Vision Paper: Improving the Accessibility of Port Operations in Supply Chain Management using Graph Data Analysis

Mert Ayas
Westphalian UAS
Gelsenkirchen, Germany
mert.ayas@w-hs.de

Frank Laarmann
Westphalian UAS
Gelsenkirchen, Germany
frank.laarmann@w-hs.de

Leif Meier
Westphalian UAS
Gelsenkirchen, Germany
leif.meier@w-hs.de

Katja Zeume
Westphalian UAS
Gelsenkirchen, Germany
katja.zeume@w-hs.de

ABSTRACT

In supply chain management of port operations querying and visualizing network data often requires complex joins, nested queries and predefined reporting templates, which can hinder exploratory analysis and decision-making. When relational databases become too rigid and cumbersome for transactional processing, we envision an alternative approach using graph data models, provided by transforming entities (e.g. containers, vessels, terminals) into nodes and their relationships (e.g. arrival, loading, handling) into edges. To investigate the potential benefits of transforming relational supply chain data into a graph-based model, we designed and implemented a structured approach that integrated data processing, transformation and performance analysis across both relational and graph databases.

PVLDB Reference Format:
Mert Ayas, Frank Laarmann, Leif Meier, and Katja Zeume. Vision Paper: Improving the Accessibility of Port Operations in Supply Chain Management using Graph Data Analysis. PVLDB, 14(1): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

Supply chain management and port operations generate highly interconnected, dynamic data. As critical nodes in global logistics networks, ports must handle large volumes of containers with varying statuses, origins, destinations and modes of transport. Tracking these containers on a global route through several ports relies heavily on relational databases. These systems typically manage container information through structured tables containing timestamps, statuses, and references to carriers or vessels.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

While relational databases are robust for transactional processes, they can become rigid and cumbersome when tasked with representing the real-world complexity of supply chain networks, like route tracing, identifying optimal transfer points, and detecting inconsistencies [5]. The process of querying and visualizing such data often requires complex joins, nested queries and predefined reporting templates, which can hinder exploratory analysis and decision-making. In addition, these limitations can negatively impact the usability of data tools for non-technical users, limiting access to critical insights.

On top of that, the dynamic lifecycle of container handling involves temporal phases, such as:

(1) **Pre-Vessel-Arrival**: Ensuring containers scheduled for loading are in the yard before the vessel arrives.
(2) **Operations Phase**: Coordinating discharge and load operations in parallel while the vessel is berthed.
(3) **Post-Vessel-Departure**: Managing the dwell time and further movement of discharged containers, including those earmarked for transshipment onto other vessels.

These operational phases and the transitions between them create dense networks of events, dependencies and movements that are difficult to visualize and explore intuitively using relational data alone. Identifying patterns - such as container turnaround times, ship-to-ship flows, or yard utilization bottlenecks - requires not only data querying, but also relationship tracing and temporal pattern recognition.

Graph data models (see e.g. [1]) provide an alternative approach by transforming entities (e.g. containers, vessels, terminals) into nodes and their relationships (e.g. arrival, loading, handling) into edges. This structure allows for a more natural exploration of complex relationships, such as identifying transshipment containers from vessel *A* to vessel *B*, analyzing container lifecycles within the port, or visualizing capacity utilization over time.

In this paper we present our vision for transforming relational port operations data into graph-based models that can improve the visualization, understandability, usability and accessibility of supply chain and port management data. The aim is to explore whether graph data analysis can enable both technical and operational stakeholders to better understand the complexity of port operations, make faster data-driven decisions and improve overall operational awareness.

2 RELATED WORK

In a previous study [3], Feng and Huang proposed a structured approach to converting relational database schemas into graph-based representations. Their method divides this process into three stages. The first stage involves analyzing metadata to identify and understand the relationships between tables. The second stage focuses on converting entities, which are typically represented as rows in relational tables, into graph nodes. The final stage transforms foreign key relationships into edges between nodes.

Furthermore, an introductory examination of graph database principles is provided by Miller [7], with a particular focus on Neo4j. They work emphasizes the limitations of traditional relational databases when managing complex, highly interconnected data structures. In such structures, multiple joins can lead to performance bottlenecks. In contrast, graph databases, such as Neo4j, are shown to offer clear advantages in scenarios requiring efficient relationship traversal.

Similar studies [4, 9] have evaluated the performance, memory usages, and time complexity of relational databases (such as MySQL and PostgreSQL) and NoSQL databases across different data volumes. Findings from [9] indicate that, despite similar theoretical time complexity for certain operations, MySQL often demonstrates faster query execution times in practice.

Numerous studies in the field of supply chain management have explored the potential benefits of using graph structures to represent data instead of traditional relational models. In [5, 6] they argue that graph-based approaches are well-suited to capturing the complex, interconnected nature of supply chains such as RFID data. In such chains, entities such as suppliers, transport links, products, and distribution centers are deeply interrelated. By modeling these elements as nodes and their interactions as edges, graph databases can offer more intuitive insights and enable more intuitive queries.

Antony et al. [2] use a graph data model to help identify risks in supply chain management more effectively, manage logistics, and detect fraud. This line of research highlights the importance of graph technologies, such as Neo4j, in modern supply chain analysis. Nevertheless, more empirical case studies in different sectors are needed to quantitatively evaluate the impact of graph-based methods on manufacturing resilience [8].

3 A STRUCTURED APPROACH TO SCM ANALYSIS USING GRAPH DATA

We explore realistic SCM data from real-world port operations in Section 3.1. In Section 3.2. we explain our graph model for the port operations data. We present a structured transformation from relational data to a graph data model in Section 3.3.

3.1 The Relational Data Model for Supply Chain Management

The relational data model for one port is shown in Figure 1. It supports the analysis of port operations and container flows, and is structured around key relational tables representing entities such as containers, carriers, modes, services, liners and flexible attribute options. The model captures dynamic events (such as arrival and departure times), enabling a wide range of operational queries and analyses.

In addition, the data model supports vessel-centric analysis by enabling queries that associate containers with specific vessels. It allows analysts to observe whether the number of containers loaded or unloaded matches the carrier's declared plans, and to classify operations into different phases. Before the ship arrives, all the containers to be loaded must already be in the yard. During the ship's stay, the terminal has to manage the parallel operations of discharging the arriving containers and loading those scheduled for departure. After the ship has departed, any remaining discharge containers should be moved out of the yard, unless they are transshipment containers waiting for their next carrier.

Operational questions such as "How does the volume of containers change over the course of the day?" or "Are there any service/-liner combinations for which the expected export container dwell times are above average?" are answered by SQL queries using this relational model. However, the complexity of these queries increases significantly as analysts attempt to understand container movements over time, identify handoffs between carriers, or analyze capacity utilization patterns over smaller time intervals. Queries often involve multiple joins and conditions, requiring technical expertise and limiting the accessibility of the data to non-technical stakeholders.

3.2 The Graph Data Model

Graph-based models offer a compelling alternative, also for non-technical stakeholders. Figure 2 shows the graph data model for port operations data for one port. By transforming the relational data into a graph structure, where containers, carriers, services and modes are represented as nodes and their relationships as edges, it becomes possible to more naturally visualize and explore the complex network of container flows and vessel operations. This transformation not only improves comprehension and accessibility, but also supports the identification of patterns such as transshipment flows, bottlenecks and operational inefficiencies that are difficult to identify using traditional relational queries alone.

An overview of the modeled graph in Neo4j is shown in Figure 3 which shows all nodes directly connected to a specific container node via a single edge. It also shows how these nodes are linked to each other and if they share any direct relationships. The result provides a clear view of the container's immediate network and the structure of its surrounding connections. The output shows that the container is directly linked to one inbound and one outbound carrier. Additionally, both carriers consistently share the same service and liner connections with the container, revealing a clear, uniform structure to these relationships. This result is presented in the form of an effective graph visualization. Any inconsistency caused by a container being linked to a different service or liner than its carriers would be immediately apparent. This makes it easy to spot modeling errors or data issues at a glance.

3.3 Transforming SCM Data using Neo4j

To investigate the potential benefits of transforming relational supply chain data into a graph-based model, we design and implement a structured approach that integrates data processing, transformation and performance analysis across both relational and graph databases.

Figure 1: Relational Schema for Port Operations Data

Carrier

Feld	Typ
id (PK)	Long
modeId	Long
name	String
eta	LocalDateTime
etd	LocalDateTime
load	Long
discharge	Long
restow	Long
fromPosition	Long
length	Long
serviceId	Long (FK to ServiceOption)
linerId	Long (FK to LinerOption)

Container

Feld	Typ
unitName (PK)	String
timeIn	LocalDateTime
timeOut	LocalDateTime
attributeSizes	Long
attributeStatus	Long
attributeTypes	Long (FK to AttributeOption)
attributeServices	Long
attributeDirections	Long
attributeWeights	Long
inboundModeId	Long (FK to ModeOption)
outboundModeId	Long (FK to ModeOption)
inboundCarrierId	Long (FK to Carrier)
outboundCarrierId	Long (FK to Carrier)
inboundServiceId	Long (FK to ServiceOption)
outboundServiceId	Long (FK to ServiceOption)
inboundLinerId	Long (FK to LinerOption)
outboundLinerId	Long (FK to LinerOption)

AttributeOption

Feld	Typ
id (PK)	Long
attributeId	Long
description	String

LinerOption

Feld	Typ
id (PK)	Long
name	String

ServiceOption

Feld	Typ
id (PK)	Long
name	String

ModeOption

Feld	Typ
id (PK)	Long
name	String

Figure 2: Graph Data Model for Port Operations Data

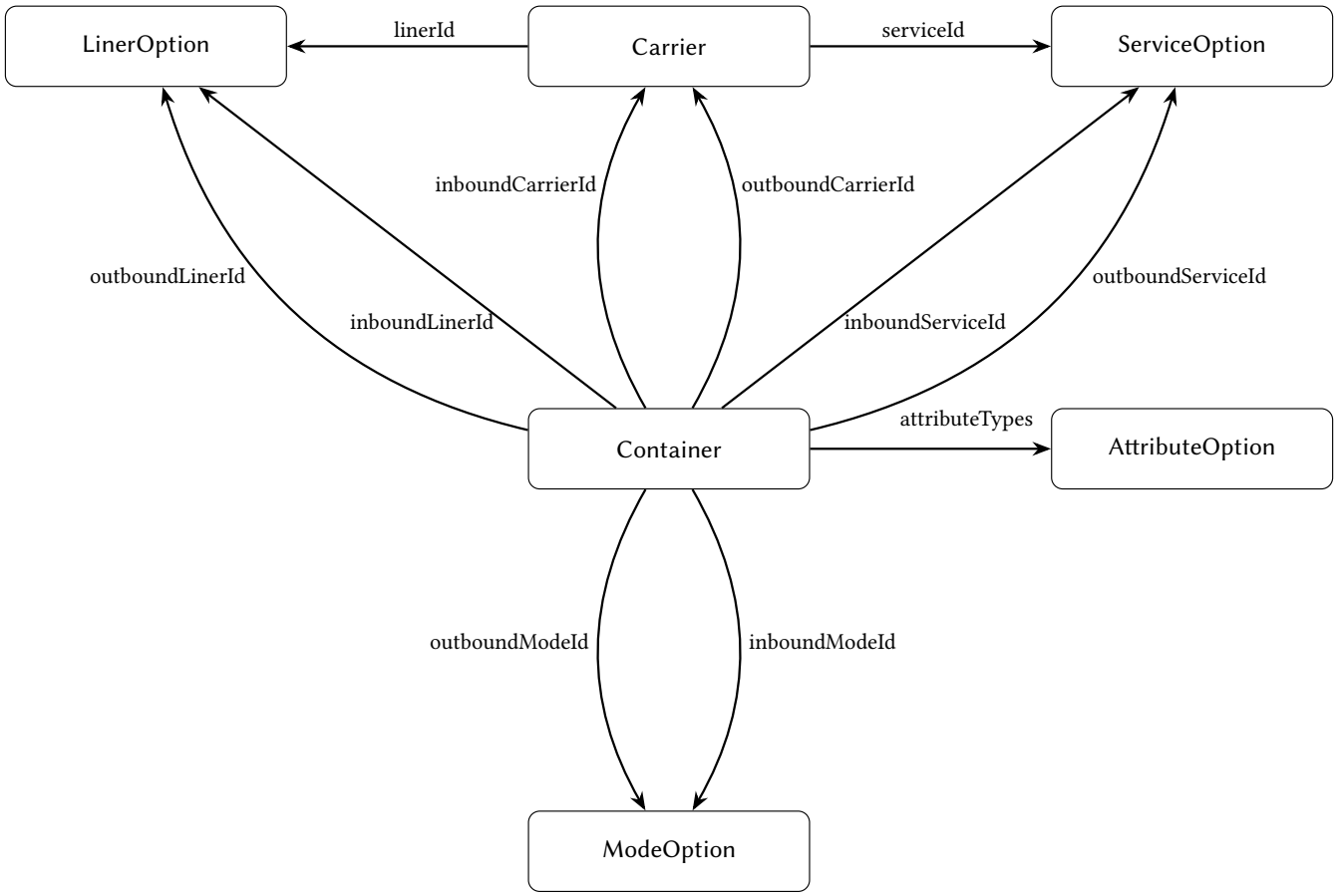


Figure 3: Details of Container 02b30A9iogS (extracted with Neo4j desktop)

Our process starts with preparing the data environment. Existing relational supply chain and port operations data, see Figure 1, is imported into a relational MySQL database. This ensures that the data reflects its original transactional structure, including all relevant entities, attributes and relationships. MySQL serves as baseline system for comparison, representing the conventional approach of handling such data.

Next we transform and load the relational data into a graph database environment. We use Neo4j, a widely used graph database system optimized for highly connected data. A dedicated Java application using Spring Boot handles data extraction, transformation, and loading (ETL) processes. The application reads data from the MySQL database, transforms it according to graph modeling principles, and writes it to Neo4j. The transformation logic is based on retrieving complete data for each table via SQL queries. The entities and corresponding graph nodes are modeled inside the Spring Boot application. Rather than treating foreign key values as simple attributes, they are mapped directly to relationships between nodes in the node classes, aligning the structure with Neo4j's native graph paradigm.

In determining the appropriate transformation logic, we follow Neo4j's recommended methodology for migrating relational data to graph models which is similar to the approach described in [3]. Specifically, all primary entities such as containers, carriers, services and liners are modeled as nodes. The existing foreign key relationships in the relational model are translated directly into explicit relationships between nodes in the graph model. For example, the relationship between a container and its in- and outbound carriers is represented as graph relationships connecting the respective nodes. This approach ensures that the inherent relationships present in the relational model are preserved and made navigable within the graph database.

To enable a direct comparison of the performance of queries and data exploration in the two systems, the Spring Boot application is built to support executing both MySQL and Neo4j queries. This setup permits controlled experiments in which identical queries are formulated in SQL and Cypher, and executed through the same interface. Each query is executed a predetermined number of times to collect consistent performance data and mitigate the impact of caching and other runtime factors. The MySQL and Neo4j servers are hosted on separate virtual machines running Ubuntu 24.04 to ensure consistent, isolated environments for fair benchmarking. Our study focuses solely on query execution time in milliseconds as a consistent metric for comparing performance across systems. CPU, memory, and disk I/O usage are not included, though these can be measured separately using system monitors or database profiling tools for a more in-depth analysis.

4 ANALYZING THE FUTURE SCM GRAPH DATA MODEL

This section examines two key aspects of the system: data visibility and query performance. By analyzing how information can be intuitively accessed and how efficiently queries are executed, we can better understand the practical strengths and limitations of each database management approach.

For example, the standard query in Figure 4 returns a simple overview in Neo4j, as shown in Figure 3. This query returns all nodes directly connected to a specific container node via a single edge. It also shows how these nodes are linked to each other and if they share any direct relationships. The result provides a clear view of the container's immediate network and the structure of its surrounding connections. This query's result is the immediate network around each container. The output shows that the container is directly linked to one inbound and one outbound carrier. Additionally, both carriers consistently share the same service and liner connections with the container, revealing a clear, uniform structure to these relationships. This result is presented in the form of an effective graph visualization. Any inconsistency caused by a container being linked to a different service or liner than its carriers would be immediately apparent. This makes it easy to spot modeling errors or data issues at a glance.

Figure 4: A Cypher Query to generate a Container Overview

```
MATCH(n:Container{unitName:"02b30A9iogS"})-->(x)
RETURN n,x
```

This doesn't mean that MySQL can't produce the same result. It can, but achieving the same level of visibility requires a more complex query. In this case, we would need to join the container table with the carrier table twice: once for the inbound carrier and once for the outbound carrier. Then, we would need to compare the serviceId and linerId of both carriers to ensure they match the values documented in the container attributes. While this approach is possible, it adds more steps, which can make it harder to quickly interpret the relationships. Extracting the same information from MySQL requires multiple redundant joins across all tables in the schema. This increases query complexity and makes the results

harder to interpret, see e.g. Figure 5. Issues such as mismatched liners or services must be identified manually in the result set because the relationships aren't as visually or structurally apparent as they are in a graph model.

Figure 5: Details of Container 02b30A9iogS (MySQL)

Field	Value
container	02b30A9iogS
inbound_liner	Twilight Beacon
inbound_carrier_liner	Twilight Beacon
outbound_liner	Oceanic Crest
outbound_carrier_liner	Oceanic Crest
inbound_service	Flicker Line
inbound_carrier_service	Flicker Line
outbound_service	Albatross Wing
outbound_carrier_service	Albatross Wing
inbound_carrier	Celestial Horizon
outbound_carrier	Emerald Skies
inbound_mode	Feeder
outbound_mode	Deepsea
attribute_type	dry

The differences in query performance between MySQL and Neo4j are shown in Figure 6. This graph illustrates three stages of complexity. All three queries aim to identify specific combinations of shipping services and liners (defined by outboundServiceID and outboundLinerId) associated with above-average dwell times for export containers in the port. Dwell time, which is calculated as the difference between timeOut and timeIn, reflects how long containers remain in the terminal. By focusing on export containers (i.e., inboundModeId > 2 and outboundModeId <= 2), the query examines whether certain service/liner combinations result in containers consistently arriving much earlier than necessary for loading, thereby occupying storage space longer than required. Identifying such patterns could inform contract adjustments or operational planning by highlighting combinations that contribute disproportionately to terminal congestion.

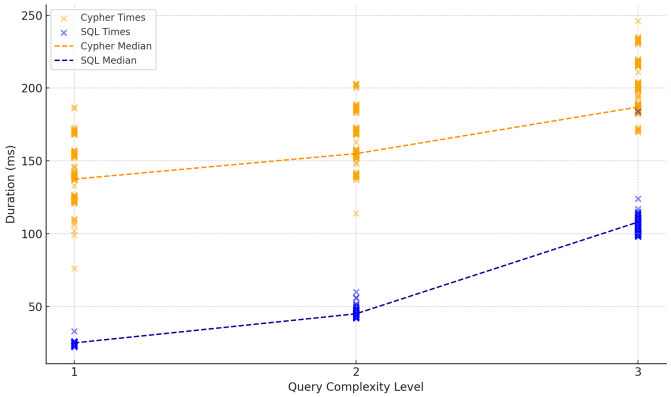
The complexity stages are set as follows:

(1) The query filters by mode IDs to find relevant service and liner combinations and returns only their IDs along with dwell times.
(2) The query filters by the names of the modes to find relevant service and liner combinations and returns their names along with the dwell times.
(3) This query builds on the second one, ensuring that the carrier has not yet arrived at the port and is associated with the same liner and service as the export container.

For Neo4j, all three queries have the same level of complexity and don't require any specific extensions. However, the second query for MySQL requires specific joins to access the names of the modes, services, and liners. The third query builds on the second by ensuring that the carrier has not yet arrived at the port and is linked to the same liner and service as the export container. These conditions are established through nested subqueries.

In Figure 6 we observe that as query complexity increases, the performance gap between Neo4j and MySQL narrows. However, Neo4j can handle complex queries with relatively simple patterns, while MySQL requires increasingly nested joins to produce the same results.

Figure 6: Query Execution Time Distribution



Looking ahead, there are several ways to extend this project to improve performance and analytical depth. As queries become more complex, we want to explore wheter Neo4j can leverage its strength in relationship-heavy data structures and outperforms the relational model.

One key opportunity lies in rethinking how certain attributes are modeled. Currently, nodes such as AttributeOption, LinerOption, ServiceOption, and ModeOption hold only a small amount of information, typically just two attributes. Since these nodes are linked to containers and carriers every query involving them requires additional traversal steps. This adds complexity and overhead without offering much structural benefit. A more efficient approach could be to remove these lightweight nodes and store their information as attributes directly on the relationships between nodes instead. For instance, rather than connecting a container and a carrier to shared ServiceOption and LinerOption nodes, the relationship between them could include the serviceId and linerId as attributes. This approach would reduce the number of nodes and edges in the graph, streamlining the data model and simplifying many queries, particularly those that only need to filter or compare basic option values. It would also eliminate the need to manage multiple edges in simple lookups, making the model easier to work with.

In addition to improving the model itself, one area for expansion is to increase the scope of the data. Currently, the dataset only represents operations within a single port. In the future we want to construct realistic sequences that simulate container movement across multiple ports. This will allow to model a more complete logistics chain and demonstrate how containers move over time and under different conditions around the world. Such a model will open up new possibilities for analysis as current databases are usually restricted to one port only. In this way, we can explore how graph databases perform when querying on longer paths compared to traditional, join-heavy SQL queries. Potential use cases are testing

how effectively routes can be traced, optimal transfer points are identified, and surface delays or inconsistencies are detected. In addition, we envision to understand a container's entire journey, including its origin, destination, and intermediate stops, simply by following its connected nodes.

5 CONCLUSION

We propose a novel approach to port operations in supply chain management, utilizing a graph data model to achieve a more comprehensive understanding of the complex interrelationships among different entities and their interactions. In this study, we employ authentic SCM data derived from actual port operations to conduct a comparative analysis of the relational data schema and a graph data model that utilizes the same set of data. In Section 3.3, an automated transformation is employed to translate the existing SCM data into a Neo4j database. In Section 4, we undertake a foundational examination of potential use cases for port operations and their query complexity. This analysis is conducted with the objective of demonstrating the merits of graph modeling. Although relational databases are nowadays the clear choice, the results across the presented complexity stages suggest a possible shift in query efficiency. As queries and data structures become more complex, Neo4j may have an advantage. However, this alone is not enough to confirm that Neo4j will consistently outperform MySQL under more complex conditions. Similar tests should be conducted in the future on larger, more complex datasets to enable drawing reliable conclusions from additional queries.

REFERENCES

[1] Renzo Angles and Claudio Gutierrez. 2008. Survey of graph database models. *ACM Comput. Surv.* 40, 1, Article 1 (Feb. 2008), 39 pages. https://doi.org/10.1145/1322432.1322433

[2] Agnes Antony, P Alaka, and B Tulasi. 2025. Enhancing Supply Chain Management through Graph Analytics. *IJSAT-International Journal on Science and Technology* 16 (2025).

[3] Hui Feng and Meigen Huang. 2022. 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)*. 674–680. https://doi.org/10.1109/ICPECA53709.2022.9719151

[4] Marin Fotache, Nicoleta Teacă, Codrin-Stefan Eşanu, Marius-Iulian Cluci, Ciprian Pinzaru, and Paul Gasner. 2024. Query Performance Comparison of PostgreSQL vs. Neo4j. A Basic Distributed Setup on OpenStack. In *2024 23rd RoEduNet Conference: Networking in Education and Research (RoEduNet)*. 1–6. https://doi.org/10.1109/RoEduNet64292.2024.10722626

[5] Young-Chae Hong and Jing Chen. 2022. Graph database to enhance supply chain resilience for industry 4.0. *International Journal of Information Systems and Supply Chain Management (IJISSCM)* 15, 1 (2022), 1–19.

[6] Chun-Hee Lee and Chin-Wan Chung. 2008. Efficient storage scheme and query processing for supply chain management using RFID. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 291–302.

[7] Justin J Miller. 2013. Graph database applications and concepts with Neo4j. In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, Vol. 2324. 141–147.

[8] Erwin Rauch, Ali Asghar Bataleblu, Michaela Golser, Asja Emer, and Dominik T Matt. 2024. Potential of Graph Database Visualization of the Supplier Network to Increase Resilience in Multi-tier Supply Chains. In *International Scientific-Technical Conference MANUFACTURING*. Springer, 125–139.

[9] Rahmatian Jayanty Sholichah, Mahmud Imrona, and Andry Alamsyah. 2020. Performance Analysis of Neo4j and MySQL Databases using Public Policies Decision Making Data. In *2020 7th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*. 152–157. https://doi.org/10.1109/ICITACEE50144.2020.9239206