2022 IEEE 22nd International Symposium on Computational Intelligence and Informatics and 8th IEEE International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics (CINTI-MACRo) | 979-8-3503-9882-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/CINTI-MACRo57952.2022.10029438

An Automated Graph Construction Approach from Relational Databases to Neo4j

1st I Made Putrama
Department of Electronics Technology,
Faculty of Electrical Engineering and Informatics,
Budapest University of Technology and Economics,
Műegyetem rkp. 3., H-1111 Budapest, Hungary
putrama.imade@edu.bme.hu

2nd Péter Martinek
Department of Electronics Technology,
Faculty of Electrical Engineering and Informatics,
Budapest University of Technology and Economics,
Műegyetem rkp. 3., H-1111 Budapest, Hungary
martinek.peter@vik.bme.hu

***Abstract*—There are still few research methods proposed to convert relational databases to graph databases. Although a graph database has been equipped with a scripting language to use for querying and converting the data, it still requires time-consuming efforts by the domain expert to analyze the various constraints present in the source database. This paper proposes a novel technique to help automate the conversion by extracting relational database metadata and then sorting the entity relationships before converting them into graphs. To validate the conversion results, the total number of records in the source database with the number of nodes and edges created in the graph database are compared, and the node properties are validated for consistency using a probabilistic data structure. Based on our test results, their completeness can be checked accurately and efficiently with test parameters that can be adjusted according to the size of the source database.**

***Index Terms*—Automate, Relational, Construction, Graph, Database, Neo4j**

I. Introduction

In recent years, the use of Graph databases in the community, practitioners, and enterprises is increasingly popular because of its advantages, especially in terms of flexibility, suitable for semi- and unstructured format, and its ability to process large amounts of data with high performance compared to relational databases. In today's era of big data, data has the characteristics of a large volume in various formats, but the information it contains is fragmented so that it is not easy to filter or transform it to get the information needed. The use of relational databases that have become mainstream for a long time, such as: MySQL, Oracle, MSSQL Server, and so on, has a shortcoming in Big Data processing which is also accompanied by the need for intensive JavaScript Object Notation (JSON) data processing [1]. The use of normalized join operations of multiple tables in complex queries to find very large data makes this process inefficient as it has to examine every related data of the many rows of the various tables involved. On the other hand, the use of graph databases is excellent in terms of performance when the search is carried out on large data especially when the data is naturally related to each other such as user data on social media that form network patterns. The core of a Graph database is a graph model consisting of a set of edges and nodes. Each node represents the data or location while each edge corresponds to the relationship between two nodes. Nodes and edges both can hold a list of attributes called properties in the form of key-value pairs [2]. With this structure, the graph database can process queries for a node and its relationships by tracing contiguous paths without the need to scan the entire data, and this makes the graph database much more efficient. Thus, depending on the business requirement enterprises may need for converting their existing relational databases into graph database to anticipate data growth and be able to take advantage of various big data processing technologies.

One of the most widely used graph databases is Neo4j. It has been ranked as one of the top 10 most popular graph database management systems along with others such as ArangoDB, Amazon Neptune, OrientDB, Dgraph, Cassandra, and many more. We also found active studies using Neo4j as the underlying graph database in research which involve graph construction from relational databases or datasets. However, in general none has discussed or proposed an automated conversion approach that can be used to some degree depending on the complexity of the requirements. Neo4j, like relational databases in general, has a declarative query language, similar to Structured Query Language (SQL), called Cyper. It is optimized for graphs and can be used to create nodes and relationships, as well as to perform query on the graph. Thus, performing a graph construction from relational data in an automate fashion will save a lot of time.

In this paper, we propose a novel technique to perform conversion of a relational database into a graph database. Initially, we need to extract the Data Definition Language (DDL) from the existing relational database. Based on the relational dependency of the tables contained in the schema, we use a modified topological sorting algorithm to sort the schema to ensure the cyclic dependencies are resolved if they exist before running a Cyper query to create nodes and edges in Neo4J. We also show how to validate the created graph by verifying the schema and relationship for completeness and perform data consistency checking by using Bloom Filter.

II. Related Works

There are several literatures discussing how to construct a knowledge graph in Neo4J based on relational data input. Such as one done by [3], which creates a graph database by importing it from a Comma Separated Value (CSV) file. The study explains how nodes are formed and their relationships, but the input file is in csv format, which contains information about the data of the author of the publication, and the data does not come from a relational database. This paper also recommends the use of an automated approach in the construction of graph databases but does not discuss in detail the automated method. Researchers have attempted to construct knowledge graph based on relational data in various domains. However, they are mainly based on CSV files, Text file, or data extracted from websites and do not specifically address data relationships as exist in a relational database [1], [2], [4]–[8]. A study done recently by [2] that specifically addresses problems in the conversion of traditional relational databases into graph databases. Its main aims are to sort out relationship from metadata schema, the completeness of the tables and their relationships and data constrains during the conversion and solving data problems after the conversion. However, in its approach, the conversion for a MySQL relational database is done by first creating an Entity Relationship Diagram (ERD) and then examining the relationship before the migration can be done using the built-in Cypher Query Language (CQL) function of Neo4J. We argue that the process of graph creation from relational database can be done by exporting the schema metadata and perform the extraction of table relationships automatically. In some situation, a cyclic dependency can be found in a schema so the creation must first solve the dependency problem before the graph creation in Neo4J.

III. Graph Construction

A. Relational Database Schema Overview

In this section, we show an example of a MySQL database schema that will be converted into a graph using the Neo4j graph database. The schema is called *'Sakila'*, and its structure consists of 15 tables with each of them has at least one relationship from one another except one table called *film_text*. From these relationships, there are two identifying M:N relationships: (*actor* → *film_actor* → *film*), and (*film* → *film_category* → *category*), as depicted in Figure 1, and several 1:N and M:N non-identifying relationships as in Figure 2

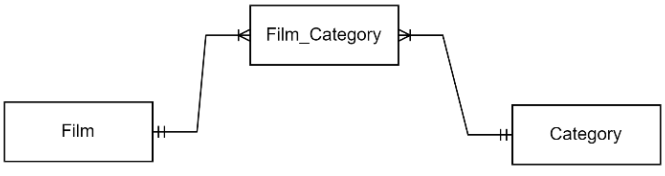

Fig. 1. Tables with identifying relationship

To perform the conversion successfully, especially to process the relationship between nodes, the topology structure should ideally have no cyclic dependencies.

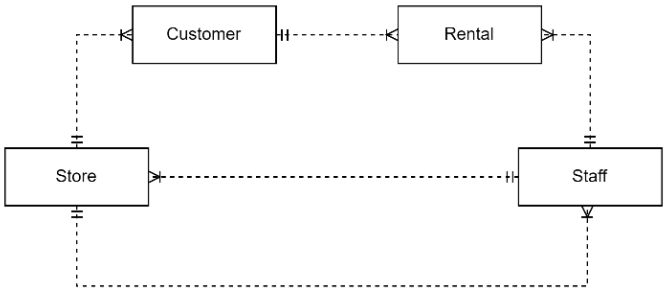

Fig. 2. Tables with non-identifying relationship and a cyclic dependency

B. Relational Database Metadata Extraction

Each table structure in a relational database schema has a standard DDL script that is specific to the database type that is used (e.g. DDL script in MySQL is different with DDL script used by other databases such as Postgres, MSSQL Server, Oracle, etc.) that can be generated based on the existing tables. Using a predefined parser, the script can be extracted for the purposes of graph construction to be performed based on the data in the table and its relationship to other tables in the schema. One thing to note is that not every database engine provides a mechanism to generate DDL by issuing an SQL statement, such as in MSSQL Server. In order to create the statement, one has to export it manually if the database engine does not support such feature. In this case, the parser can process the resulting DDL by reading it from a given input file.

Some of the important information that can be extracted depending on the algorithm. In our approach, it includes: *table name*, any *primary key columns*, *foreign key columns* if any and the corresponding list of the *foreign key tables*. The parser can be in the form of regex written in the programming language, such as Python as used to perform the graph construction in this study. The regex for the metadata extraction can be as given in Table I.

C. Handling Cyclic Dependencies

Cyclic Dependency a.k.a. circular dependency is a relationship between two or more tables in a relational schema on which one table is directly or indirectly dependent on the other, and vice versa. In the 'Sakila' database for example, there is one such relationship between *store* table and *staff* table. The *store* table has a foreign key called *manager_staf_id* which refers to the *staff_id* in the *staff* table, which at the same time the *staff* table also has a foreign key called *store_id* which refers to the same *store* table.

A cyclic dependency in a software system is considered as an architectural problem. The same issue should be avoided in any database design wherever possible so that an automated processing will be easier [9]. Even though there are no enforcement to have integrity constraints in graph database as in relational databases, it is still important to ensure edges among nodes are created based on the exact relationships as exist in

TABLE I
METADATA EXTRACTION IN MYSQL AND POSTGRES USING REGEX IN PYTHON

Rdbms	Function	Regex
MySQL	To extract the Primary tables	`'PRIMARY KEY\s\S(.*)(?=\))'`
	To extract the Foreign keys	`'FOREIGNKEY.*?\(`(.*) .*?`\)\sREFERENCES'`
	To extract the reference tables	`'REFERENCES.*?`(.*).*?`\s'`
	To extract the reference keys	`'REFERENCES.*?\(`(.*).*?`\)'`
Postgres	To extract the Primary Keys	`'PRIMARY KEY\s\S(.*?)\)'`
	To extract the Foreign Keys	`'FOREIGN KEY\s\S(.*?)\)'`
	To extract the Reference Tables	`'REFERENCES.*?\.(.*?)\s\('`
	To extract the Reference Keys	`'REFERENCES.*?\((.*?)\)'`

the original relational databases. Unfortunately, at the moment of this writing, such constraint property creation feature is only available in the Enterprise Edition of the Neo4j. Therefore, in a manual graph database generation, an expert involvement will be required to enforce the constraint definitions on the created graph or to determine whether any cyclic dependencies found in the relational database should be temporarily removed before the conversion can be performed. To overcome this problem, in this study we employ a topological sorting algorithm to detect such constraints by temporarily skipping them during the initial node creation and linked at the end once all of the dependent nodes are formed.

To ensure the data integrity is in place during nodes and edges creation for the corresponding tables and relations in a relational database, we perform topological sorting against the tables according to their hierarchical dependencies. For example, nodes for *film* and *category* tables such as shown in Figure 1, must first be created prior to the node creation for the *film_category* table. To perform ordering, our approach modifies the Kahn's topological sorting algorithm as given in Algorithm 1. The algorithm takes additional steps by examining the smallest ratio between the outgoing and the incoming relation to solve a possibility of cyclic dependencies between a table and its corresponding referenced tables. Then, it populates an additional list of edges for the cyclic dependency if it is encountered during the graph generation.

D. Nodes and Edges Creation using Cypher

Before proceeding with the graph database creation, the data from the relational database has to be extracted first. This process can be done by exporting the data into CSV file from the relational database such as by using MySQL Workbench or by using SQL export command.

Algorithm 1 Sort metadata

Require: S, the set of all nodes with incoming and outgoing edge
Initialize:
  $L \leftarrow$ Empty list that will contain the sorted elements
  $E \leftarrow$ Empty list that will contain cyclic edges
while $S \neq \emptyset$ **do**
  Calculate the ratio of outgoing and incoming edges of each node
  Get the first node N with the smallest r ratio
  {When the node has outgoing & incoming}
  if $r > 0$ **then**
    for all node m with an edge e from m to n **do**
      Add e to E
      Remove edge e from the graph
    end for
  end if
  {When the node has only incoming}
  for all node m with an edge e from n to m **do**
    Remove edge e from the graph
  end for
  Remove node n from S
  Add n to L
end while
return L (a topologically sorted order) and E (a list of cyclic edges)

To create the nodes and edges for the graph in Neo4j, we can continue to perform iteration based on the sorted list of metadata done from the Algorithm 1. Several first entries will have no dependencies and the creation can be done by using the Cypher script. For entries with dependencies, apart from creating new nodes, we have to create additional edges depending on the list of referenced tables.

E. Graph Construction Algorithm

To construct the graph database in Neo4J, we follow Algorithm 2 to process the sorted metadata information to create the nodes and relationships based on Cypher queries. This algorithm requires L, which contains sorted metadata and E, which contains cyclic dependencies (if any). This will initially loop through the metadata and create a node that has no dependencies to any other nodes. Next, it renders the rest of the nodes as well as their edges according to the nodes they depend on. Finally, additional edges will be created for nodes that have cyclic dependencies as listed in E.

Algorithm 2 Construct Graph

Require: L (the sorted metadata), E (cyclic relation)
  { Both L and E are maps which entry is a pair of $p \rightarrow q$ where p is the main table in the relation, and q contains 0 or more reference tables }
for all l_i in L **do**
  $p =$ Create node for main table in l_i
  $r_i =$ Foreign tables in l_i
  {Since L has been sorted, if r_i is not empty, the reference node is available}
  for all r_i **do**
    $q =$ Get existing node for reference table in r_i
    Create outgoing edge from node p to node q
  end for
end for
{When E is not empty, both main and reference nodes are available}
for all e_i in E **do**
  Get existing node p and q based on relation e_i
  Create outgoing edge from node p to node q
end for
return Graph in Neo4J

F. Graph Validation Approach

To ensure a proper generation of Graph database from its corresponding relational database, we perform information quality measurement on the created graph based on the schema information of the relational database. In this study, we adopted part of the Information Quality (IQ) classifications as used by [10] with some slight modification. In our study, we classify the criteria into: (1) *schema completeness* i.e., the number of the tables in a given relational schema must match the number of the nodes created in the graph database; (2) *relationship completeness* i.e., relations between tables such as 1-1, 1-N, M-N of the relational database must be found in the form of corresponding edges created in the graph database. To check Functional Dependencies between tables from a relational database that is converted into a graph database, we assume that the relationships, data, and the constraints in the source database are proper, i.e., it contains no dangling data, data with relation from a table to the others has a matching number of table relationships or constraints found intact in its configuration. For a 1-N or N-1 relation, there must be one foreign key that is referred from table A to table B, or vice versa. The same case for M-N relation between table A and table B that uses an intermediate table C, so the converted relation is an N-1 relationship from table A to table C, and from table B to table C; (3) *data consistency* i.e., information on the results of queries performed on a graph database must be consistent with the results of queries performed on a relational database. For this purpose, we adopted the *Bloom Filter* technique used by [11] to validate the converted data from a relational database to a NoSQL database. Given that the structure and query language used in relational databases and graph databases are different, we need to concatenate data of every row of tables in the relational database the same way with the data from graph database before being inserted into the Bloom Filter for automatic validation. More formally, the validation criteria are defined as follows:

Definition 1: Schema Completeness (SC). Consider an entity T_i of a relational schema S has a set of records A_i, a graph database $G\,(V, E)$ where V is a node and E is an edge between two nodes, and let N_i is the set of nodes of the graph G for the corresponding relational entity. SC is given by:

$$SC_{T_i} = \frac{n(N_i)}{n(A_i)}$$

$$SC = \frac{\sum_i SC_{T_i}}{\sum_i T_i} \tag{1}$$

Definition 2: Relationship Completeness (RC). In addition to Definition 1, consider that the entity has a set of relationships R_i among T_i and other entities in the schema. Note that a relation in the set can be optional, i.e., the foreign key of a given relation can be either "Non-nullable" or "Nullable". Let D_i is a set of directed edges between two nodes in graph database for the given entity. If we consider a record set of the given entity whose associated edge set is not created in the graph database due to the foreign key is Null in the relational database as $\mathring{A}_i$, RC is given by:

$$AR_{T_i} = n(A_i) * n(R_i)$$

$$RC = \frac{\sum_i n(D_i)}{\sum_i (AR_{T_i} - n(\mathring{A}_i))} \tag{2}$$

Definition 3: Data Consistency. To use Bloom Filter, we start by querying tables in the relational database. As an example, consider tables with relationships as shown in Figure 1 which consists of tables called *film*, *category*, and *film_category*. Every record of the tables will be inserted into the Bloom Filter and then the respective node in the graph database will be checked for membership in the Bloom Filter. Initially, an empty Bloom Filter for each table will be created in the form of an array with m bits, all set to zero. A k number of different hash functions must be defined to map or hash all tuples/nodes as n inserted elements to positions in the array generating a uniform distribution. Upon checking a data from graph node against the Bloom Filter, it guarantees that false positive matches are possible, while false negative are not –i.e., a result will be either "possibly exists" or "definitely not exists". There are several techniques proposed to reduce the false positive rate, in our case, the false positive rate δ of a "possibly exist" is given by:

$$\delta = (1 - e^{\frac{-kn}{m}})^k \tag{3}$$

To optimize the query results with a desired false positive rate δ and the number of the elements to be inserted in the Bloom Filter n, the calculation of number of bits m is given by:

$$m = \frac{-n\ ln\ \delta}{(ln\ 2)^2} \tag{4}$$

From the m value, we can calculate the optimal number of hash functions k that is given by:

$$k = \frac{m}{n} \ln 2 \tag{5}$$

By using Bloom Filter, we can get all records in the graph database that are not in the relational database up to a desired false positive probability. Then, based on the mismatch records, we can perform query into the relational database to determine whether or not they are duplicated, or extra records have been created in the graph database.

IV. Experiment

A. System Specification

Our experiment is done with the computer specification as given in Table II.

B. Graph Generation in Neo4j

The graph creation is done by converting several examples MySQL databases called *'Sakila', 'Classicmodels', 'Pubs',* and *'Imdb'*. During the conversion, the metadata is extracted from the schema information of the relational databases, and then any cyclic dependency is resolved prior to any of nodes or edges generation is done in Neo4J. Since there is a possibility

of "Nullable" relationship in the relational database, during the automated graph generation, it is important to skip non-matching node for relationship that has foreign key equal to Null in the relational database and continue to process the rest. For this purpose, Neo4J provides a convenience way to optionally match a node and continue the script execution without fail by using OPTIONAL keyword in the Cypher script. A portion of the graph created showing the cyclic dependencies depicted in Figure 3.

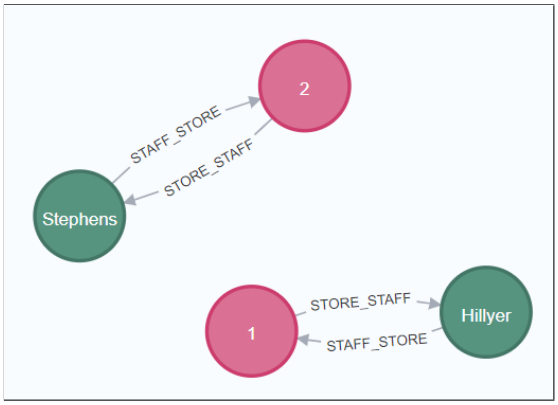

Fig. 3. Tables with non-identifying relationship and a cyclic dependency

C. Graph Validation

To perform the schema and relationship completeness as defined in Section III-F, we run Cypher query in Neo4j to verify whether or not the total number of nodes created for each of the node labels in the graph databases is indeed matched with the corresponding number of tables and relationships exist in the relational database according to (1) and (2). The schema and relationship checking results are provided in Table III and Table IV, respectively.

By using (1), we can observe the completeness of the node creation by calculating the SC rate as follows:

$$SC = \frac{\sum_1^{16} SC_{T_i}}{16} = 100\ \%$$

TABLE II
EXPERIMENTAL SYSTEM SPECIFICATION

Software/Hardware	Specific Name/Version
Operating Systems	Windows 10 Professional
Processor	Intel® Core ™ i5-3360M CPU @ 2.80 GHz
RAM	16 GB DDR3 SODIMM 1600 MHz
Hard Disk	Samsung SSD 850 EVO 500 GB
Softwares	MySQL Server 10.1.40-MariaDB distribution, PostgreSQL 14.3.1 Windows-x64, Neo4J Community Edition 4.4.6, Jupyter-Notebook 6.4.3

Notice, 16 is the total number of tables/nodes in both relational as well as graph databases.

As shown in Table IV, the number of edges of the Payment_Rental is 16,044 instead of 16,049. The Payment table of the 'Sakila' relational database has three foreign keys, in which *rentalid* can be *null*. In the table, there are 5 records having *null* values for the foreign key which causes no edges were created for these cases. Thus, the total number of edges in Neo4J is 121,769. By using (2), the RC rate can be calculated as follows:

$$RC = \frac{\sum_1^{16} n(D_i)}{(121,774 - 5)} = 100\ \%$$

In addition, in order to perform the data consistency checking, the initial step is to choose the desired False Positive rate δ for the validation or otherwise can be computed using (3) and then continue with the calculation of the width of the Bloom Filter bits m using (4) as well as the number of hash function k using (5). In this experiment, for this particular 'Sakila' database conversion, we choose $\delta = 0.01$ and calculate $k = 6$ and we get m values vary depending on the no. of the source's records n that are available. For the hash function, there are several options available for use, one of which is the *MurmurHash*. It is a non-cryptographic hash function developed by Austin Appleby in 2008 and it is suitable for general hash-based lookup which has been adopted in a number of open source projects. The next step is to fill in the Bloom Filter by using the records of tables available in the relational database. In order to perform this task, for a given records/node, the values of its attributes/properties are concatenated as a string. Any 'nan' or 'None' as well as an additional '.0' due to type conversion if any is replaced with a blank before they are inserted into the Bloom Filter. The same treatment is done to both data retrieved from the relational as well as graph databases. The last step is to query to each of the node of the graph in Neo4J by first concatenating its attribute values the same way when we input source's records into the Bloom Filter. Based on the result, we can determine the consistency of the data as converted into nodes in the Neo4J. For example, checking the *Payment* and *Rental* tables which have the highest number of records, the process took about 17 and 14 seconds respectively as seen in Table V.

To see the performance of the algorithm in various databases with varying number of tables and records, we provide the results as given in Table VI. As observed, the number of nodes tally with the number of records of each database. Therefore

TABLE III
SCHEMA COMPLETENESS CHECKING RESULTS OF 'SAKILA' DATABASE

Node	A	N	SC_T
Inventory	4,581	4,581	4,581 / 4,581
Payment	16,049	16,049	16,049 / 16,049
Rental	16,044	16,044	16,044 / 16,044

Note: Only 3 out of 16 entities are shown

TABLE IV
RELATIONSHIP COMPLETENESS CHECKING RESULTS OF 'SAKILA' DATABASE

Node	Edge	D	AR_T	$\mathring{A}$
Inventory	Inventory_Film	4,581	9,162	0
	Inventory_Store	4,581		
Payment	***Payment_Rental***	***16,044***	48,147	5
	Payment_Customer	16,049		
	Payment_Staff	16,049		
Rental	Rental_Customer	16,044	48,132	0
	Rental_Inventory	16,044		
	Rental_Staff	16,044		

Note: Only 3 out of 16 entities are shown

TABLE V
DATA CONSISTENCY CHECKING RESULTS OF 'SAKILA' DATABASE

Table Name	Records	m	k	Time (mins)
Inventory	4,581	43,909	6	00:02
Payment	***16,049***	***153,830***	***6***	***00:17***
Rental	***16,044***	***153,782***	***6***	***00:14***

Note: Only 3 out of 16 entities are shown

TABLE VI
CONVERTION OF SEVERAL MYSQL DATABASES

Database	Table	Records	Nodes	Edges	Time (mins)
Sakila	16	47,273	47,273	121,769	06:50
Classicmodels	8	3,864	3,864	6,846	00:16
Pubs	11	255	255	291	00:03
Imdb_small	7	4,395	4,395	4,448	00:09

the SC value for each is 100%. However, to verify the RC value of each database, the actual number of edges can be queried from the Neo4J user interface, and by using (2) we can validate that the relationship completenes is 100%, and the converted records are consistent with the created nodes for each of the database.

CONCLUSION

In order to automate the conversion of relational database which consists of relationship dependencies into its corresponding graph database, this paper proposes a novel technique to perform the conversion process, dependency checking, and validation against the created graph.

We use the metadata extraction method to get information about the tables and their relation dependencies before we sort them with a modified topological sorting algorithm to solve the cyclic dependency issues in the schema if any. We propose three ways of conversion validation by ensuring schema completeness, relationships, and data consistency using Bloom Filter. Our test results show that the approach converts relational databases to graphs effectively and conversion checks can be performed in a matter of seconds. However, this method has been tested against a few sample databases with less than 100,000 records only.

In our future research, we would like to test the performance of the approach against different databases such as Oracle, MSSQL Server, Postgres, as well as with larger records. In addition, in this study we have not checked the data types that must be properly converted into the properties of the resulting graph database.

REFERENCES

[1] Y. Zou and Y. Liu, "The Implementation Knowledge Graph of Air Crash Data based on Neo4j," *Proceedings of 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2020*, no. Itnec, pp. 1699–1702, 2020. [Online]. Available: https://doi.org/10.1109/ITNEC48623.2020.9085182

[2] H. Feng and M. Huang, "An Approach to Converting Relational Database to Graph Database: from MySQL to Neo4j," *Proceedings of the 2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA) 2022*, no. January, pp. 674–680, 2022. [Online]. Available: https://doi.org/10.1109/ICPECA53709.2022.9719151

[3] T. Wang, Y. Wang, and C. Tan, "Construction and Application of Knowledge Graph System in Computer Science Department," *Proceeding of the 2018 IEEE International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pp. 169–172, 2018. [Online]. Available: https://doi.org/10.1109/SPAC46244.2018.8965547

[4] P. Idziaszek, S. Kujawa, W. Mueller, and M. Łukomski, "Mapping of relational structures in graph database Neo4j," *Journal of Research and Applications in Agricultural Engineering*, vol. 63, no. 4, p. 121, 2018.

[5] R. Arora and S. Goel, "Javarelationshipgraphs (jrg): Transforming Java projects into graphs using neo4j graph databases," *ACM International Conference Proceeding Series*, pp. 80–84, 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3305160.3305173

[6] H. Liu, G. Jiang, L. Su, Y. Cao, F. Diao, and L. Mi, "Construction of power projects knowledge graph based on graph database Neo4j," *Proceedings of the 2020 IEEE International Conference on Computer, Information and Telecommunication Systems, CITS 2020*, pp. 20–23, 2020. [Online]. Available: https://doi.org/10.1109/CITS49457.2020.9232609

[7] S. Cheng, T. Wang, X. Guo, and Y. Wang, "Knowledge Graph construction of Thangka icon characters based on Neo4j," *Proceedings - 2020 IEEE International Conference on Intelligent Computing and Human-Computer Interaction, ICHCI 2020*, pp. 218–221, 2020. [Online]. Available: https://doi.org/10.1109/ICHCI51889.2020.00054

[8] J. Bai and L. Che, "Construction and Application of Database Micro-course Knowledge Graph Based on Neo4j," *ACM International Conference Proceeding Series*, vol. PartF16898, 2021. [Online]. Available: https://doi.org/10.1145/3448734.3450798

[9] M. Goldstein and D. Moshkovich, "Improving software through automatic untangling of cyclic dependencies," *ACM 36th International Conference on Software Engineering, ICSE Companion 2014*, pp. 155–164, 2014. [Online]. Available: https://doi.org/10.1145/2591062.2591182

[10] C. Moraes and A. C. Salgado, "Information Quality Measurement in Data Integration Schemas," *Proceedings of the Fifth International Workshop on Quality in Databases, VLDB*, 2007.

[11] A. Goyal, A. Swaminathan, R. Pande, and V. Attar, "Cross platform (RDBMS to NoSQL) database validation tool using bloom filter," *2016 International Conference on Recent Trends in Information Technology, ICRTIT 2016*, 2016. [Online]. Available: https://doi.org/10.1109/ICRTIT.2016.7569537