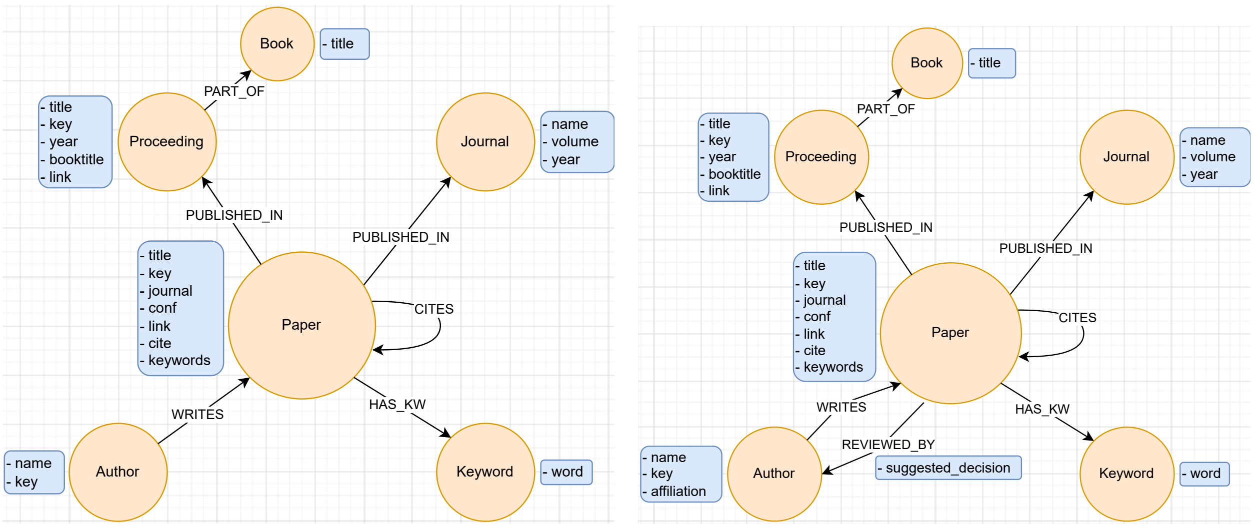
- A `Paper` is `PUBLISHED_IN` a `Journal`
- A `Paper` is `PUBLISHED_IN` a `Proceeding`
- A `Paper` `CITES` another `Paper`
- A `Paper` `HAS_KW` a `Keyword`
- A `Proceeding` is `PART_OF` a `BOOK`
- An `Author` `WRITES` a `Paper`

We argue that using five nodes and six edges above is sufficient for graph modeling in our project. The graph is able to represent all objects in the problem and their relationships. In fact, from the DBLP data that we use, the nodes `Paper` and `Proceeding` can have more attributes, such as `type, page` for `Paper` and `location, editor, isbn` for `Proceeding`. However, most of them are redundant because they are not necessary in Cypher queries in Section B, so we decided not to include them in the node. In other words, we only keep most typical attributes for nodes.

A.2. Loading

We get the DBLP[1] data from `https://dblp.uni-trier.de/xml/` in XML format (`dblp.xml.gz` and `dblp.dtd`).We downloaded about 3.92 GB xml from there.It was so large. In order to facilitate the handling of XML files, we chose to extract a portion of the subdataset from it as our experimental object.We also used this method to reduce the amount of data in the generated csv files to facilitate our subsequent conversion operations.

[1] https://dblp.org/



(a) Original graph. (b) Evolved graph.

Figure 1: Graph modeling.

Then we transferred the xml file into a csv file and extracted useful information from the whole CSV file.For example, we can just got author information directly from the origin CSV file using simple Excel operation.Because it just have two attributes: name and key.They corresponded to the two columns in the original data csv file.

We also used the tool at `https://github.com/ThomHurks/dblp-to-csv`, to convert DBLP data from XML to CSV, which allowed us to generate CSV files that were Neo4j-compatible providing some options, e.g.:

```
python XMLToCSV.py --annotate --neo4j data/dblp.xml data/dblp.dtd dblp.csv --relations author:authored by journal:published_in
```

Listing 1: Getting DBLP CSV files.

Based on the different structure and content of each document, we parsed and split them one by one and extracted useful key information from them, especially the proceeding information.

A.3. Evolving the graph

In consideration of paper reviews, we add a relationship: A `Paper` is `REVIEWED_BY` an `Author`, which has an attribute `suggested_decision`. We also add an attribute `affiliation` for node `Author`. To create suggested decision for each review, we randomly generate a real number in $[0, 1]$ as the acceptance probability. If this value is greater than 0.5, then the decision will be True (accepted), otherwise it will be False (rejected). The evolved graph is shown in Figure 1b.

B. Querying

1. Find the top 3 most cited papers of each conference

For this query, we match a relationship `(paper1:Paper)-[cites:CITES]->(paper2:Paper)`, then count the number of cites, sort the result in a descending order, and get the top three `paper2`, which are cited papers). The query result is shown in Figure 2.

```
MATCH (paper1:Paper)-[cites:CITES]->(paper2:Paper)
WITH COUNT(cites) AS cite_count, paper2 ORDER BY cite_count DESC
WITH COLLECT(paper2.title) AS cited_paper, cite_count
RETURN cited_paper AS top_3_most_cited_papers, cite_count
LIMIT 3
```

Listing 2: Cypher query 1.

```
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$ python3 PartB_Jintao_Hieu.py
Top three most cited paper:
                                             paper  number of citations
0  [The Entity-Relationship Model - Toward a Unif...                  604
1  [A Relational Model of Data for Large Shared D...                  580
2  [Access Path Selection in a Relational Databas...                  370
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$
```

Figure 2: Result of query 1.

2. For each conference, find authors who have published papers in at least 4 editions

We firstly match the following two relationships between three nodes `Author`, `Paper`, and `Proceeding`:

```
(author:Author)-[:WRITES]->(paper:Paper)-[published:PUBLISHED_IN]->(proc:Proceeding),
```

then count the `paper.key` as edition count. We filter out papers belonging to a conference with the edition count at least 4, and get the corresponding authors. The query result is shown in Figure 3.

```
MATCH (author:Author)-[:WRITES]->(paper:Paper)-[published:PUBLISHED_IN]->(proc:
    Proceeding)
WITH author, proc, COUNT(paper.key) AS edition_count
WHERE proc.key =~ 'conf.*' AND edition_count >= 4
RETURN author.name AS author_name, proc.key AS conference, edition_count
ORDER BY edition_count DESC
```

Listing 3: Cypher query 2.

```
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$ python3 PartB_Jintao_Hieu.py
Authors who have published in at least 4 editions:
                   author      conference  number of editions
0    Hector Garcia-Molina  conf/sigmod/97                   6
1    Hector Garcia-Molina  conf/sigmod/95                   4
2   Laks V. S. Lakshmanan  conf/sigmod/99                   4
3   Michael J. Carey 0001  conf/sigmod/97                   4
4          H. V. Jagadish    conf/vldb/99                   4
5          H. V. Jagadish    conf/vldb/92                   4
6    Hector Garcia-Molina    conf/vldb/98                   4
7       Raghu Ramakrishnan   conf/pods/87                   4
8   Michael J. Carey 0001  conf/sigmod/94                   4
9          Rajeev Rastogi    conf/icde/99                   4
10          Peter P. Chen      conf/er/81                   4
11          Clement T. Yu    conf/icde/95                   4
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$
```

Figure 3: Result of query 2.

3. Find the impact factor of the journals

The journal impact factor for a given year is the ratio between the number of citations received in that year for publications in that journal that were published in the two preceding years and the total number of "citable items" published in that journal during the two preceding years[2].

$$\mathrm{IF}_y = \frac{\mathrm{Citation}_y}{\mathrm{Publication}_{y-1} + \mathrm{Publication}_{y-2}}. \tag{1}$$

This value represents the average number of citations that papers published in years $y-1$ and $y-2$ received in year y. For this query we compute the impact factor for the year 2000:

$$\mathrm{IF}_{2000} = \frac{\mathrm{Citation}_y}{\mathrm{Publication}_{1999} + \mathrm{Publication}_{1998}}. \tag{2}$$

We firstly match a relationship `(paper:Paper)-[:PUBLISHED_IN]->(journal:Journal)` with a condition of journal published in 1999 or 1998. Then we get the journal title, the papers and their total number. Next, we match a relationship `(paper1:Paper)-[cites:CITES]->(paper2:Paper)`, where paper1 is in the journal found in previous step. We count the number of `cites` and divide it by the total number of papers to get the impact factor. The query result is shown in Figure 4.

```
MATCH (paper:Paper)-[:PUBLISHED_IN]->(journal:Journal)
WHERE toInteger(journal.year)=1999 OR toInteger(journal.year)=1998
WITH journal.name as journal_title, size(COLLECT(paper)) AS n_papers, COLLECT(
    paper) AS papers_in_journal
MATCH(paper1:Paper)-[cites:CITES]->(paper2:Paper)
WHERE paper1 IN papers_in_journal
RETURN journal_title, (toFloat(COUNT(cites))/n_papers) AS impact_factor
ORDER BY impact_factor DESC
```

Listing 4: Cypher query 3.

```
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$ python3 PartB_Jintao_Hieu.py
Impact factor of journals:
                         journal  impact factor
0  IEEE Trans. Knowl. Data Eng.           1233
1                   SIGMOD Rec.            825
2                       VLDB J.            726
3     ACM Trans. Database Syst.            552
4          IEEE Data Eng. Bull.            355
5             ACM Comput. Surv.            230
6        ACM SIGMOD Digit. Rev.             85
7                   Commun. ACM             23
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$
```

Figure 4: Result of query 3.

4. Find the h-index of the authors

The h-index is defined as the maximum value of h such that the author has published at least h papers that have each been cited at least h times[3]. Formally, assuming that f is the function corresponding to the number of citations for each paper in descending order of values, then:

$$h\text{-index} = \max\{i \in N : f(i) \geq i\}. \tag{3}$$

We firstly match two relationships between a node `Author` and two other nodes `Paper`:

`(author:Author)-[writes:WRITES]->(paper1:Paper)-[cites:CITES]->(paper2:Paper).`

[2]https://en.wikipedia.org/wiki/Impact_factor
[3]https://en.wikipedia.org/wiki/H-index

Then we create an enumeration of `paper2` (cited papers), count the number of `paper2`s and get h-index as the position in the enumeration where number of `paper2`s is at least that position value. The query result is shown in Figure 5

```
MATCH (author:Author)-[writes:WRITES]->(paper1:Paper)-[cites:CITES]->(paper2:Paper
    )
WITH author, paper2, COLLECT([id(paper1), paper1.title]) AS rows
WITH author, paper2, RANGE(1, SIZE(rows)) AS enum_rows
    UNWIND enum_rows AS erow
WITH author, erow AS pos, COUNT(paper2) as n_cited
WHERE n_cited >= pos
RETURN author.name as author_name, pos as h_index
ORDER BY h_index DESC
```

Listing 5: Cypher query 4.

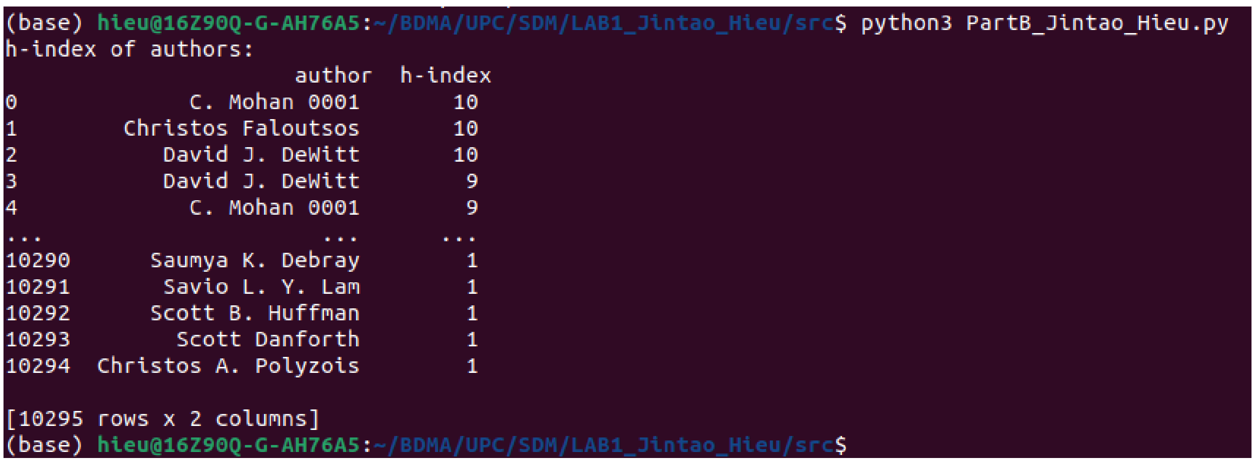


Figure 5: Result of query 4.

C. Recommender

1. The community

The first step is to find the database community, which is defined by the set of keywords: data management, indexing, data modeling, big data, data processing, data storage and data querying. For a book, if 90% of its papers contain one of the above keywords then it is considered to belong to that community. In the Cypher query, to find the papers having given keywords in a book, we match the following relationships:

```
(keyword:Keyword)<-[:HAS_KW]-(paper_has_kw:Paper)-[:PUBLISHED_IN]->
                (proceeding)-[:PART_OF]->(book:Book),
```

where `keyword` is in the given keyword set. For each book, we count the number of papers having any of the given keywords. After that, to find all papers in a book, we match the following relationships:

```
(book)<-[:PART_OF]-(proceeding)<-[:PUBLISHED_IN]-(paper_in_book:Paper),
```

and count the number of papers in a book. Next, we calculate the keyword percentage to find the community. For a book, we define the keyword percentage as the ratio of the number of papers having any of given keywords over the total number of papers. A book is considered to belong to a community if the keyword percentage is at least 0.9 (i.e. at least 90% of the papers has community keywords).

```
MATCH (keyword:Keyword)<-[:HAS_KW]-(paper_has_kw:Paper)-[:PUBLISHED_IN]->(
    proceeding)-[:PART_OF]->(book:Book)
WHERE keyword.word in ['data management', 'indexing', 'data modeling', 'big data',
     'data processing', 'data storage', 'data querying']
WITH book, COUNT(DISTINCT paper_has_kw) AS n_papers_has_kw
MATCH (book)<-[:PART_OF]-(proceeding)<-[:PUBLISHED_IN]-(paper_in_book:Paper)
WITH book, n_papers_has_kw, COUNT(paper_in_book) as n_papers_in_book
WITH book, n_papers_has_kw, n_papers_in_book, (tofloat(n_papers_kw)/
    n_papers_in_book) as kw_percent
WHERE kw_percent >= 0.9
WITH book.title as book_title, n_papers_has_kw, n_papers_in_book, kw_percent
ORDer BY kw_percent DESC
RETURN COLlECT (book_title)
```

Listing 6: Cypher query for defining the database community.

2. Top-100 papers

To find top-100 papers, firstly we match and count the number of citations for each paper. Then we find the book of that community by matching . Finally we sort the paper in descending order of number of citations and get the first 100 papers.

```
MATCH (paper1:Paper)-[cites:CITES]->(paper2:Paper)
WITH COUNT(cites) AS cite_count, paper2 ORDER BY cite_count DESC
WITH COLLECT(paper2.title) AS cited_paper, cite_count
MATCH (book:Book)<-[:inBook]-(:Proceeding)<-[:published]-(paper)
WHERE book.title IN '''+str(community)+'''
WITH paper2.title as top_paper, book.title as book_title, cite_count ORDER BY
    cite_count DESC
RETURN COLLECT(top_paper)
LIMIT 100
```

Listing 7: Cypher query for getting top-100 papers.

3. The gurus

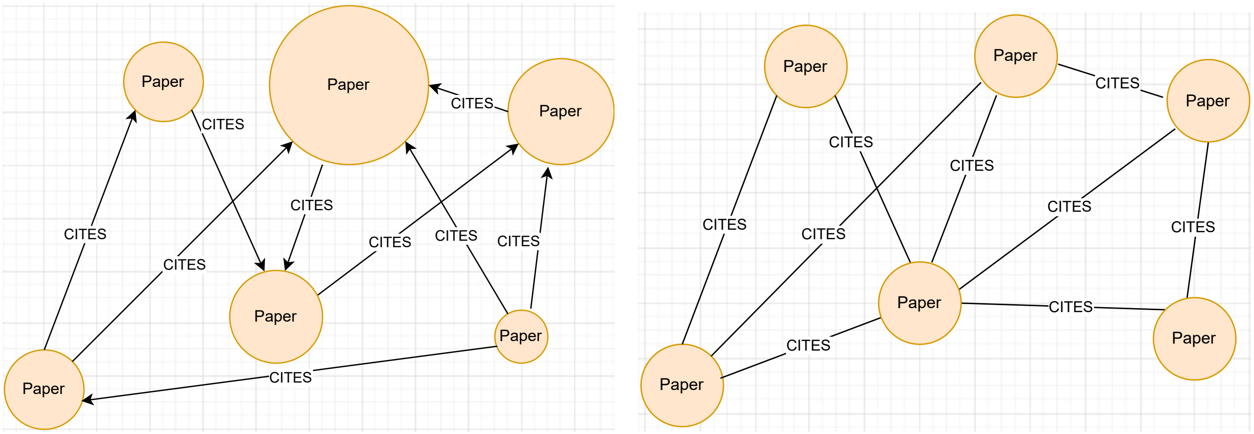
Gurus are identified as authors of at least two papers among the top-100 papers in a community. Therefore, to find gurus we match a relationship `(paper:Paper)<-[:WRITES]-(author:Author)`, where `paper` is in the top-100 papers. We then count the number of papers belonging to every author, and get the gurus as ones having that count value at least 2.

```
MATCH (paper:Paper)<-[:WRITES]-(author:Author)
WHERE paper.title IN '''+str(top_100_papers)+'''
WITH author.name as author_name, COUNT(paper) AS paper_count
WHERE paper_count >= 2
RETURN COLLECT(author_name)
```

Listing 8: Cypher query for finding gurus.

D. Graph algorithms

We choose two graph algorithms: PageRank and Triangle Count. We installed the library Graph Data Science (GDS)[4] from Neo4j.



(a) PageRank graph. (b) Triangle Count graph.

Figure 6: Illustration of graph algorithms.

1. PageRank

In a graph, PageRank algorithm measures the importance of each node based on the number incoming relationships and the importance of the corresponding source nodes[5]. The PageRank graph of papers is illustrated in Figure 6a. The bigger the circle is, the more citations that paper receives, and the higher score it receives from the algorithm.

To implement the algorithm, firstly we create a graph of papers:

```
CALL gds.graph.project('graph1', 'Paper', 'CITES')
```

Listing 9: GDS call for creating a graph of papers.

And we call the PageRank algorithm from GDS. The result is shown in Figure 7.

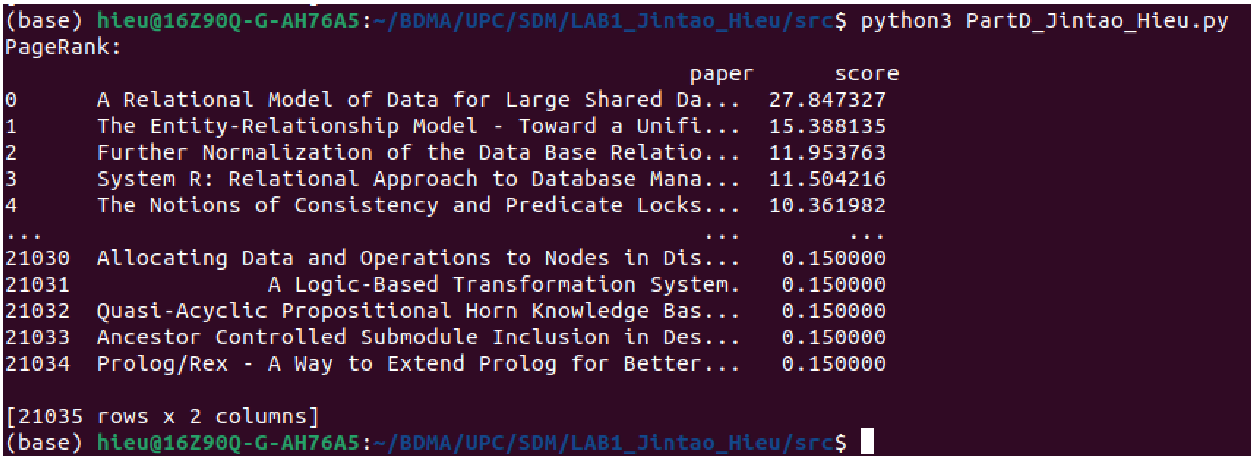
```
CALL gds.pageRank.stream('graph1')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).title AS paper, score
ORDER BY score DESC
```

Listing 10: GDS call of PageRank algorithm.

2. Triangle Count

In a graph, Triangle Count algorithm counts the number of triangles for each node. A triangle is a set of three nodes where each node has a relationship to the other two. The Triangle Count

[4]https://neo4j.com/docs/graph-data-science/current/
[5]https://neo4j.com/docs/graph-data-science/current/algorithms/page-rank/



```
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$ python3 PartD_Jintao_Hieu.py
PageRank:
                                                  paper      score
0      A Relational Model of Data for Large Shared Da...  27.847327
1      The Entity-Relationship Model - Toward a Unifi...  15.388135
2      Further Normalization of the Data Base Relatio...  11.953763
3      System R: Relational Approach to Database Mana...  11.504216
4      The Notions of Consistency and Predicate Locks...  10.361982
...                                                 ...        ...
21030  Allocating Data and Operations to Nodes in Dis...   0.150000
21031               A Logic-Based Transformation System.   0.150000
21032  Quasi-Acyclic Propositional Horn Knowledge Bas...   0.150000
21033  Ancestor Controlled Submodule Inclusion in Des...   0.150000
21034  Prolog/Rex - A Way to Extend Prolog for Better...   0.150000

[21035 rows x 2 columns]
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$
```

Figure 7: Result of PageRank algorithm.

algorithm in the GDS library only finds triangles in undirected graphs.[6]. The Triangle Count graph of papers is illustrated in Figure 6b. For example, the center circle forms four triangles with its neighbor nodes.

Like PageRank, we also create a graph of papers, but this graph is *undirected*:

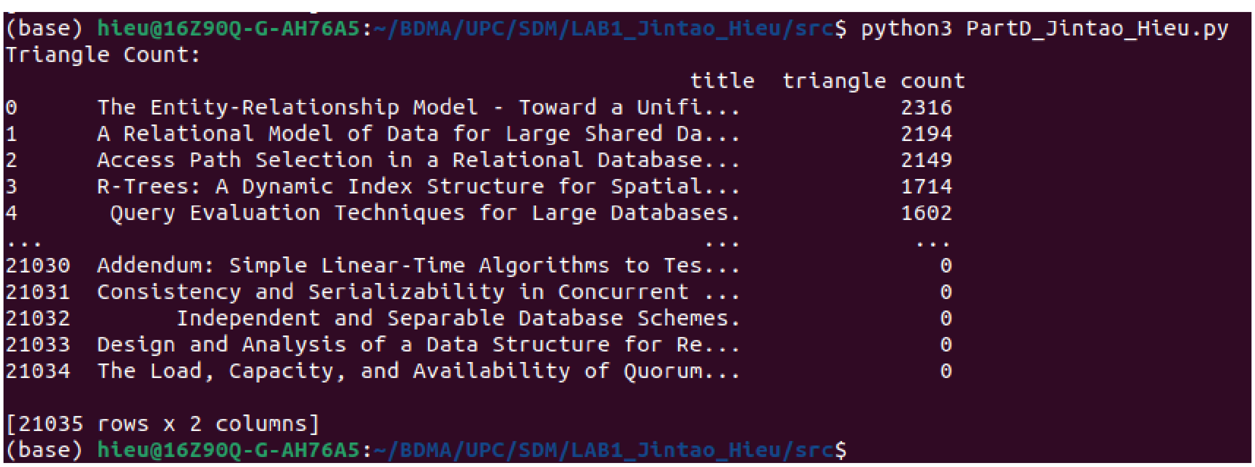
```
CALL gds.graph.project('graph2', 'Paper', {CITES: {orientation: 'UNDIRECTED'}})
```

Listing 11: GDS call for creating an undirected graph of papers.

And we call the Triangle Count algorithm from GDS. The result is shown in Figure 8.

```
CALL gds.triangleCount.stream('graph2')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).title AS title, triangleCount
ORDER BY triangleCount DESC
```

Listing 12: GDS call of Triangle Count algorithm.



```
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$ python3 PartD_Jintao_Hieu.py
Triangle Count:
                                                  title  triangle count
0      The Entity-Relationship Model - Toward a Unifi...            2316
1      A Relational Model of Data for Large Shared Da...            2194
2      Access Path Selection in a Relational Database...            2149
3      R-Trees: A Dynamic Index Structure for Spatial...            1714
4       Query Evaluation Techniques for Large Databases.            1602
...                                                 ...             ...
21030  Addendum: Simple Linear-Time Algorithms to Tes...               0
21031  Consistency and Serializability in Concurrent ...               0
21032        Independent and Separable Database Schemes.               0
21033  Design and Analysis of a Data Structure for Re...               0
21034  The Load, Capacity, and Availability of Quorum...               0

[21035 rows x 2 columns]
(base) hieu@16Z90Q-G-AH76A5:~/BDMA/UPC/SDM/LAB1_Jintao_Hieu/src$
```

Figure 8: Result of Triangle Count algorithm.

[6]https://neo4j.com/docs/graph-data-science/current/algorithms/triangle-count/