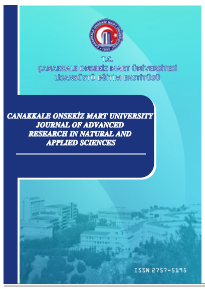
e-ISSN 2757-5195



Çanakkale Onsekiz Mart University
Journal of Advanced Research in Natural and Applied Sciences
Open Access

doi.org/10.28979/jarnas.936702 2022, Vol 8, Issue 2, Pages: 281-292 dergipark.org.tr/tr/pub/jarnas

Analyzing The Encountered Problems and Possible Solutions of Converting Relational Databases to Graph Databases

Ramazan Altın[1*], Ahmet Cumhur Kınacı[1]

[1]Department of Computer Engineering, Faculty of Engineering, Çanakkale Onsekiz Mart University, Çanakkale, Türkiye

Article History

Received:	12.05.2021
Accepted:	13.01.2022
Published:	10.06.2022

Research Article

Abstract − Relational database management systems have been used for storing data for a long time. However, these systems are insufficient to analyze the large and complex structure of the data. Graph databases are becoming more common day by day due to their capacity to contribute to the analysis. Also, graph databases are better at modeling and querying complex relationships than relational databases. To use graph databases with old data stored in relational databases a transfer process is needed. In this study, the problems to be encountered in transferring the data stored in a relational database to a graph database were examined and methods that could be used as solutions to them were proposed. In addition, it is aimed to prevent data loss and data inconsistency that may occur with design errors in relational databases. For this purpose, the normalization process needs to be applied to a relational database before transferring data to a graph database. In our study, we developed a method that converts data to the first normal form during the transfer. But for better data consistency in practice third normal form is the minimum requirement. By using the functional dependencies found, it is possible to make relational databases suitable for higher normal forms. For functional dependency detection, which is normally a very time-consuming and costly process, we developed a method based on a graph database.

Keywords *− Functional dependency, graph database, normalization, normal form, relational database,*

1. Introduction

In file systems it is difficult to define relationships between data therefore relational databases are developed. It is necessary to prevent possible data loss and data inconsistency in relational databases and to ensure data integrity. In order to fulfill this requirement, the standards that relational databases should have been defined (Haerder & Reuter, 1983). The NoSQL database approach, in which these standards are operated more flexibly, emerged as a concept in 1998. NoSQL databases provide data storage without applying defined standards for relational databases but maintaining data integrity and consistency (Nayak, Ameya, Anil Poriya, and Dikshay Poojary, 2013). Since NoSQL does not need structures such as tables, rows, and columns it is not affected by structural changes and provides scalability and usability for systems containing large data. Graph databases used under NoSQL systems consist of nodes that are very similar to real-world entities (objects) instead of tables. Node and simple relations are used by all graph databases to fit the model that creates the simplest representation of data (Angles, 2012). In accordance with the graph theory, edges define the relationship between nodes in these databases. Graph databases are faster, lower cost and simpler compared to relational databases, theoretically and often in practice (Celko, 2014). Also, the query efficiency of Neo4j is faster than relational database (Nan & Bai, 2019). For these reasons, the use of graph databases has continued to increase in recent years. The most popular graph database is the Neo4j database, which we preferred in the study (Table 1).

[1] ramazan@comu.edu.tr
[2] cumhur.kinaci@comu.edu.tr
*Corresponding Author

Table 1

October 2021 popularity ranking of graph databases according to db-engines

Rank	**DBMS**	**Database Model**	**Score**
1	Neo4j	Graph	57.87
2	Microsoft Azure Cosmos DB	Multi-model	40.29
3	Virtuoso	Multi-model	4.69
4	ArangoDB	Multi-model	4.45
5	OrientDB	Multi-model	4.05
6	GraphDB	Multi-model	2.65
7	JanusGraph	Graph	2.52
8	Amazon Neptune	Multi-model	2.39
9	TigerGraph	Graph	1.99
10	Stardog	Multi-model	1.93

The ranking[3] is based on number of search engine results when searching for the system names, Google Trends, Stack Overflow discussions, job offers with mentions of the systems, number of profiles in professional networks such as LinkedIn, mentions in social networks such as Twitter.

Organizations that operate with large amounts of data choose to use graph databases together with relational databases as a hybrid on an application (Vyawahare, H. R., Pravin P. Karde, and Vilas M. Thakare, 2018). In addition, it may be preferable to transfer a relational database completely to a graph database. Within the scope of this study, a method and application that will enable transfer from any relational database to Neo4j database, which is a graph database, has been developed. The application was developed with the Java programming language and Cypher, which is Neo4j query language, was also used in the transfer.

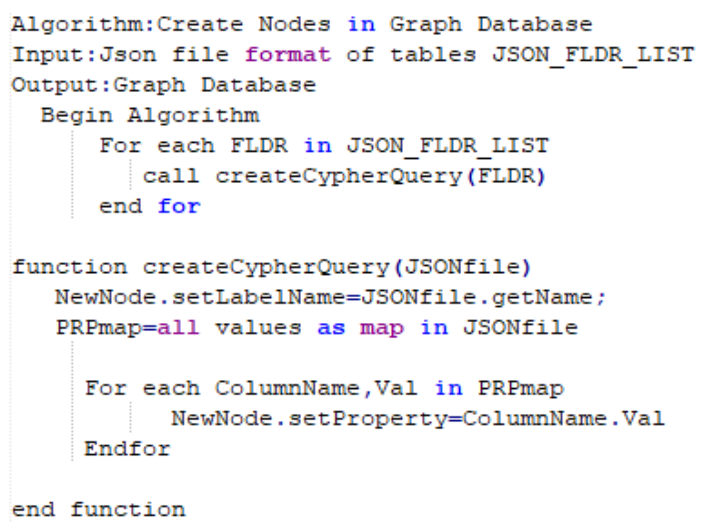
During the transfer, unnecessary data repetitions and design errors detected in the relational database, in this way problems in updating, deleting, and adding operations are prevented. All of these processes are called normalization. Normalization can be performed with many different algorithms using functional dependencies (Bahmani, Amir Hassan, Mahmoud Naghibzadeh, and Behnam Bahmani, 2008), (Dongare, Y. V., P. S. Dhabe, and S. V. Deshmukh, 2011). Our goal is to transfer from the relational database to the graph database without data loss. Also, we achieved functional dependencies that play a vital role in finding the difference between good and bad database design

2. Materials and Methods

In related studies, either data is transferred to a graph database by a direct connection (JDBC) or using a CSV formatted file contains exported data. JSON formatted files can keep hierarchical data more organized than CSV files. In the developed method, we can transfer data from any relational database to a graph database by using JSON. Method-1 in section 2.1 includes the pseudocode shown in Figure 1 that generates the cypher query. This query transfers all data at once by creating nodes. Method-2 uses a different method than Method-1 and transfers the cell as a node.

Singh, M., & Kaur, K. (2015) transferred the database containing health data created from 24 tables on MySQL to the Neo4j database. While transferring from the relational database to the graph database, the most used data were recorded close to each other by considering 5 cases. It has been shown that query times give better results in this way than a relational database. While it is advantageous in medium and large-scale

[3]https://db-engines.com/en/ranking_definition



```
Algorithm:Create Nodes in Graph Database
Input:Json file format of tables JSON_FLDR_LIST
Output:Graph Database
  Begin Algorithm
      For each FLDR in JSON_FLDR_LIST
         call createCypherQuery(FLDR)
      end for

function createCypherQuery(JSONfile)
   NewNode.setLabelName=JSONfile.getName;
   PRPmap=all values as map in JSONfile

     For each ColumnName,Val in PRPmap
           NewNode.setProperty=ColumnName.Val
     Endfor

end function
```

Figure 1. The pseudocode of data migration

databases, applying these steps in the process of transferring databases with more rows but few tables will bring extra costs.

Yelda Unal and Halit Oguztuzun (2018) used JDBC Connection and Java SQL library for extracting data and metadata from Relational Database Management Systems. The output data was transferred to the Neo4j database with the Neo4J Parallel Batch Importer API. Searching for the given data value between two thousand law data items has resulted 0.01 second in graph database and this result is ten times faster than relational database.

Vyawahare, H. R., Karde, P. P. & Thakare, V. M. (2019), after exporting the data in the relational database as csv file, they moved it to the graph database. Tables that have more than two foreign keys get converted to nodes and the foreign keys to the other tables are converted as relationships. Comparing the query times between the two databases for 5 queries after the transfer, it was observed that the graph database gave better results.

2.1. Proposed Transfer Method-1: Database Transfer Without Normalization

Most databases provide migration methodologies commonly used with relational databases. The first of these is to transfer through file reading. JSON is unstructured data, unlike a CSV file which has to make each row hold the same type of data. It is, therefore, more flexible. There are also databases that use JSON as their primary data format. Due to these features, we can transfer any database to be given in JSON file format, which we prefer, to the graph database by using Method 2-1. As in Table 2 and Table 3, the data in the defined relational database can be exported in JSON file format via various queries or tools. In this way, it is possible to transfer data from all databases to graph databases, regardless of the database type. Primary and foreign key information was taken in JSON file format with the queries and these fields were transferred to the graph database.

Table 2

First table in database

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4

Table 3

Second table in database

X	Y	Z	T
A1	Y1	Z1	T1
A2	Y2	Z2	T2
A3	Y3	Z3	T3
A4	Y4	Z4	T4

In databases that do not need normalization, all data in a row can be transfer as a single node. The transfer is carried out by applying the steps shown in Figure 2.

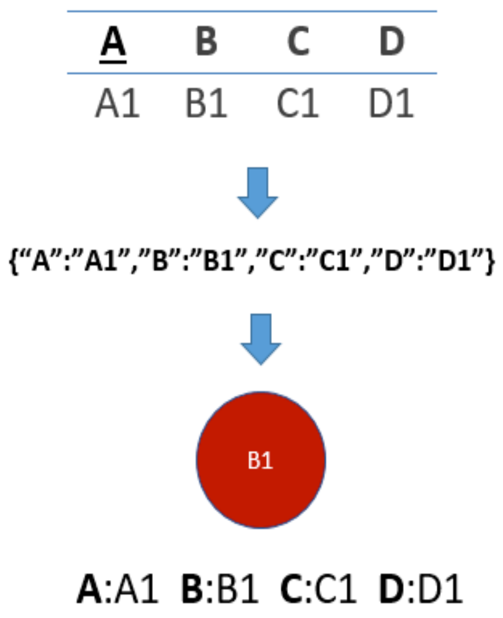


Figure 2. The process of transfer a row to the graph database

In the first stage, all nodes are created one by one without any relationships being established. The table name creates the label of the node and all values in the row are saved to the node. The representation of the tables is shown in Figure 3 before relationships are established between the nodes.

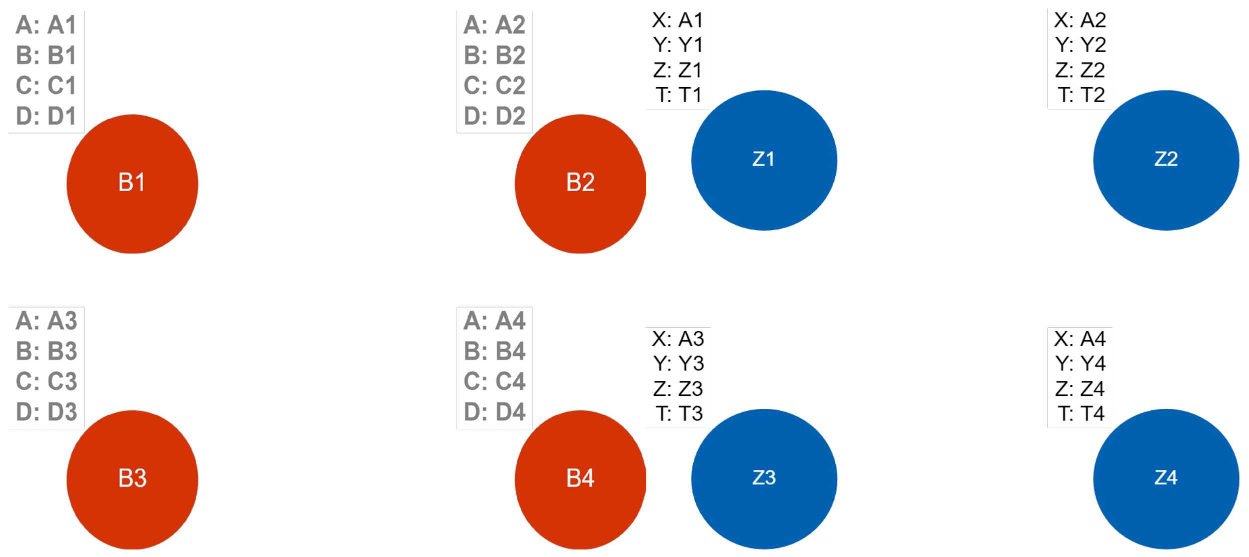


Figure 3. The situation before the relationship is established

Relationships between nodes are defined by associating columns designated as foreign keys. As the last step of the transfer, for the primary key and the fields that should be unique, these fields are also performed in the graph database with the command written with Cypher. All data were transferred as in Figure 4, and all relationships were established in the Neo4j database.

It is possible to give a label to the relationship established between two nodes and to keep extra data about the relationship. A special type is not specified for the relation names, since there will not be any retroactive transfer from the graphic database to the relational database. For this reason, Neo4j's default relationship type (Reltype) is used.

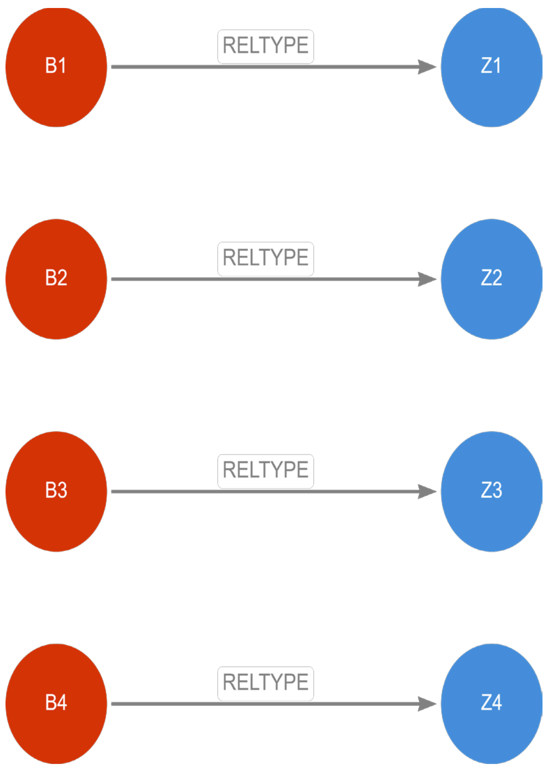


Figure 4. Neo4j database example after transfer end

2.2. Proposed Transfer Method-2: Database Transfer with Normalization

The relationship between the two data sets can be defined by the mapping function: In the representation F: D → R, the name of the F mapping function is D and R is the data set (Ball-Rokeach & DeFleur, 1976). Functional dependency is a match type; expressed as an arrow "→". The representation A → B is the statement that A is the determinant of B. This expression implies that for each value of A there is only one value of B, that is, column A determines column B. Knowing the functional dependencies is a requirement for the implementation of normalization steps.

In the study, possible functional dependencies of a table were found by using the graph database. In tables that do not require normalization, each row is kept on a single node. Each cell is transferred to match a node on the graph database when finding functional dependencies. Repetitive cell values in the relational database will be added to the graph database only once. We used string similarities to detect these fields. In this way, unnecessary data duplication, which is one of the problems that normalization tries to solve, will be prevented. In this study, the transfer on the normalization level up to the Third Normal Form (3NF) was controlled.

2.2.1. First Normal Form (1NF)

The 1NF process aims to prevent unnecessary data duplication and the steps to be taken for this are as follows;

- Eliminating duplicate groups within a table
- Create a separate table for each related data set
- Identifying each related dataset with a primary key

If null value occurs in table then it would be removed from table by entering corresponding data type value (G. Sunitha & Jaya, 2013).

Columns that do not contain any data or are completely defined as null will not be transferred to the graph database. The composite key definition (A, B) in the R (A, B, C, D, E) relationship is shown in Table 4.

Table 4

Table with composite primary key

A	B	C	D	E
A1	B1	C1	D1	E1
A1	B2	C1	D2	E2
A2	B1	C2	D1	E2
A2	B2	C2	D2	E2

Cells are places where rows and columns intersect. In transfer, each node in the graph database represents a cell. With this method, it will be provided to remove data duplication, which is one of the requirements of the first normal form. After the transfer, the display of the relevant table in the graph database is as in Figure 5.

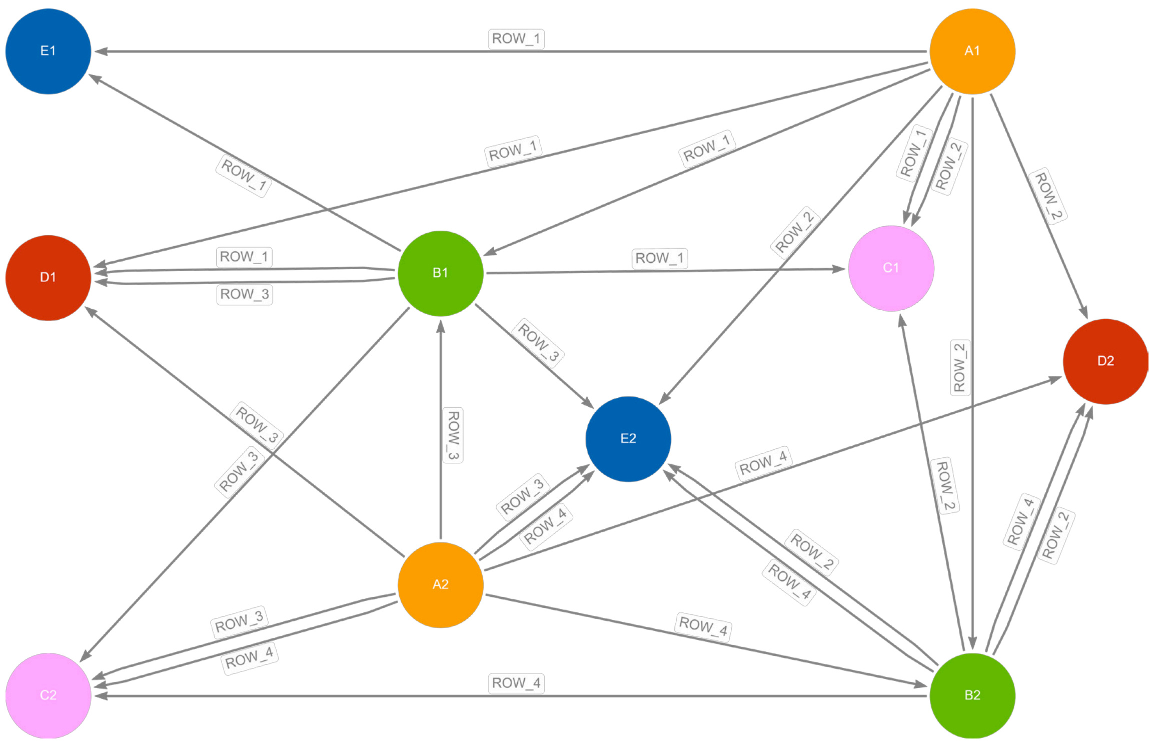


Figure 5. First normal form proper transfer

In order to transfer back from the graph database to the relational database, line numbers are given to the relation names.

2.2.2. Second Normal Form (2NF)

Two rules are defined for a table to conform to the second normal form.

1. The first should conform to the normal form.
2. If no non-key attribute is partially linked to any candidate key, it is in the second normal form (Elmasri, Ramez, 2003).

The candidate key is the key that is suitable for use as the primary key and allows to uniquely identify the row on which it is located. Composite keys are used in databases in cases where the use of a single column as a primary key is not sufficient. If one or more of the columns with the primary key defines a different column by itself, there is partial dependency and the table is not a suitable table for the second normal form. In this study, in order to determine the partial dependencies, the relations between the nodes are checked after the data is transferred to the graph database. Primary keys need to be checked for their relationship to non-key nodes.

The relationship between the primary keys, nodes A and B, with other nodes is shown in Figure 6.

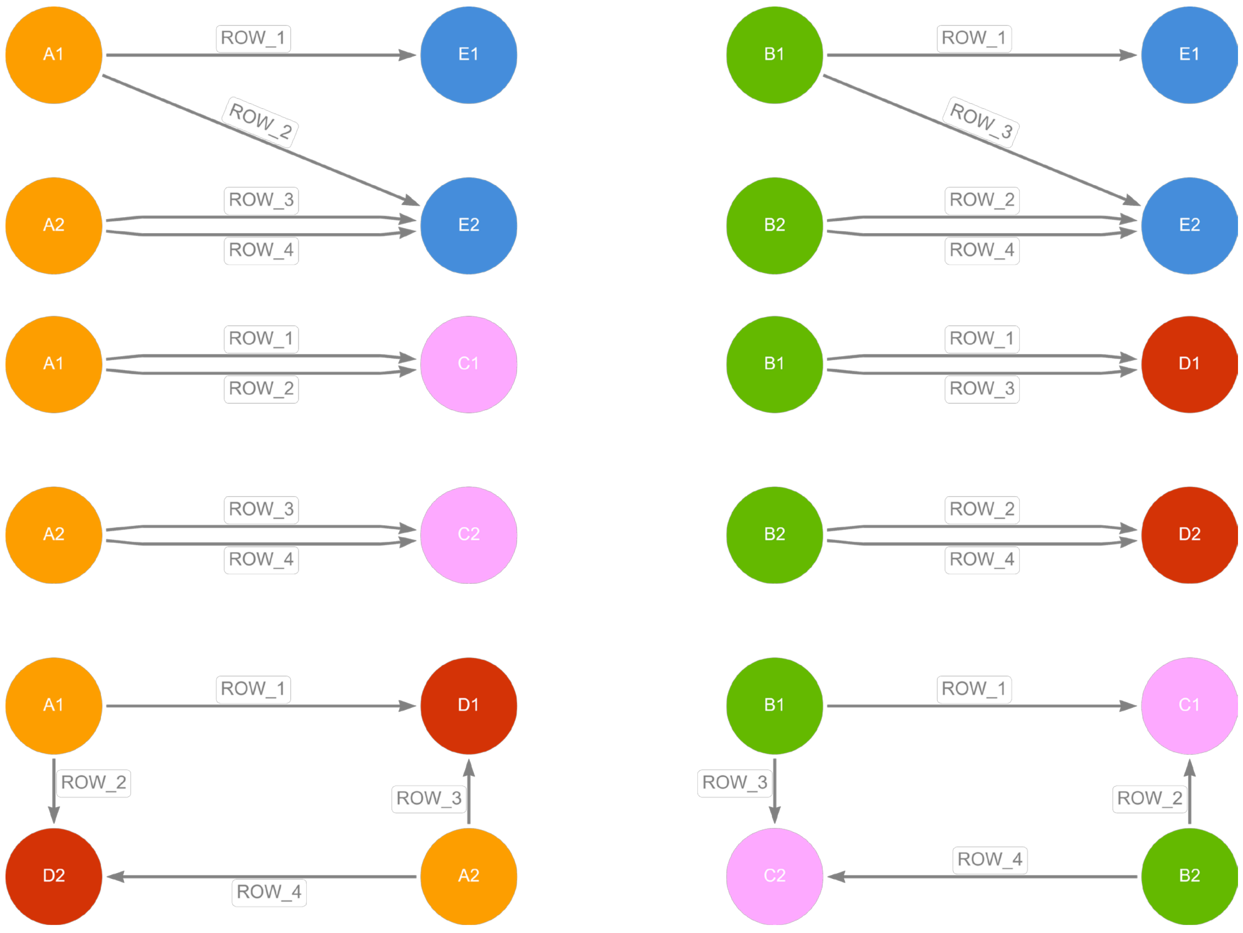


Figure 6. Relationship of key nodes with non-key nodes

For all A-type values, A {i} is always associated with a C {j} node of the same value. Since node A1 is associated only with node C1 and node A2 is only associated with node C2, column A is the determinant of column C and there is partial dependency. Since the same situation is detected within nodes B and D, column B is the determinant of column C and there is partial dependence between them.

Since it was found that there are partial dependencies for all A values and all B values with the outputs, the functional dependencies of our table are formed as A-> C, B-> D. Partial dependencies can be removed by applying various methods and the table can be made suitable for the second normal form (Dongare, Y. V., Dhabe, P. S., and Deshmukh, S. V, 2011), (Bala & Martin, 1997).

2.2.3. Third Normal Form (3NF)

The third normal form has been developed in relation to having a direct or indirect relationship between the records, without making unnecessary data repetition, by taking over the second normal form. R (X, Y, Z) relationship to fit the third normal form;

- Meeting the second normal form (2NF) criteria.
- Each non-key attribute of R is non-transitive depend on every key of R (Demba, 2013).

If X -> Y is defined and X is the primary key of R, then X -> Y and Y -> Z should not be together. If a column that does not have a primary key definition is capable of defining another column, there is transitivity in the table. In the third normal form, a relationship is established between the non-key nodes on the graph database to find whether there is transitivity or not.

Transitivity control is provided by controlling the relations of each node with other nodes as in partial dependency detection. Table 5 is a sample prepared appropriately for transitivity detection, and column A is defined as the primary key.

Table 5

Table example with transitivity

<u>A</u>	B	C	D
A1	B1	C1	D1
A2	B1	C1	D2
A3	B2	C2	D2

In the R (A, B, C, D) relationship, the A column is a key defined for the table, the cells on the graph database are transferred to the nodes as in Figure 7.

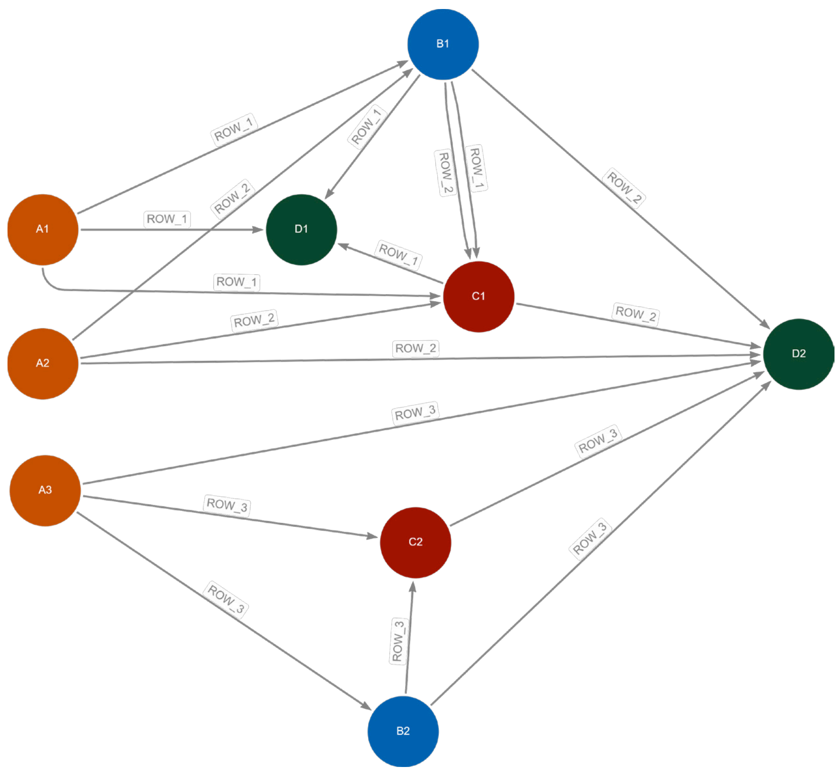


Figure 7. Transferring table 5 to the graph database

After the transfer, the relationships of non-key columns with each other will be examined. All binary combinations will be looked at. In this example, the binary combinations are B-C, B-D, and C-D. All relations are formed as in Figure 8.

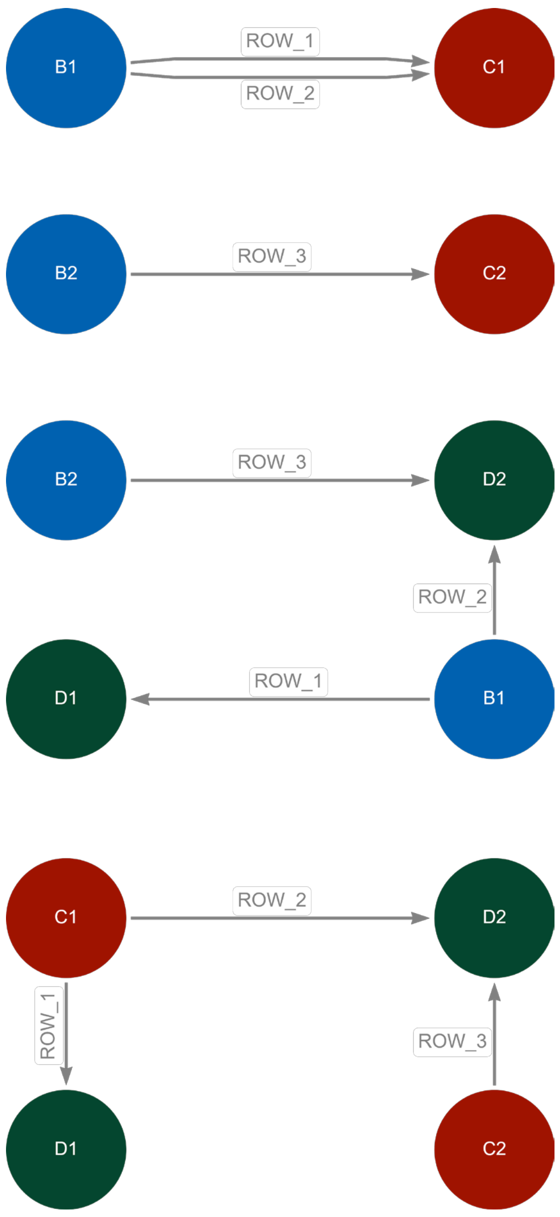


Figure 8. Relationships between non-key nodes

It is determined that the table is not suitable for 3NF and the transitivity must be removed. A new table containing columns B and C should be created and the database can be easier to modify and maintain.

3. Results and Discussion

Evaluations are made using some metrics to calculate the correct transfer of data. For databases where normalization cannot be applied, the number of relational database table rows and the number of graph database nodes were obtained equally. After the transfer is completed, it is seen in Table 6 that the number of rows of the tables in the relational database is equal to the Table 7 number of nodes in the graph database. The classicmodels.db[2] database, where the outputs are compared, consists of 8 tables and 3,864 rows.

[2]https://relational.fit.cvut.cz/dataset/ClassicModels

Table 6

Relational database table names and row counts

Sql_Query	Table_name	Row_count
Select table_name as ‘Table_name’,table_rows as ‘Row_count’ from information_schema.tables where table_schema=’classicmodels’;	customers	122
	employees	23
	offices	7
	orderdetails	2996
	orders	326
	payments	273
	productlines	7
	products	110

Table 7

Graph database node names and node counts

Cypher_Query	Node_Names	Node_Count
MATCH (n) RETURN count(labels(n)) as node_count, labels(n) as node_names;;	customers	122
	employees	23
	offices	7
	orderdetails	2996
	orders	326
	payments	273
	productlines	7
	products	110

Also, it is seen that the number of features kept in the node is equal to the number of cells in the relational database. The transfer of tables created in JSON format provides a performance advantage, and it offers a general use regardless of the type of database to be transferred. The proposed Transfer Method-1 is not suitable for finding functional dependencies and applying normalization, so it was necessary to specify a different transfer method. For this reason, instead of transferring the row as a node, the method of transferring the cell as a node is applied. In order not to lose the pattern, a node is created for null-defined cells in the relational database and a relationship is established with the cells in the same row. No nodes have been created in the graph database for a fully null-defined column. For fields defined as keys or unique in the relational database, these properties are defined while creating the node in order to avoid problems while entering new data in the graph database. While determining partial dependency, the same solution is applied for tables with one or more keys.

In the database transfer to which normalization will be applied, the relationships are named with row numbers, and this method allows backward transfer from the graph database to the relational database. The consistency of the results was checked by performing the transfer process for more than one database.

4. Conclusion

This paper examines what may be required to transfer data from a relational database to a graph database consistently. The normalization level of the relational database directly affects this situation. For this reason, we developed two methods to transfer by considering the normalization levels. The first one (method 1) simply converts every row to a node without any normalization. If the relational database is not 1nf, we have applied the 1nf criteria that can be found structurally on data while transferring to a graph database. Primary keys, unique columns, empty defined fields, empty cells were checked during the transfer. In order to ensure the 2nf and 3nf levels, functional dependencies (partial dependency and transitive) should be determined. Functional dependencies are found by examining the relationships between nodes on the graph database created with the transfer method 2 we propose. When functional dependencies are given in the literature, there are various algorithmic methods for 2nf and 3nf transformations and these can be used. Thus, after transforming to 2nf and 3nf levels, the transfer method 1 we proposed is applied, and the data is transferred more consistently. Normalization criteria defined specifically for relational databases can also be applied in graph databases with certain changes. In future studies, the methods of applying these normalization level criteria to the graph database will be studied.

Acknowledgement

The author received no specific funding for this study.

Author Contributions

Ramazan ALTIN: Gathered the data, run algorithms and evaluated the conclusions.

Ahmet Cumhur KINACI: Performed data analysis and wrote the results and discussion

Conflicts of Interest

The authors declare no conflict of interest.

References

Ameya, N., Anil, P., & Dikshay, P. (2013). Type of NOSQL databases and its comparison with relational databases. *International Journal of Applied Information Systems*, *5*(January 2013), 16–19.

Angles, R. (2012). A comparison of current graph database models. *Proceedings - 2012 IEEE 28th International Conference on Data Engineering Workshops, ICDEW 2012*, *April 2012*, 171–177. https://doi.org/10.1109/ICDEW.2012.31

Bahmani, A. H., Naghibzadeh, M., & Bahmani, B. (2008). Automatic database normalization and primary key generation. *Canadian Conference on Electrical and Computer Engineering*, *June*, 11–16. https://doi.org/10.1109/CCECE.2008.4564486

Bala, M., & Martin, K. (1997). A mathematical programming approach to data base normalization. *INFORMS Journal on Computing*, *9*(1), 1–14. https://doi.org/10.1287/ijoc.9.1.1

Ball-Rokeach, R., & DeFleur, C. (1976). Dependency Model. *Communication Research*, *3*, 6–17.

Celko, J. (2014). Graph Databases. In *Joe Celko's Complete Guide to NoSQL*. https://doi.org/10.1016/b978-0-12-407192-6.00003-0

Demba, M. (2013). Algorithm for Relational Database Normalization Up to 3NF. *International Journal of Database Management Systems*, *5*(3), 39–51. https://doi.org/10.5121/ijdms.2013.5303

Dongare, Y. ., Dhabe, P. ., & Deshmukh, S. . (2011). RDBNorma: - A semi-automated tool for relational database schema normalization up to third normal form. *International Journal of Database Management Systems*, *3*(1), 133–154. https://doi.org/10.5121/ijdms.2011.3109

Elmasri, Ramez, and S. B. N. (2003). Dbms. In *Encyclopedia of Genetics, Genomics, Proteomics and Informatics*. https://doi.org/10.1007/978-1-4020-6754-9_4159

G. Sunitha, & Jaya, A. (2013). A knowledge based approach for automatic database. *International Journal of Advanced Research in Computer Engineering and Technology*, *2*(5), 1816–1819.

Haerder, T., & Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, *15*(4), 287–317. https://doi.org/10.1145/289.291

Nan, Z., & Bai, X. (2019). The study on data migration from relational database to graph database. *Journal of Physics: Conference Series*, *1345*(2). https://doi.org/10.1088/1742-6596/1345/2/022061

Singh, M., & Kaur, K. (2015, June). SQL2Neo: Moving health-care data from relational to graph databases. In *2015 IEEE International Advance Computing Conference (IACC)* (pp. 721-725). IEEE.

Unal, Y., & Oguztuzun, H. (2018, March). Migration of data from relational database to graph database. In Proceedings of the 8th International Conference on Information Systems and Technologies (pp. 1-5).

Vyawahare, H. R., Karde, P. P., & Thakare, V. M. (2018). A Hybrid Database Approach Using Graph and Relational Database. *Proceedings of the 2018 3rd IEEE International Conference on Research in Intelligent and Computing in Engineering, RICE 2018*, *September*, 1–4. https://doi.org/10.1109/RICE.2018.8509057

Vyawahare, H. R., Karde, P. P., & Thakare, V. M. (2019). An efficient graph database model. *Int. J. Innov. Technol. Explor. Eng.*, *88*(10), 1292-1295.