A Systematic Approach for Converting Relational to Graph Databases

Đukić, Marija; Pantelić, Ognjen; Pajić Simović, Ana; Krstović, Stefan; and Jejić, Olga

In database design, a system can be abstracted into three conceptual elements: a collection of entities, the relationships among them, and the attributes describing each entity. The database serves as a system for storing data through the mentioned conceptual elements. Different database design approaches are customized to suit particular use cases e.g. the comparison between graph databases and relational databases. Graph databases are particularly well-suited for handling data with dense relationships, as they are designed to store and represent complex networks of interconnected data. Relational databases pose a challenge in scenarios where the graph would be better suited. The migration process involves restructuring the data and adapting the application logic which can be resource-intensive and time-consuming. Current solutions for database migration are often too generalized, resulting in a lack of effectiveness in addressing common migration cases. These solutions fail to provide the necessary level of specificity required to overcome the challenges that arise during the migration process. This paper proposes a structured approach for transferring data from a relational to a graph database. The proposed approach introduces strategies dedicated to the conversion of specific relational elements, such as associations, specializations, and many-to-many relationships. The approach was tested using Microsoft's Northwind sample database. Upon transferring the data from a relational to a graph database, the paper reports that queries produced identical results, indicating that the details of the data were accurately preserved during the migration. Following an experimental analysis, the results indicate that the proposed approach exhibits better performance, as evidenced by shorter query execution times. These findings affirm the feasibility and veracity of the proposed approach.

Index Terms: ***conversion, graph database, relational database, relationship***

1. INTRODUCTION

A system can be built in various ways, depending on the intended purpose. In database design, a system is typically represented as a collection of entities, their attributes, and the relationships that define their interconnections. Databases handle the aforementioned three concepts in distinct ways, depending on the selected approach. This study specifically focuses on relational and graph database models.

Relational databases are defined by a rigid structure. In the relational model, a table represents an entity with its attributes, while relationships among tables are established through primary and foreign key constraints. Table records are distinguished by a unique attribute or a unique combination of attributes known as the primary key. The primary key identifies and differentiates each record within the table. On the other hand, a foreign key stands for an attribute or a set of attributes within a table that references the primary key of another table. By utilizing primary and foreign keys, relational databases effectively maintain the integrity and connectivity of the data, allowing for efficient retrieval and manipulation of information [3].

IBM Corporation introduced SQL, or Structured Query Language, in 1974 to manage structured data. SQL has become the de facto standard for interacting with relational databases, enabling seamless data management, retrieval, and manipulation operations. Relational databases are also known as SQL databases [8].

Non-relational or NoSQL (Not Only SQL) databases are characterized by their flexible data structures. They are less affected by structural changes and can provide usability and scalability for systems containing large amounts of data. These databases offer various data models, such as key-value stores, document stores, columnar databases, and graph databases, each catering to specific use cases and requirements.

Unlike relational databases, where SQL is the widely accepted database query language, NoSQL databases do not have a universally standardized query language (e.g., Cypher, PGSQL, Morpheus, GraphQL, or Gremlin). Each NoSQL database may have its specific query language designed to work with its particular data model.

This paper examines the conversion from a relational to an attribute graph database. Each table record from a relational database is converted to a node. The relationships between tables are treated as "first-order citizens" and represented as elements that connect the source and target nodes.

The crucial differences between relational and

graph databases are [12]:

1. Relational database structure is stricter than graph structure,
2. Relationships in relational systems are inferred from foreign keys, while graph models define relationships as elements with their properties, describing the relationship in more detail,
3. Graph models support no NULL values; consequently, non-existing value entries (properties) are not present,
4. Relational database models have a primary focus on data, whereas graph models pay attention to relationships,
5. A relational database supports reading and writing equally; however, a graph database is optimized for reading.

Graph databases are well-suited for scenarios where data exhibits strong relational patterns and connectivity. Some of the notable use cases where graph databases are particularly advantageous are fraud detection, telecommunication, recommendation engines, social networks, and supply chain mapping.

A 2011 study [16], conducted on the then-active 721 million Facebook users, discovered that an average user had 190 friends. With each "friendship" connecting two users, the total number of friendships reached over 70 billion, or almost 100 per user. Extracting a list of individuals and their friends using SQL queries necessitates two JOIN operations. This process involves generating a resource-intensive Cartesian product of all possible user pairs and subsequently filtering out non-friends. Such an approach is prohibitively costly. Conversely, in a graph system, the equivalent query seamlessly connects friends through relationships, eliminating the need for a Cartesian product. Graph databases are faster at modeling and identifying associations between elements as they do not require expensive join operations.

By exploring the conversion process, the paper aims to provide insights into and guidance on how to effectively migrate data from a relational database to an attribute graph database. The presented approach ensures that the attributes associated with tables and relationships from the relational database are accurately preserved during the conversion process into the graph database. This paper outlines the specific steps and considerations involved in migrating the data and relationships from a relational model to a graph model. The data migration from MS Access to the Neo4j database is implemented as a practical example of the proposed approach. By utilizing directed graphs, Neo4j enhances the expressiveness and efficiency of working with relational data. Directed graphs differentiate between the starting and ending points of a relationship, unlike undirected graphs, where relationships are defined between two nodes without specifying which one is the “start”, and which one is the “end” [6].

The need for database migration between different systems is expected to grow in the future. This paper attempts to bridge this gap by presenting a systematic approach to database migration that is suitable for future automation. The proposed approach offers a methodology that covers various aspects of the migration process, ensuring that all essential elements are considered and preserved. Leveraging graph concepts, it offers dedicated strategies to handle the conversion process of specific relational elements, such as associations, specializations, and many-to-many relationships, ensuring accurate conversion and better query performance.

2. Related Work

Researchers have recognized the significance of having a systematic approach to converting widely used relational databases into different database models. This surge is primarily driven by the growing demand to handle semi-structured and unstructured data, which traditional relational databases are not inherently designed to accommodate. Over the past two decades, there has been a notable increase in the number of studies focused on the conversion of relational databases. In terms of the conversion to a graph-based NoSQL database, various approaches were introduced using different starting points. The entity-relationship (ER) diagram is the starting point of the conversion process in [1], [9], and [13]. As for [2], [4], [11], [15], and [17], the conversion process starts with a relational database, using a table, tuple, or table record as a starting point.

In [1], the focus of the conversion process is on the relationship types that can be found in relational databases, association and inheritance. The authors propose transformation rules that result in each entity being transformed into a node. The starting node is the entity on one side of the relationship (in the case of a one-to-one relationship, either entity can be the starting node). The ID of the start node is included as a node property for the end node. The validation of the proposed rules is performed by comparing the number of query outcomes.

The authors of [9] propose an algorithm that can migrate data by traversing the ER diagram and using transformational rules. As a result, each unrelated entity becomes a node. Related entities are also transformed into nodes, with foreign keys used to determine the direction of graph relationships. An approach based on the ER diagram is also proposed in [13]. The author

defines transformation rules for one-to-one, one-to-many, and many-to-many relationships.

Two transfer methods for converting relational to graph databases were proposed in [2]. In the first transfer method for databases that do not need normalization, each table row is converted to a single node. The second transfer method aims to find functional dependencies and apply normalization up to 3NF. For that purpose, each table cell is converted to a node. Both proposed methods map relationships from a relational model to relationships in a graph model.

In [4], the authors suggest that data values most likely to be retrieved together should be stored in the same node. Therefore, a node can contain values from different tuples. Foreign keys are inserted as properties, while relationships are converted to graph relationships.

The authors of [11] propose a relational-to-graph data conversion algorithm that can be used in the preparation of data for graph mining analysis. The algorithm follows the defined conversion order, resulting in an undirected graph whose connectedness and acyclicity depend on the relational database structure and the data contained within. Table tuples are converted to either nodes or relationships, with foreign keys becoming relationships. Attributes that are not part of any key are converted to either node or relationship properties.

An automated mechanism for the automatic conversion of relational databases to graph databases is presented in [15]. The relational database schema needs to be in the 5th normal form for the mechanism to be applied.

In [17], the authors studied the hierarchical legal document system. Using relational tables as a starting point, they propose an approach to migrating hierarchical data from relational to graph databases.

The authors of [5] draw an ER diagram for the relational database to complete the mapping between metadata. Firstly, tables are mapped into graph nodes. Later in the process, relationships are mapped to graph relationships through the “direct construction method” or by creating a connection table.

The aforementioned studies used the ER diagram or a relational database as a starting point and proposed transformation rules that addressed various types of relationships. This paper introduces a method of transforming a relational database into a graph database, starting with the relational model. The proposed approach focuses on the migration of keys, aiming to preserve all relationship types during migration.

3. The Approach

The conversion process can be structured into three distinct phases: (1) preparation, (2) loading data and generating relationships, and (3) optimizing the graph database.

The approach focuses specifically on the migration of primary and foreign keys during the database transformation process. It emphasizes the significance of these key attributes and provides insights into their preservation and handling during the migration. By specifically targeting these key attributes, the approach narrows down the scope of the migration process and delves into their translation and integration into the new database system.

This section provides both a formal and a graphical representation. The graphical representation of the entire approach can be seen in Figure 1.

A. Preparation

As a prerequisite to data migration, the relational database needs to be examined and, if necessary, modified. The graphical representation of the Preparation phase can be seen in Figure 2. The Preparation phase includes the following steps:

1. Examine the relational database for normalization – to preserve the data and complex relationships among the data, the approach applies the concept of normalization. By applying normalization, the proposed approach ensures that the relational data is structured in a way that accurately represents the relationships among different tables. This step helps to eliminate data redundancy and improve the overall integrity of the data. Denormalized tables contain derived foreign keys that violate data integrity. To preserve the consistency of redundant data, the implementation of constraints at the procedural level is required (triggers). Normalization of tables eliminates the need for trigger migration.

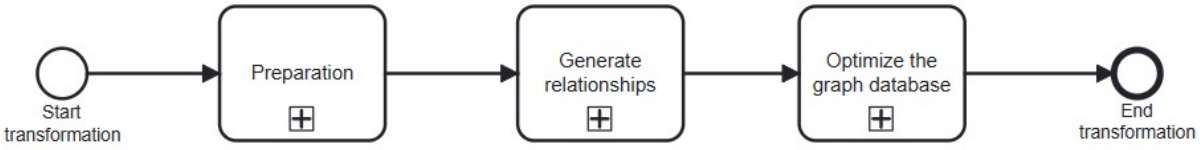

Figure 1 Graphical representation of the proposed approach

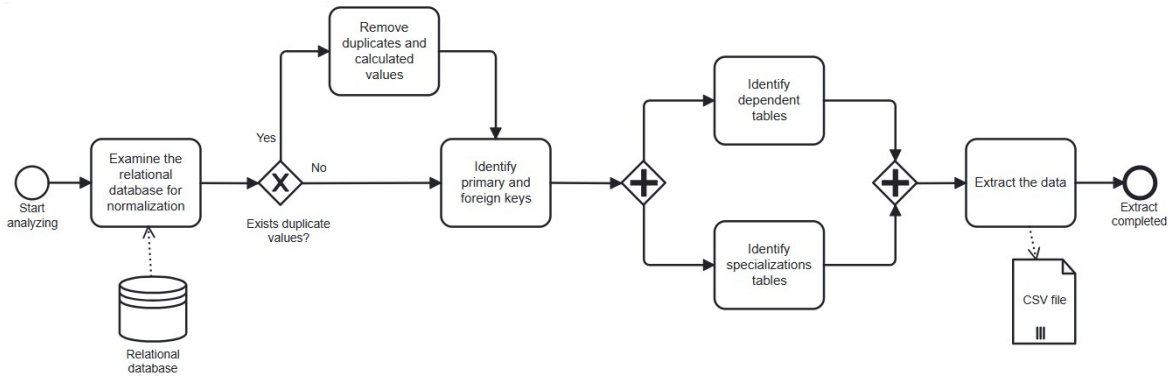

Figure 2 Graphical representation of the Preparation phase

2. Remove duplicates and calculated values - all duplicate or calculated values added previously for query optimization should be removed [13] or ignored.
3. Identify primary and foreign keys - analyze the database to identify all primary and foreign keys, as the approach focuses specifically on the migration of the key attributes.
4. Identify dependent tables – usually, it can be inferred simply by observing the keys of a table (e.g., if a primary key consists of multiple fields, the tables are most likely dependable).
5. Identify specialization tables – the specialization table is recognized based on its key. The primary key, inherited from the general table, is also a foreign key, consisting of the same attributes.
6. Extract the data – a format supported by the graph database should be used.

B. Loading the Data and Generating Relationships

This step relies on the results from the previous step to properly connect the nodes and use the potential of the graph system. The graphical representation of the Load data and generate relationships phase can be found in Figure 3. The steps of the Load data phase are:

1. Load the data - each relational database table should be loaded into the graph system using available functions.
2. Map the tables - a new node should be generated for each record of the table. Each node should be enriched with a label of the same name as the table from which the data in the node came.
3. Map the specialization tables - data from specialization tables should be stored as nodes with multiple labels. The first label should show the general table from which the node's data is taken. The other label should show which specialized table the data is taken from.

   Specialization is needed in relational models because each record in a given table needs to have the same columns. However, this is not the case in graph systems for nodes with a given label. As graph systems allow elements under the same label to have different properties, the proposed approach suggests mapping records of specialization tables into nodes with multiple labels.
4. Map one-to-one and one-to-many relationships – these types of relationships should be mapped first. In this step, relationships between basic tables are mapped into graph model relationships. Basic tables are not dependent on other tables. Relationships connecting tables that depend on only one other table are mapped next. Foreign keys should be used to form directional relationships between the nodes.

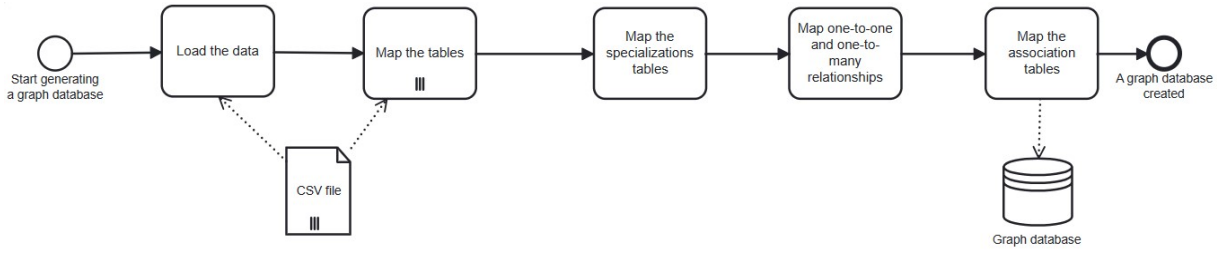

Figure 3 Graphical representation of the Load data and generate relationship phase

5. Map the association tables - records of tables created as a result of many-to-many relationships should be mapped as nodes. Association tables can be dependent on two or more tables and reference or be referenced by other tables. This makes the records of association tables unsuitable to be transformed into relationships inside graph systems. The records in these tables, therefore, are stored as nodes.

C. Optimizing the Graph Database

Some of the elements from the relational database should be removed in the following steps. They were needed in the relational model and the earlier steps of the conversion process, but have no purpose onward. Figure 4 graphically describes the final phase of the approach – the optimization phase. The steps of the Optimization phase are as follows:

1. Remove foreign keys – foreign key attributes are no longer necessary in the graph model because of the actual relationships that exist between nodes.
2. Remove technical primary keys - all technical primary keys should be removed as well, as the graph model will provide those on its own [10].
3. Add unique value restrictions - this restriction can be added both before and after the data import. The restriction will fail to be set if any of the required data is not unique.
4. Split lists of properties into individual nodes – following the suggestion form [13], lists of properties should be split into individual nodes.

4. The Approach in Action

The proposed approach was tested using Microsoft's Northwind sample database [8]. This paper outlines the specific steps involved in migrating the data and all types of relationships from the MS Access relational model to the graph model implemented in Neo4j, as a practical example. For research purposes, the authors scaled the Northwind database model and chose the entities that are crucial for the graph model.

The description of the approach given in the previous section (Section 3) will be adhered to in this section.

A. Preparation

The steps given in Figure 2 and described in Section 3A are explained in the Northwind database.

1. The database is normalized to avoid redundant data.
2. The database contains no duplicate values.
3. Primary and foreign keys are available as table metadata. Neo4j recommends distinguishing between natural primary keys and generated or technical primary keys, which hold no value outside the database [10].
4. The mapping process of dependent tables will be shown in the example of the Order Details table (Figure 7). Order Detail has its own primary key, which distinguishes it from association tables. The table can be mapped to a graph using the Cypher language code in Listing 1.

Listing 1 Conversion of Order Details to graph

```
1: match (o: Order),
2: (od:OrderDetails),
3: (p: Product)
4: where o.ID= od.OrderID
5: and od.ProductID = p.ID
6: create (o)-[: details]->
7: (od)<-[:references]-(p)
```

5. The mapping process of specialization tables will be shown in the example of the Person and Employee tables. If a person with ID = 1001 exists as a graph node with a Person label, it should be enriched with the corresponding Employee data, and the Employee label should be added. If a node does not exist, create the node with the Employee label and ID = 1001 first. Then add the Person label and the data.
6. The extraction of the data is performed through the built-in Microsoft functions.

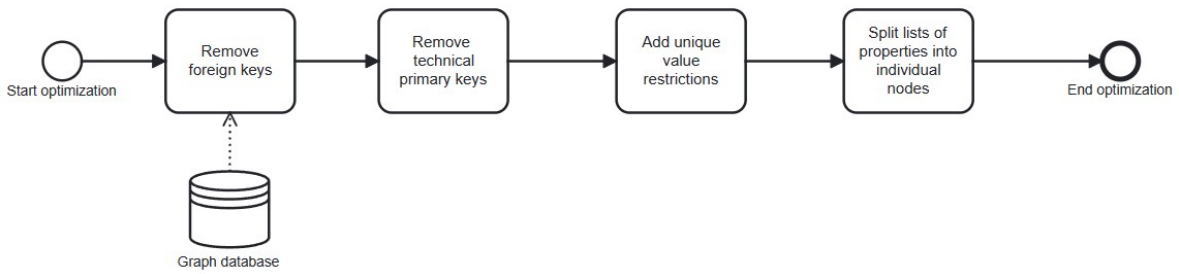

Figure 4 Graphical representation of the Optimization phase

B. *Loading the Data and Generating Relationships*

The steps described in Section 3B are explained in the Northwind database examples.

1. For data loading, the authors used the built-in function in the Cypher language. The data was loaded from CSV files, as Neo4j supports CSV files and provides simple commands.
2. A new node is generated for each record in the relational table. Each node is enriched with a label of the same name as the table whose data it contains. The Cypher code used for creating Orders from the file is given in Listing 2.

Listing 2 Cypher code for Orders node

```
1: load csv with headers
2: from 'file:///Orders.csv'
3: as row
4: merge (o: Order
5: {orderID: row.orderID})
6: on create set
7: o.shipName= row.ShipName;
```

3. The demonstration for specialization will be based on the simplified example shown in Figure 5. Attributes from specializations should be stored together with the attributes from their general objects. For example, a person with ID = 1001 should have both a name and a salary. Cypher language uses a MERGE statement for this purpose.

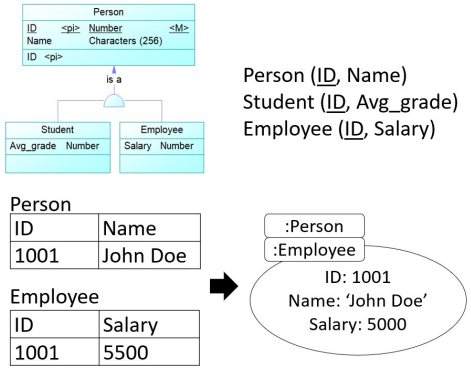

Figure 5 Example of specialization table migration - in a relational database, data is stored in two tables. In the graph database, a single node can store the same data

If the node with ID = 1001 and the Person label exists, it will be updated with an additional Employee label and the Salary property. The Cypher language code in Listing 3 can be used.

Listing 3 Cypher code for existing Person node

```
1: load csv with headers
2: from
3: 'file:///Employee.csv'
4: as row fieldterminator ';'
5: merge (pers: Person
6: {ID: row.ID})
7: on match set
8: pers.Salary = row.Salary,
9: pers: Employee
10: on create set
11: pers.Salary = row.Salary,
12: pers: Employee
```

If the node with ID = 1001 and the Person label does not exist (the Employee file is read before the Person file), the new node will be created. The node will contain ID = 1001, Person label, Employee label, and both the Name and Salary properties. The Cypher language code in Listing 4 can be used.

Listing 4 Cypher code for new Person node

```
1: load csv with headers
2: from 'file:///Person.csv'
3: as row fieldterminator ';'
4: merge (p: Person
5: {ID: row.ID})
6: on match set
7: p.PersName=row.PersonName
8: on create set
9: p.PersName=row.PersonName
```

Figure 6 shows the data used. Both the tables (relational model) and the nodes (graph model) are displayed. The graph shows four nodes with the Person label and two of them with the Employee label (in orange).

Person

ID	Name
1001	John Doe
1002	GI Joe
1003	Mr X
1004	Stewart Little

Employee

ID	Salary
1001	5500
1004	6200

Figure 6 Data for the specializations in the relational (left) and the graph (right) model

4. Foreign keys have been imported as regular properties. Now they can be used to properly connect the right nodes and implement relationships (Listing 5).

Listing 5 Connecting node Customer and node Order

```
1: match (c:Customer),(o:Order)
2: where c.ID = o.CustomerID
3: create (c)-[:orders]->(o)
```

5. The records of association tables are stored as nodes, not relationships. The association tables can reference or be referenced by two or more other tables. In Neo4j, relationships are formed between two nodes, and cannot be referenced. If the association tables were forced into becoming relationships, all table references beyond the first two would have to be stored as properties reminiscent of foreign keys. As this is not strictly according to the graph paradigm, it could create problems if keys are dropped while tables referencing them in the relational model endure.

The Employee Privileges is the only true association table. Order Details can be interpreted as an association table, the difference being that it has its own primary key (Figure 7). The proposed approach works with both types without the need for adaptation. Employee Privileges table can be converted using Cypher language code in Listing 6.

Listing 6 Conversion of EmpoyeePrivileges

```
1: match (e: Employee),
2: (ep: EmployeePrivileges),
3: (p: Privileges)
4: where e.ID= ep.EmployeeID
5: and ep.PrivilegeID =
6: p.PrivilegeID
7: create (e)-[:privileges]->
8: (ep)<-[:references]-(p)
```

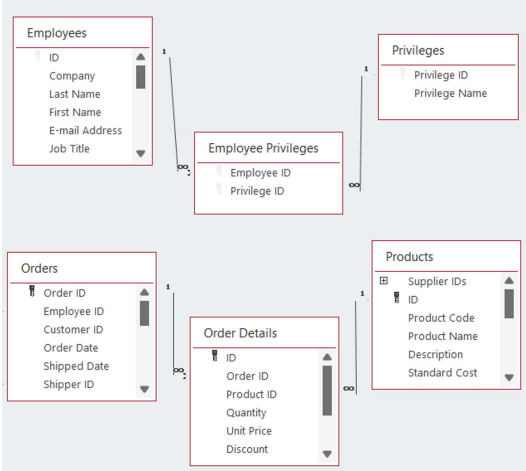

Figure 7 Example of association table - Employee Privileges and dependant table – Order Details

C. *Optimizing the Graph Database*

The optimization process includes the following steps:

1. All properties representing foreign keys can be deleted. They are already used to properly connect the right nodes and are no longer needed. When a new node is created for each record in the table Orders, foreign keys (Customer ID, Shipper ID) are deleted.
2. Neo4j distinguishes between natural primary keys and technical primary keys. As the graph model provides the technical primary keys, all technical keys are removed. Upon converting each record of the dependent table Order Details to a node, the technical primary key (ID) is deleted.
3. The ID properties have unique values, so unique value restrictions are added. The following Cypher code can be used to add a unique value constraint to the OrderID of the Order node (Listing 7).

Listing 7 Cypher code for unique constraint

```
1: create constraint orderCon
2: on (o: Order)
3: assert o.OrderID is unique
```

4. Splitting the list of properties into individual nodes is not required.

5. *Comparative Analysis with Existing Approaches*

To provide a more in-depth analysis, the authors have compared the proposed approach to several previous studies. More recent studies relevant to this research were taken into consideration. The basis of the comparison (shown in Table 1) is the number of relationship types for which a conversion method was provided in each approach.

Existing studies oftentimes use an ER model as a starting point for the conversion process. In many cases, the model does not exist or has not been accurately transferred to the relational schema. In some instances, technical primary keys are added, or the rules for constructing the relational model are not strictly followed. This could lead to unreliable conversions from a relational to a graph system. To address the above issue, the authors of this paper propose a conversion method that does not rely on the ER model.

Table 1 Comparison with existing studies
(Y – a rule for the relation type conversion is proposed)

RDB relationship	One-to-one	One-to-many	Many-to-many	Specialization
Proposed	Y	Y	Y	Y
[1]	Y	Y	Y	Y
[5]	Y	Y	Y	-
[9]	Y	Y	Y	-
[17]	Y	Y	Y	-

Table 1 presents the comparison between the proposed approach and the previous studies. It

demonstrates that the proposed approach covers a greater number of relationship types compared to studies referenced as [5], [9], and [17]. Additionally, compared to the study [1], the proposed approach handles the migration of specializations and many-to-many relationships differently. This section provides a detailed breakdown of the differences observed in each step of the conversion process among the mentioned approaches.

A. Preparation

Before migrating the data, it is important to examine the relational database and make any necessary modifications. As preparation for data migration, [5] creates an ER diagram to complete the mapping between metadata, that is, sorting out the relationships between tables. [17] focuses on eliminating data with default values and converting denormalized and duplicated data into separate nodes. In contrast, studies [1] and [9] conduct no specific preparation activities before data migration. It is unclear from the information provided whether these studies directly migrate the data as it is or if they employ other methods for preparation that are not mentioned.

In the proposed approach, the authors include table normalization as the first step in the preparation phase to preserve data integrity and prevent data loss. This helps in optimizing the structure of the tables by eliminating redundancy and ensuring data consistency. Furthermore, the proposed approach involves removing calculated values used for query optimizations. This step simplifies the data migration process by focusing on the key attributes for migration. As the authors propose the migration based on key attributes, primary and foreign keys should be identified, followed by dependent tables and specializations.

B. Loading the Data and Generating Relationships

The migration process to the graph system relies on the outcomes of the Preparation phase.

1. Load the data – no specific distinction was identified between the compared approaches for this step of the conversion process.
2. Map the tables – in studies [1] and [9], a new node is generated for each entity in the ER model during the migration process. However, [9] migrates entities that do not reference any other entities first, followed by entities that reference only the already migrated entities, and so on. This sequential approach helps in managing dependencies between entities during the migration. But it also introduces the challenge of circular references. If entity A references entity B, which in turn references entity C, and entity C references entity A, none of these entities can be imported due to the circular reference. Overcoming this issue would require an algorithm that detects and resolves circular references, which adds complexity to the migration process.

   In contrast, the approach proposed in this paper avoids situations where data cannot be imported due to circular references or when a node needs to be connected to another node that does not exist yet. This is achieved by loading the data before the connection process, ensuring that all required nodes are available for establishing relationships.

   In studies [5] and [17], a new node is created for each record in the relational table, and the table columns are transferred as node properties. The label of the node corresponds to the name of the table.

   The mentioned studies have different approaches to node creation and relationship establishment during the migration process. The proposed approach in this paper takes a specific stance on avoiding circular references and ensures the data is loaded before the connection process. Studies [5] and [17] share similarities with the proposed approach in terms of node creation.
3. Map the specialization tables – Studies [5], [9], and [17] do not explicitly describe any specific conversion activities for handling specializations in the migration process. It remains unclear whether these studies similarly migrate specialization tables or entities as other tables or if they employ alternative methods for conversion that are not mentioned.

   In the study [1], a node is created for each participating entity, with the general entity becoming a start node and the specialized entity becoming an end node. The ID of the start node is included as a node property in the end node. This approach does not fully utilize the capabilities of graph systems but instead adheres to the limitations of the relational system. Specializations are used when some tables of the same group have some varying columns; for example, each person has an ID and a name, but only some (employees) have a salary. Storing this data in a single table would result in many columns with missing values, which is inefficient. Graph systems do not have such limitations, as graph nodes with the same label can have different properties. Based on this understanding, the authors of this paper propose mapping specialization tables into nodes with multiple labels, thus maximizing the benefits of graph systems in terms of data representation and efficiency. Each node contains two labels: the label of the specialized node and the label of the general node. Following the example from Section 4B, this approach generates a single node with two labels (Person and Employee) instead of

having a Person node connected to an Employee node representing a single person.

4. Map one-to-one and one-to-many relationships - in [1], each entity is converted to a node, with an entity that has a minimum cardinality of 0 (one-to-one relationship) or an entity that is on the one side (one-to-many relationship) becoming the start node. The ID of the start node is then included as a node property in the end node. Similarly, in the study [9], a node is created for each entity, and foreign keys are used to connect the nodes and determine the direction of the relationships in the graph.

   In [17], foreign keys are utilized to form relationships between nodes and removed afterwards. Study [5] employs a similar conversion method to the approach proposed in this paper. A foreign key is used to create a directional relationship between the nodes in the graph. This ensures that the relationships between the entities in the relational model are accurately represented in the graph system.

5. Map the association table - In the study [1], the conversion of a many-to-many relationship follows a similar principle as the one-to-many relationship. However, the relationship itself is represented as a relationship property between the nodes in the graph system. In the study [9], the approach for handling many-to-many relationships involves creating a node for each entity and connecting them with two relationships in different directions.

   The association table resulting from a many-to-many relationship is transformed into a graph relationship in a study [17]. Study [5] proposes a method similar to [17], with the addition of an intermediate step. A connection table based on foreign keys is created, serving as an intermediate step in establishing the relationships between nodes.

   The aforementioned approaches retain the information from the many-to-many relationship as a relationship property. However, this paper suggests that mapping records of many-to-many relationship tables as nodes would be a more practical approach. True association tables, which solely represent many-to-many relationships, are uncommon. In many cases, these tables have technical primary keys (such as an ID or a counter) and are not directly dependent on any other tables. Therefore, mapping their records as nodes provides a more straightforward representation in the graph system and avoids potential future-proofing issues if the business requirements change and additional connections are needed.

C. Optimizing the Graph Database

In the conversion process from a relational model to a graph model, it is common to identify relational model elements that are no longer necessary and should be removed. The approaches discussed in studies [5] and [17] address this optimization step, while studies [1] and [9] mention no specific optimization activities after data migration.

In the study [5], foreign keys are utilized to establish directional relationships between different nodes in the graph model. During the conversion process, technical primary keys are removed, while natural primary keys are used to name the nodes in the graph. Once all the nodes and relationships have been converted, unique value constraints are added to certain fields. Similarly, in the study [17], unique constraints are added for the natural primary keys during the migration process, while all technical primary keys are removed. After forming the relationships in the graph system using foreign keys, the foreign keys themselves are removed.

In the proposed approach, the authors suggest removing foreign key properties when they are no longer required in the graph model, as the actual relationships between nodes define the connections. Additionally, the authors recommend removing all technical primary keys, as the graph model provides those on its own. However, it is important to maintain unique value restrictions for the ID properties, as they should still have unique values to ensure data integrity. Furthermore, if necessary, lists of properties can be split into individual nodes, which can enhance the graph model's flexibility and efficiency.

Overall, the proposed approach focuses on optimizing the graph model after data migration by removing unnecessary elements such as foreign keys and technical primary keys. This allows for a more streamlined and efficient graph model while still maintaining data integrity and uniqueness through appropriate constraints.

6. EXPERIMENTAL RESULTS

The presented approach was tested using Microsoft's Northwind sample database. The objective was to migrate both the data and the relationships from the MS Access relational model to a graph model implemented in Neo4j. This practical example served as a validation of the conversion process.

A. Approach Validation

To verify the data integrity of the graph database after the migration, the results of the conversion process are compared.

The number of records in relational tables in the original database is identical to the number of graph nodes in the converted graph database.

Consistency in the number of records and nodes demonstrates that the relational data is successfully converted and represented in the graph database without loss or distortion. This validation step provides confidence in the integrity of the converted graph database and confirms that the conversion process effectively maintains the data structure during migration.

To validate the accuracy of data conversion, data query results between the relational and graph databases were compared. Two pairs of equivalent queries were executed in both databases to verify the consistency of the results.

The first pair of queries is presented in Listing 8 and Listing 9. The corresponding results are presented in Figure 8. Listings showcase the executed queries, highlighting their equivalence between the relational and graph databases.

Listing 8 SQL code of the first validation query

```
1: SELECT c.ID, o.ID, od.ID
2: FROM Customer c INNER JOIN Order o
3: ON c.ID = o.CustomerID INNER JOIN
4: OrderDetails ON o.ID =  od.OrderID
5: ORDER BY c.ID, c.ID, od.ID;
```

Listing 9 Cypher code of the first validation query

```
1: match (c:Customer)-[]->(o:Order)
2: -[]-> (od:OrderDetails)
3: return c.ID, o.ID, od.ID
4: order by c.ID, o.ID, od.ID
```

Figure 8 displays the results obtained from executing these queries, demonstrating matching outcomes between the two databases. The queries provide identical results with the same number of records. The results are sorted so that the first few tuples can be directly compared between the databases.

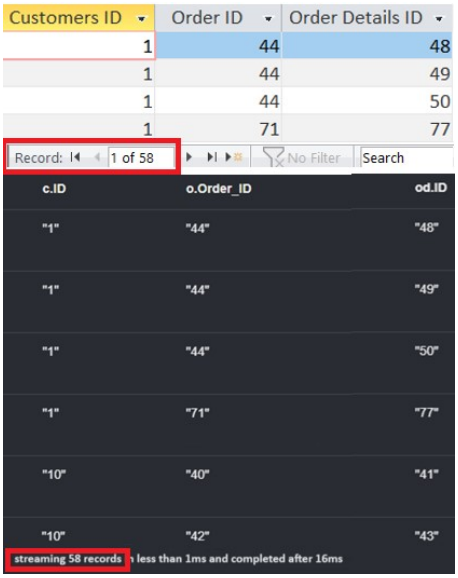

Figure 8 Results from equivalent queries show the same number of records and the same tuples in the results

In addition, a second pair of queries were executed to further validate the migration of the specialization table to the graph. The queries are presented in Listing 10 and Listing 11. The respective results are displayed in Figure 9.

Listing 10 SQL code of the second validation query

```
1: SELECT PersonName, Salary
2: FROM Person INNER JOIN Employee
3: ON Person.ID = Employee.ID;
```

Listing 11 Cypher code of the second validation query

```
1: match (e: Employee)
2: return e.PersonName, e.Salary
```

Figure 9 demonstrates that the results of the queries in both databases are identical, resulting in the same number of records. By successfully reproducing identical results for the queries related to the specialization table, the approach demonstrates its capability to handle and preserve specialized attributes during the conversion process.

Figure 9 Results from equivalent queries show the same number of records in the results

The execution of equivalent queries resulted in identical outcomes, confirming that the data details were accurately preserved during the transfer from the relational database to the graph database.

B. Comparison of Query Performance

To provide a comprehensive analysis, the authors compared the approach proposed in this paper to the approaches presented in previous work, specifically the approaches described in [1] and [5]. The approach discussed in [5] was selected as the most recent work in the field at the time of the study. Similar to the proposed approach, [1] offers conversion methods covering all relationship types. However, it addresses associations and specializations distinctively. This approach was selected to evaluate how variations in handling associations and specializations impact the performance of the generated graphs.

Table 2 displays the size of the generated graphs, quantified in terms of the number of

nodes. By adhering to the proposed approach, a notably higher number of nodes are generated. This discrepancy arises from the conversion of association tables.

Table 2 Size of generated graphs

Approach	This paper	[1]	[5]
Number of nodes	319	260	260

To facilitate the comparison between the approaches, the authors utilized the Northwind database available to them. Equivalent queries were executed on the resulting graphs generated by each approach to assess their performance and correctness. The initial testing query, displayed in Listing 12, is taken over from [5]. Subsequently, the second testing query, which relates to the dependent table Order Details, is showcased in Listing 13. The third testing query, referring to the specialization table Employee, is exhibited in Listing 14. The chosen queries serve the purpose of assessing the performance of the approach on different types of relational elements, considering the distinct guidelines provided by the approach for converting association and specialization tables. By executing this query on all graphs, the authors were able to compare the query results and evaluate the performance of each approach in terms of data retrieval.

Listing 12 The first testing query

```
1: match (p: Product{Category:
2: "Produce"})<--(s:Supplier)
3: return distinct
4: s.Company as ProduceSuppliers
```

Listing 13 The second testing query for Order Details node

```
1: match(o:Order)-[]->(od:OrderDetail)
2: <-[]-(p:Product)
3: where p.ProductName="Coffee"
4: return p.ProductName,
5: sum(od.Quantity)
```

Listing 14 The third testing query for Employee node

```
1: match (emp: Employee) where
2: emp.JobTitle='Sales Representative'
3: return emp.LastName, emp.FirstName,
4: emp.Salary
```

All approaches demonstrated identical query execution time of 2ms for the first query. This finding indicates that the proposed approach exhibits equally good performance compared to the approaches described in [1] and [5].

The second and third testing queries revealed better performance of the proposed approach. As depicted in Table 3, the execution time for these queries using the proposed approach was shorter in comparison to the approaches outlined in [1] and [5]. This finding underscores that the proposed approach leverages graph concepts, leading to enhanced performance outcomes.

Table 3 Query execution time (in ms) for the proposed approach and approaches presented in [1] and [5]

Approach / Query	(1)	(2)	(3)
This paper	2	4	2
[1]	2	5	3
[5]	2	5	3

The distinguishing factor of the presented approach is its conversion process, as highlighted in Section 5 of the paper. The authors propose a more straightforward representation of association tables in the graph system to address potential future-proofing issues. By avoiding the mapping of association tables as relationships with relationship properties, the approach ensures that any future changes in business requirements, such as the need for additional connections, can be accommodated more easily.

On the other hand, the approach described in [5] does not explicitly discuss any specific conversion activities for handling specializations in the migration process. In contrast, the proposed approach in this paper leverages the capabilities of graph systems by mapping specialization tables into nodes with two labels: one label representing the specialized node and another label representing the general node.

7. CONCLUSION

Considering the increasing volume of data and its dense connections, the need for efficiently and effectively migrating data from relational to graph systems is ever more present.

The proposed approach offers a method of transforming a relational database into a graph database without the need to consult a common, more abstract ER model. In many cases, the model does not exist or has not been accurately transferred to the relational schema.

A notable advantage of the proposed approach is its distinct method of converting relational model elements to a graph database. It leverages graph concepts by allowing nodes with the same labels to have different attributes, connections, and even multiple labels. This flexibility is not possible in traditional relational models. Moreover, the approach considers the specific characteristics of various relational elements, such as associations, specializations, and many-to-many relationships. It provides dedicated strategies to handle these elements during the conversion process, ensuring a comprehensive and accurate conversion.

The performance and accuracy of the compared approaches were evaluated by executing equivalent queries on the generated graphs. The queries were chosen considering the distinct guidelines for converting association and specialization tables. The results showcase identical results and better performance of the proposed approach compared to other

approaches, evident in shorter query execution times. The validation and comparison process confirmed the successful preservation of data during the conversion, showcasing the feasibility and enhanced performance of the proposed approach.

A. Future Work

To validate the proposed approach, the authors employed experimentation by comparing the query results obtained from the original database with the query results obtained from the target database format. This comparison helped assess the effectiveness of the proposed approach in maintaining the integrity of the data throughout the migration process. As a direction for future work, the authors consider validating the conversion process through formalization techniques.

Another area of future research involves extending the application of the proposed approach beyond the conversion from MS Access to Neo4j. The intention is to demonstrate the feasibility and effectiveness of the approach by applying it across a wider range of relational and graph databases. This will further emphasize that the proposed approach is not dependent on any particular database system and can be successfully adapted to various environments.

An additional area of future investigation is mapping data from various structured formats, such as XML, JSON, texts, or documents. This research would aim to extend the versatility and adaptability of the proposed approach. As a further extension of the proposed approach, the inclusion of application-specific logic is considered, namely business logic and triggers.

The successful preservation of data details indicates the potential for automation in the migration process, aiming to minimize the need for extensive human involvement and improve efficiency. The authors will attempt to fully automate the approach so that no human intervention is required. The next step is to automate the extraction of table metadata and use it to efficiently load, connect, and optimize the data.

REFERENCES

[1] ALOTAIBI, Obaid; PARDEDE, Eric. Transformation of schema from relational database (RDB) to NoSQL databases. *Data*, vol. 4, no. 4, p. 148, 2019.

[2] ALTIN, Ramazan; KINACI, A. Cumhur. Analyzing The Encountered Problems and Possible Solutions of Converting Relational Databases to Graph Databases. *Journal of Advanced Research in Natural and Applied Sciences*, vol. 8, no. 2, p. 281-292, 2022.

[3] CODD, Edgar F. A relational model of data for large shared data banks. *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[4] DE VIRGILIO, Roberto; MACCIONI, Antonio; TORLONE, Riccardo. Converting relational to graph databases. In: *First International Workshop on Graph Data Management Experiences and Systems*. p. 1-6, 2013

[5] FENG, Hui; HUANG, Meigen. An Approach to Converting Relational Database to Graph Database: from MySQL to Neo4j. In: *2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)*. IEEE, p. 674-680. 2022.

[6] Gabrovšek, P; Mihelič, J. Graph Covering and Subgraph Problems. *IPSI Transactions on Internet Research*, 2019.

[7] IBM, “Structured vs. unstructured data: What's the difference?,” *IBM*. [Online]. Available: https://www.ibm.com/cloud/blog/structured-vs-unstructured-data. [Accessed: 17-Dec-2022].

[8] Microsoft, “Get the sample SQL Server databases for ADO.NET code samples - ADO.NET,” *Get the sample SQL Server databases for ADO.NET code samples - ADO.NET | Microsoft Learn*, 21-Sep-2022. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/downloading-sample-databases. [Accessed: 14-Dec-2022].

[9] NAN, Zhihong; BAI, Xue. The study on data migration from relational database to graph database. In: *Journal of Physics: Conference Series*. IOP Publishing, vol. 1345, no. 2, p. 022061, 2019.

[10] Neo4j, “Model: Relational to graph - developer guides,” Neo4j Graph Data Platform. [Online]. Available: https://neo4j.com/developer/relational-to-graph-modeling/. [Accessed: 28-Dec-2022].

[11] OREL, Ognjen; ZAKOŠEK, Slaven; BARANOVIČ, Mirta. Property oriented relational-to-graph database conversion. *automatika*, vol. 57, no. 3, pp. 836–845, 2016.

[12] POKORNÝ, Jaroslav. Integration of relational and graph databases functionally. *Foundations of computing and decision sciences*, vol. 44, no. 4, p. 427-441, 2019.

[13] Ramachandran, S. Graph database theory. Comparing graph and relational data models, LambdaZen, 2015.

[14] Rodrigues, Cajetan; Jain, Mit Ramesh, Khanchandani, Ashish. Performance Comparison of Graph Database and Relational Database. 2023.

[15] SHAHZAD, Ahmad; COENEN, Frans. Automated Generation of Graphs from Relational Sources to Optimise Queries for Collaborative Filtering. *DBKDA 2020*.

[16] UGANDER, Johan, et al. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.

[17] UNAL, Yelda; OGUZTUZUN, Halit. Migration of data from relational database to graph database. In: *Proceedings of the 8th International Conference on Information Systems and Technologies*, p. 1-5, 2018.

Marija Đukić is a Teaching Associate at the University of Belgrade, Faculty of Organizational Sciences. Her areas of research are business analytics, ERP systems, and process mining (corresponding author – e-mail: marija.djukic@fon.bg.ac.rs).

Ognjen Pantelić is an Associate Professor at the University of Belgrade, Faculty of Organizational Sciences. His areas of research are ERP systems and process mining (e-mail: ognjen.pantelic@fon.bg.ac.rs).

Ana Pajić Simović is a Teaching Assistant at the University of Belgrade, Faculty of Organizational Sciences. Her areas of research are relational databases, ERP systems, business process modeling, and process mining (e-mail: ana.pajic.simovic@fon.bg.ac.rs).

Stefan Krstović is a Teaching Assistant at the University of Belgrade, Faculty of Organizational Sciences. His areas of research are databases and process mining (e-mail: stefan.krstovic@fon.bg.ac.rs).

Olga Jejić is a Teaching Assistant at the University of Belgrade, Faculty of Organizational Sciences. Her areas of research are event sourcing, event and relational databases, business process modeling, and process mining (e-mail: olga.jejic@fon.bg.ac.rs).