

Method for developing and partitioning graph-based data warehouses using association rules

Redouane Labzioui, Khadija Lettrache, Mohammed Ramdani

LIM Laboratory, Faculty of Sciences and Techniques of Mohammedia, University Hassan II, Casablanca, Morocco

Article Info

Article history:

Received Mar 23, 2024

Revised Oct 17, 2024

Accepted Oct 21, 2024

Keywords:

Association rules

Business intelligence

Graph warehouse

Graph-oriented-databases

Not only structured query language

ABSTRACT

The evolution of modern databases has led to a variety of not only structured query language (NoSQL) models, particularly graph-oriented-databases. This growth has encouraged businesses to explore graph-based business intelligence (BI) solutions. This paper explores three essential aspects in the domain of graph warehouse: the establishment of efficient graph warehouses, the significance of data historization, and the development of effective strategies for graph partitioning. It starts by building a BI system within a graph database. Subsequently, the paper emphasizes the pivotal role of data historization, highlighting the slowly graph changing dimension (SGCD) approach as a versatile framework for accommodating varied dimensional changes, additionally; the paper introduces a novel partitioning strategy utilizing association rules algorithms, for optimized and scalable graph warehouse management.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Redouane Labzioui

LIM Laboratory, Faculty of Sciences and Techniques of Mohammedia, University Hassan II

Casablanca, Morocco

Email: redouane.labzioui-etu@etu.univh2c.ma

1. INTRODUCTION

Expanding data sets have significantly altered the landscape of modern databases, particularly the emergence and expansion of various not only structured query language (NoSQL) models like document, column, key-value, and graph databases [1], [2]. Among these, graph-oriented databases have garnered substantial attention, they offer a unique abstraction to handle densely connected data, allowing for complex domain modeling and execution of intricate queries [3], [4]. The increased attention has led to a number of businesses starting projects to develop business intelligence (BI) solutions with graphs [5], [6].

However, data historization-the process of conserving data across time-is becoming more and more necessary due to the increasing complexity of data. For the purpose of monitoring changes over time, examining past trends, adhering to legal obligations, and making wise decisions [7]. In graph-based data warehouses, where dimensions are frequently used to describe entities within the graph, this becomes very important, understanding the evolution of the graph and its components becomes difficult in the absence of data historization [8].

In addition to historization, this paper addresses another critical challenge: data partitioning within these graph-based data warehouses. Graphs, as complex structures, require a nuanced approach to effectively manage partitions [9]. Well-managed partitions are essential for optimizing queries, ensuring efficient storage, and enabling quick access to relevant data [10], [11]. Therefore, this paper aims to concurrently address three crucial aspects: the creation of efficient data warehouses under graph database systems, the importance of data historization, and the effective management of data partitions.

First, we construct a BI system within a graph database. We then examine the importance of data historization, focusing on the slowly graph changing dimension (SGCD) approach, which is adaptable to different types of dimensional changes over time. This focus on historization addresses the gap in previous research by detailing strategies for tracking and managing changes over time in graph data warehouses. Finally, we explore strategies and algorithms based on association rules for effectively managing partitions in the context of graph-based data warehouses.

The rest of this document is structured as outlined below: section 2 explores the existing literature on graph data warehousing. In section 3, we provide the necessary background of the approach. Section 4 delves into the implementation of the strategy. Section 5 discusses the principal results of the study. Finally, this work is concluded in section 6, which also suggests options for future research.

2. RELATED WORKS

In the literature, discussions about data partitioning and historization have predominantly focused on relational data warehouses, with a limited exploration into graph warehouse. However, recent years have witnessed a growing interest in integrating BI technology with graph databases, resulting in various proposed methodologies documented in the literature. Zhao *et al.* [12] have presented a brand-new idea known as the graph cube, that is a new data warehouse paradigm intended to handle multidimensional queries inside large-scale multidimensional networks. This approach organizes dimensions based on node attributes while employing computed measures to aggregate these node attributes, authors did not address the partitioning and historization aspect within the graph warehouse. Castelltort and Laurent [13] introduced a methodology suggesting the use of graph structure for online analytical processing (OLAP) queries, leveraging the performance of the graph database for storage and query time processing. This approach involves converting measures and dimensions into nodes within the graph, using arcs to establish relationships between dimensions and measurements. Hierarchical interactions among nodes retain hierarchical dimensions. However, this strategy is confined solely to the snowflake model, potentially limiting its applicability to other data models.

Muttipati and Padmaja [14] offer a comprehensive overview of existing tools and approaches for graph partitioning and frequent sub-graph extraction. It explores a wide array of topics, encompassing graph-partitioning techniques, frequent sub-graph mining algorithms, parallel processing frameworks, and managing substantial volumes of graph data. However, it is worth noting that the study might lack in-depth experience and empirical results to substantiate its findings fully. While its broad scope provides a wealth of information, it might fall short in thoroughly examining specific aspects or offering detailed insights into certain algorithms or parallel frameworks. Additionally, it does not delve into partitioning of the graph warehouse, limiting its exploration to certain areas within the realm of graph databases. Dai *et al.* [15] introduced the incremental online graph partitioning (IOGP) algorithm, aimed at addressing performance needs in distributed graph databases. IOGP dynamically adjusts through three operational stages, efficiently accommodating continuous graph changes. It swiftly produces optimized partitioned graphs, proficiently serving complex traversals. Implementation details, including in-memory data structures like edge counters, facilitate rapid online graph partitioning. Extensive evaluations across diverse graphs affirm IOGP's advantages, aiding in establishing key parameter selection guidelines. Akid *et al.* [16] suggest guidelines for transforming a multidimensional data model into a graph data model (MDM2G) and compare the performance of snowflake model and star designs in both graph databases and relational, focusing on size and dimensionality. Their comparison suggests that employing a graph-based implementation for a multi-table data warehouse has greater efficiency than a relational approach. Moreover, within graph databases, a star model demonstrates similar performance to a snowflake model.

Andriamampianina *et al.* [17] suggested a conceptual framework for temporal graphs depicting evolving graph data. It captures how data changes over time without sacrificing information or introducing redundancy, which differs from how snapshot-based models handle this evolution. Firstly, it is tailored towards business needs, offering non-expert users a complete understanding of data and its evolving elements. Secondly, it is versatile, capable of representing various types of changes in graph data; including topology shifts and alterations in attribute sets and values. Lastly, it accurately captures the temporal evolution of data, preserving information and avoiding redundancy, unlike snapshot-based models. However, when applied to large datasets, temporal context analysis and manipulation can occasionally result in slower processing times and reduced performance. Moreover, temporal graphs can quickly grow in size with added data over time. Managing these large datasets can be costly in terms of storage and processing, requiring additional resources. Zhou *et al.* [18] describe a novel method for representing data and queries using graph topologies, providing a distinctive way to visualize column correlations. It chooses partitioning keys by utilizing graph embedding's, which may increase the effectiveness of data partitioning. It also suggests a learning assessment model that estimates the performance of partitioning strategies without physically dividing data, potentially conserving computing power. Benhissen *et al.* [19] suggest a method centered around a progressive schema model

featuring multiple versions, where a graph data warehouse accommodates data instances aligned with distinct schema versions. The handling of these versions is made easier using a dedicated meta-model for warehouse schema versions, along with the integration of evolutionary functions introduced at the schema level. Nevertheless, the research does not delve into the aspect of OLAP cube partitioning within graphs.

It is clear from the previous overview that the majority of these papers primarily concentrate on transforming classical data warehouse into graphs. This emphasis could potentially restrict the thorough exploration and utilization of the complete benefits provided by graph databases. Additionally, these studies overlook the aspect of historical data and partitioning graph warehouses, creating another significant gap in their coverage. In contrast, our suggested method makes advantage of graph features to build graph warehouses, addresses also the aspect of data historization, and uses association rules for partitioning graph warehouses.

3. OUR APPROACH

Our method entails first building a graph warehouse, with a particular emphasis on the SGCD method for data historization. This method is specifically designed to manage changes in graph dimensions over time, ensuring that the historical evolution of data is accurately captured. The SGCD enables the warehouse to maintain a comprehensive historical record of the dimensions, which is crucial for analyzing trends and performing time-based queries [20]. Subsequently, we use user OLAP queries as input for two crucial algorithms: The Apriori algorithm [21], which identifies frequently occurring itemsets in the graph cube, and a rule-based association algorithm is employed to detect the most used partitions [22]. The primary objective of this technique is to optimize our OLAP cube for maximum performance, ensuring efficient handling of user queries and data analysis. There are four steps in this approach:

- Building a graph warehouse.
- Integrate a data historization component into the process, emphasizing the SGCD method, this step ensures a comprehensive view of data changes over time, highlighting adaptability to various dimensional modifications.
- Utilizing user queries as input for the Apriori algorithm to identify frequently occurring itemsets within the graph cube.
- Using a rule-based association algorithm to identify the most frequently used partitions.

3.1. Graph warehouse

Our method builds the graph warehouse by combining the inherent flexibility of graph structures with the concepts of multidimensional modeling. In this context, dimensions are modeled as nodes, and facts, which contain measurements, are represented as nodes, creating a clear and navigable structure for querying and analyzing data [23]. The edges in this graph model are crucial as they define the relationships between facts and dimensions, capturing both the structure of the data and the links between different entities.

Dimension D^S the node identification in this architecture is used to represent nodes (L^N, P^N) [24] where:

- L^N indicates the node's name.
- P^N indicates dimensions' attributes.

Fact: The fact node is represented as a node connected by edges to dimension nodes [25]. Furthermore, properties related to the measurements, such as applied aggregate functions or actual values, may be present in the fact node [26]. The fact node is determined by (M^F, N^F) where:

- N^F : the name of the fact.
- M^F : it consists of multiple measurements functioning as attributes for nodes, with each one linked to an aggregation function.

The link between a fact node and its associated dimensions: Edges in the model represent the relationship between a fact and its associated dimensions; this relationship is defined by (L^E, N^F, N^D, P^E) [27], where:

- L^E is the relationship's label.
- N^F is the fact node.
- N^D the node representing the associated dimension.
- P^E define the characteristics of the connection. The attributes include key-value pairs used to store relationship-related data [28].

3.2. Data historization in the graph warehouse

Upon the establishment of our graph warehouse, our next strategic step involves the implementation of the SGCD methodology. This approach aims to enhance our data warehousing capabilities by enabling the systematic tracking and management of changing dimensions within the graph structure [29]. By adopting SGCD, we plan to meticulously capture and preserve historical data changes, ensuring a comprehensive record of evolving information [30]. This implementation will enable us to efficiently handle variations in data

dimensions over time, ensuring accuracy and precision in analyzing historical trends and patterns [31]. With SGCD in place, our graph warehouse will possess a robust foundation for maintaining data integrity and facilitating in-depth historical analyses crucial for informed decision-making.

In our approach, we will implement SGCD by utilizing Algorithm 1: Managing historization dimensions in graph warehouses. This algorithm is designed to effectively manage changes in dimension data by creating new versions when necessary and updating current versions. Additionally, it maintains proper relationships within the graph-based data warehouse, ensuring data integrity and consistency.

Algorithm 1: Managing historization dimension in graph warehouse

```

1.  Require: New dimension data: NewData
2.  Ensure: Successful operation: Success
3.  Function ManageSGCDForDimension(NewData):
4.    ExistingRecord ← SearchDimensionRecordByKey (NewData.Key)
5.    If ExistingRecord is not empty :
6.      If ExistingRecord.Value ≠ NewData.Value:
7.        UpdateCurrentVersion(ExistingRecord)
8.      Else DoNothing()
9.    EndIf
10.   Else CreateNewVersion(NewData)
11.   EndIf
12.   Return Success
13. End Function
14. Function SearchDimensionRecordByKey(Key):
15.   Return DimensionRecord
16. End Function
17. Function UpdateCurrentVersion(Record):
18.   Record.ValidTo ← CurrentTimestamp()
19.   NewVersion ← CreateNewVersion(Record)
20.   Record.CurrentVersion ← NewVersion
21. End Function
22. Function CreateNewVersion(Data):
23.   NewNode ← CreateNewNode(Data)
24.   NewNode.ValidFrom ← CurrentTimestamp()
25.   Return NewNode
26. End Function
27. Function VersioningRelationship (DimensionRecord, Version):
28.   CreateRelationship (DimensionRecord, Version)
29. End Function
30. Function DoNothing ():
31.   // This function does nothing, used when the record has not changed
32. End Function

```

3.3. Generate frequently used itemsets

The next stage in this process entails retrieving user queries from the OLAP system logs subsequent to the construction of our graph warehouse. This retrieval is essential as it allows us to analyze user behavior and tailor the system to better meet their needs. Next, we use the Apriori technique to find common itemsets of predicates [32].

The Algorithm 2: Generate frequently used itemset has a significant impact in our graph cube model for determining frequent itemsets within OLAP queries. Its primary objective is to generate these frequent itemsets from transactional data by evaluating item occurrence frequency and filtering those surpassing a minimum support threshold (min-sup). The process comprises two main steps:

- Initial generation of unique itemset: The algorithm traverses transactions to establish the occurrence frequency of each item. It then forms a set of frequent itemsets based on the minimal support threshold.
- Extension of itemsets: Starting from frequent itemsets of size 1, the algorithm progressively generates larger itemsets. It creates potential candidates by combining previously identified frequent itemsets. These new sets are assessed within transactions to determine their frequency and are retained only if they surpass the minimal support threshold.

Algorithm 2: Generate frequently used itemset

```

1. frequentItemsets  $\leftarrow \{\}$ 
2. previousFrequentItemsets  $\leftarrow$  GenerateInitialFrequentItemsets
3. while previousFrequentItemsets is not empty do
4.   CurrentFrequentItemsets  $\leftarrow$  generateCandidates(previousFrequentItemsets)
5.   for each partition in graph cube partitions do
6.     for each candidate in currentFrequentItemsets do
7.       if candidate is employed in partition then
8.         candidate.frequency  $\leftarrow$  candidate.frequency + 1
9.       end if
10.    end for
11.  end for
12. previousFrequentItemsets  $\leftarrow$  currentFrequentItemsets
13. currentFrequentItemsets  $\leftarrow$  filterCandidates (currentFrequentItemsets, minSup)
14. frequentItemsets  $\leftarrow$  frequentItemsets  $\cup$  currentFrequentItemsets
15. end while

16. function GenerateInitialFrequentItemsets ()
17.   initialFrequentItemsets  $\leftarrow \{\}$ 
18.   for each item in all items do
19.     if item.frequency  $\geq$  minSup then
20.       initialFrequentItemsets  $\leftarrow$  initialFrequentItemsets  $\cup \{\text{item}\}$ 
21.     end if
22.   end for
23.   return initialFrequentItemsets
24. end function

25. function GenerateCandidates (itemsets)
26.   candidates  $\leftarrow \{\}$ 
27.   i  $\leftarrow$  0
28.   while i < size (itemsets) do
29.     itemset  $\leftarrow$  itemsets[i]
30.     j  $\leftarrow$  0
31.     while j < size(itemset) do
32.       element  $\leftarrow$  itemset[j]
33.       candidate  $\leftarrow$  itemset without element
34.       if candidate is not already included among the candidates then
35.         Add candidate to the set of candidates
36.       end if
37.       j  $\leftarrow$  j + 1
38.     end while
39.     i  $\leftarrow$  i + 1
40.   end while
41.   return candidates
42. end function

43. function FilterCandidates (candidates, minSup)
44.   frequentCandidates  $\leftarrow \{\}$ 
45.   i  $\leftarrow$  0
46.   while i < size (candidates) do
47.     candidate  $\leftarrow$  candidates[i]
48.     if candidate.frequency  $\geq$  minSup then
49.       Add candidate to frequentCandidates
50.     end if
51.     i  $\leftarrow$  i + 1
52.   end while
53.   return frequentCandidates
54. end function

```

3.4. Generate partitions using association rules

The next algorithm in our approach utilizes the attributes and relationships identified within the frequent itemsets to establish association rules. These rules serve as the basis for partitioning a graph based on these attributes and relationships. This partitioning allows for more efficient data retrieval and analysis, ultimately enhancing the performance of our graph-based system.

The Algorithm 3: Generate partitions using association rules begins by iterating through each frequent itemset. For each itemset, it generates all possible subsets. Subsequently, for each subset, it constructs an association rule, where the antecedent comprises the subset, and the consequent consists of the complement of the subset within the frequent itemset. This systematic process allows for the creation of association rules that capture the inherent patterns and correlations within the data.

Algorithm 3: Generate partitions using association rules

```

1. // Initialization
2. minSup: the minimum support threshold
3. PredicatesItemsets: list of frequent predicate itemsets
4. associationRules: list of rules
5. // Iterate through each frequent itemset
6.   for all frequentItemset In PredicatesItemsets do
7.     subsets ← GENERATESUBSETS(frequentItemset)
8.     rules ← []
9.   // Create rules based on confidence
10.    for all subset In subsets do
11.      antecedent ← subset
12.      consequent ← frequentItemset - subset
13.      confidence ← CALCULATECONFIDENCE(antecedent, frequentItemset)
14.      if confidence ≥ minSup then
15.        rule ← { antecedent: antecedent, consequent: consequent      confidence:
confidence }
16.        rules.add(rule)
17.      end if
18.    end for
19.    // Generate association rules for this frequent itemset
20.    associationRules ← associationRules ∪ rules
21.  end for

22. // Partition graph based on association rules
23. function PartitionGraphBasedOnRules(graph, associationRules)
24.  partitions ← {}
25.  for all associationRule IN associationRules do
26.    antecedentNodes ← FINDNODESWITHPROPERTIES(graph,
associationRule.antecedent)
27.    consequentNodes ← FINDNODESWITHPROPERTIES(graph,
associationRule.consequent)
28.    partition ← CREATEPARTITION(antecedentNodes, consequentNodes)
29.    partitions.add(partition)
30.  end for
31.  return partitions
32. end function

```

4. METHODOLOGY IMPLEMENTATION

4.1. Implementation of the graph warehouse

Neo4j (version 5.1.0) is the database that we used to create our graph warehouse. Neo4j is a graph database management system that efficiently stores and querying complex, interconnected data by representing and storing data in graph structures containing nodes, relationships, and characteristics [33]. Neo4j offers a configurable data format, enables ACID-compliant transactions and utilizes a query language known as Cypher specifically designed for handling graph data [34]. Additionally, we employed a comma-separated values (CSV) file including data from a flat meta-model and the transaction processing performance council - H (TPC-H) database as our source file [35]. The following are the model's dimensions:

- Product dimension with the Types of Products level.

- Customer dimension with the two levels RegionCustomer and cityCustomer.
- Dimension of the Supplier using the two levels citySupplier and RegionSupplier.
- Dimension time with the levels Year and Month.

Listing 1's script imports and manages supplier data from a CSV file into a Neo4j graph database. It effectively handles SGCD data by maintaining historical versions and ensuring data integrity.

- Data loading: The LOAD CSV WITH HEADERS clause loads the CSV data from the specified file into a temporary variable named row.
- Node merging: The MERGE clause creates or updates nodes in the graph database.
- Node creation: If a node with the specified SUPPLIER-ID does not exist, the MERGE clause creates a new SUPPLIER node with the provided properties.

Listing 1. The Supplier Dimension

```
LOAD CSV WITH HEADERS FROM "file:///scd.csv" AS row FIELDTERMINATOR ",";
MERGE (S: SUPPLIER {SUPPLIER_ID: row.SUPPLIER_ID})
ON CREATE SET
  S.SUPPLIER_NAME = row.SUPPLIER_NAME,
  S.SUPPLIER_CITY = row.SUPPLIER_CITY,
  S.SUPPLIER_REGION = row.SUPPLIER_REGION,
  S.SUPPLIER_CODE = row.SUPPLIER_CODE,
  S.START_DATE = row.START_DATE,
  S.END_DATE = (CASE WHEN row.IS_CURRENT = '1' THEN date ('9999-12-31')
ELSE date (row.START_DATE) END),
  S.IS_CURRENT = row.IS_CURRENT,
  S.VERSION = 1;
```

We use a similar methodology to generate the dimensions "CUSTOMER" "Time" and "PRODUCT" just as we did for the "SUPPLIER" dimension. The fact node, which stores the measures, must be created after all the dimension nodes. Listing 2 has the script that shows how to create a fact node in Neo4j and how to link a fact node to a dimension node. The script creates a node named "FACT" with the attributes "ID," "QUANTITY," and "Price" that are obtained from the appropriate columns in the CSV file using the MERGE clause. The "SUPPLIER" node with the matching "SUPPLIER-ID" and the "FACT" node with the matching "ID" property are located for each row by the script using the MATCH clauses. The "FACT-SUPPLIER" relationship is established by the MERGE clause between the matched "FACT" and "SUPPLIER" nodes.

Listing 2. The Fact Node

```
LOAD CSV WITH HEADERS FROM "file:///scd.csv" AS row FIELDTERMINATOR ",";
MERGE (meas:Measure {mid: row.INTEGRATION_ID})
ON CREATE SET
  meas.Price = toFloat(row.O_TOTALPRICE),
  meas.QUANTITY = toInteger(row.L_QUANTITY);
// Relationship FACT/SUPPLIER
LOAD CSV WITH HEADERS FROM "file:///scd.csv" AS row
FIELDTERMINATOR ",";
WITH row
MATCH (S: SUPPLIER {SUPPLIER_ID: row.SUPPLIERID})
MATCH (meas:Measure {mid: row.INTEGRATION_ID})
MERGE (meas)-[:FACT_SUPPLIER { START_DATE:row.START_DATE,
  END_DATE: (CASE WHEN row.IS_CURRENT = '1' THEN date ('9999-12-31') ELSE date
(row.START_DATE) END),
  ACTIVE: (CASE WHEN row.END_DATE = date ('9999-12-31') THEN 1 ELSE row.IS_CURRENT
END)}] ->(S);
```

We establish connections between the fact and the dimensions (TIME, CUSTOMER, and PRODUCT) with a similar method. To effectively import data from a CSV file and construct the associations between the "FACT" nodes and its associated nodes in the "PRODUCT," "TIME," and "COSTUMER" dimensions in the Neo4j database. Figure 1 stands for the realization of graph warehouse using Neo4j.

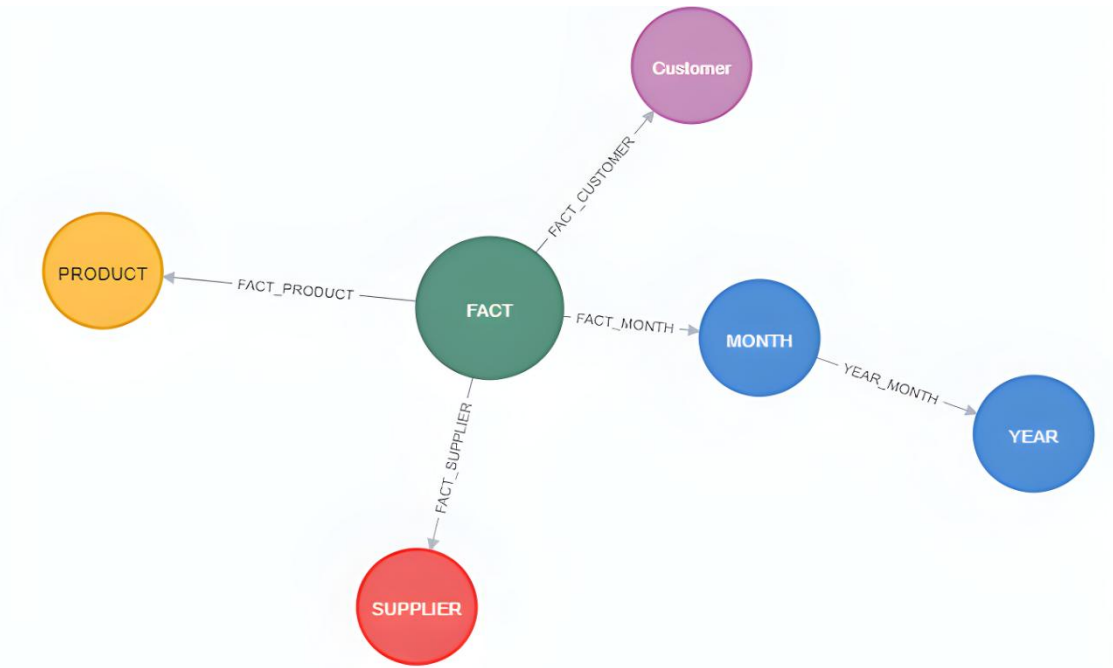


Figure 1. Our model presented in the sample study

4.2. Implementing historical dimension management with cypher queries

To implement the SGCD concept in Neo4j, we employ a cypher query-based approach to effectively manage changing dimensions. One common scenario in data management is the need to track changes in entity attributes over time. In the context of a graph database, these entities can be represented by nodes with changing properties, such as customers or products. For illustration, consider the case of a SUPPLIER dimension in a graph data warehouse. When a SUPPLIER changes their name or other information, it is essential to track these changes while maintaining a history of previous versions. The following cypher query demonstrates how this task can be accomplished:

Listing 3. Historical Dimension Management

```

LOAD CSV WITH HEADERS FROM "file:///scd.csv" AS row FIELDTERMINATOR ";"
WITH row
MATCH (S: SUPPLIER {SUPPLIER_ID: row.SUPPLIER_ID})
WHERE S.SUPPLIER_NAME <> row.SUPPLIER_NAME
and S.SUPPLIER_CITY <> row.SUPPLIER_CITY
and S.SUPPLIER_REGION <> row.SUPPLIER_REGION
SET S.IS_CURRENT = 0, S.END_DATE = datetime()
CREATE (newS: SUPPLIER {
  SUPPLIER_ID: row.SUPPLIER_ID,
  SUPPLIER_NAME: row.SUPPLIER_NAME,
  SUPPLIER_CODE: row.SUPPLIER_CODE,
  START_DATE: datetime(),
  END_DATE: date('9999-12-31'),
  IS_CURRENT: 1
})
CREATE (S)-[:VERSION_1]->(newS);
  
```

This cypher query manages changing dimensions in our data graph. It handles supplier updates by marking the current version as obsolete, creating a new version with the updated data, and establishing a relationship between the old and new versions to enable historical tracking. This approach ensures that changes in dimensions are appropriately accounted for, which is crucial for maintaining data consistency and enabling historical analysis in our graph data warehouse.

4.3. Partitioning graph warehouse

Once the frequently used queries were generated, the algorithm based on Apriori was employed to identify the most used itemsets. Subsequently, the second algorithm was executed to generate prevalent combinations for partition creation. Our parameter settings included a support threshold of 0.3 and a confidence level of 0.4. These combinations of the most commonly utilized partitions are illustrated in Figure 2.

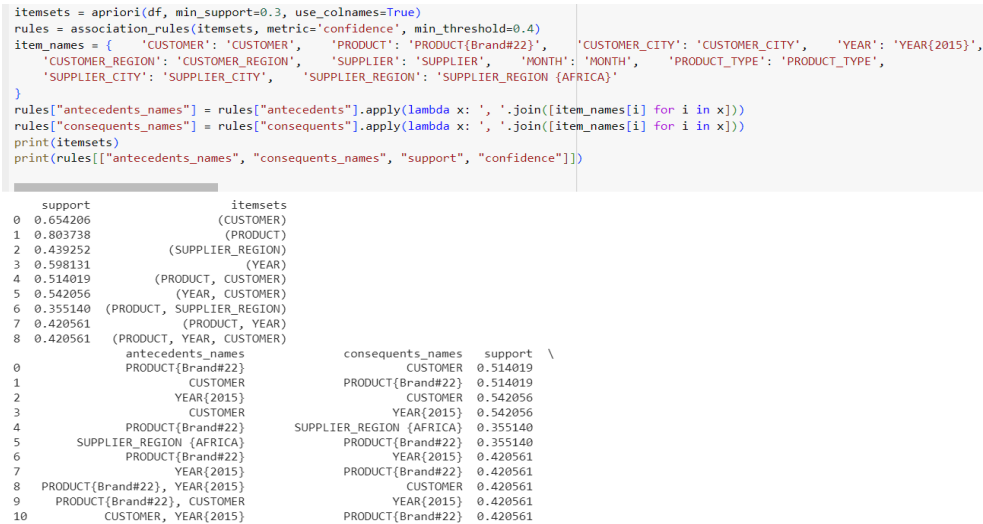


Figure 2. The combinations frequently utilized for partitioning

The partitions are sorted in decreasing order of support. Support is a measure of the frequency of occurrence of a partition in the dataset. The most commonly used partitions are as follows:

- P1: (PRODUCT {Brand#22}, CUSTOMER).
- P2: (YEAR {2015}, CUSTOMER).
- P3: (PRODUCT {Brand#22}, SUPPLIER-REGION {AFRICA}).
- P4: (PRODUCT {Brand#22}, YEAR {2015}).
- P5: (PRODUCT {Brand#22}, YEAR {2015}, CUSTOMER).

The number of partitions defined depends on the minimum support and minimum confidence specified by the user. A higher minimum support means that partitions must be more frequent in the dataset to be included in the results. By adjusting the values of the minimum support and minimum confidence, the user can control the number of partitions defined and the quality of the results.

5. DISCUSSION

We performed a sequence of experiments to validate our strategy and assess the impact of partitioning OLAP cube within the graph. These experiments involved evaluating performance levels before and after implementing our partitioned method. We measured query execution times for both the original cube and the partitioned cube. We performed our testing on a machine equipped with an i7 processor, 16 GB random access memory (RAM), and 1 TB storage capacity. Alongside, we utilized the TPC-H database at a scale factor of SF1, SF5 and SF10 which correspond to sizes of 1 GB, 5 GB, and 10 GB. In Table 1, we use the queries from the 5 derived partitions to subsequently compare the execution time.

The Figure 3 illustrates the query execution time comparison, displaying the times before partitioning the graph warehouse and after implementing the partitioning approach using a scale of 1 GB. The findings reveal a decrease in query execution times following the optimization and utilization of partitions. As an illustration, in the initial query, the execution time was around 3 ms milliseconds in the initial cube OLAP, and the same query it dropped to 2 ms when using partition cube. This signifies a notable percentage enhancement of about 33%. Similarly, in the query 5, significant improvements were realized as the execution time decreased from 14 to 12 ms. These substantial performance gains clearly highlight the efficacy of our approach, rendering the system nearly 16 times faster than its previous state. The Figure 4 illustrates the comparison of query execution times for the same query, displaying the times before partitioning the graph warehouse and after implementing the partitioning approach using scales of 5 GB and 10 GB respectively.

Table 1. Query used in our case study

Query	Dimension	Measure
Q1	– PRODUCT {Brand#22}	Sum(Price)
	– CUSTOMER	
Q2	– PRODUCT {Brand#22}	Sum(Price)
	– SUPPLIER	
	– REGION {AFRICA}	
Q3	– PRODUCT {Brand#22}	Sum(Price)
	– SUPPLIER	
	– REGION {AFRICA}	
Q4	– PRODUCT {Brand#22}	Sum(Price)
	– YEAR {2015}	
Q5	– PRODUCT {Brand#22}	Sum(Price)
	– YEAR {2015}	
	– CUSTOMER	

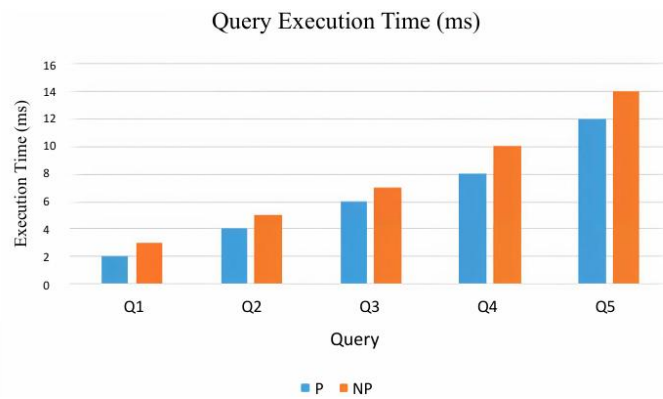


Figure 3. Query execution time for 1 GB

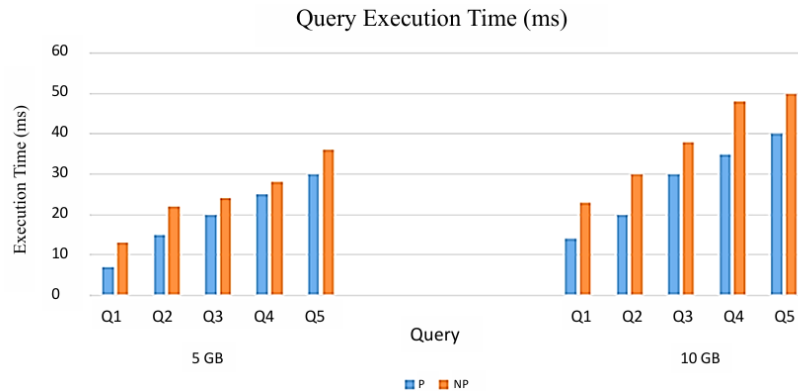


Figure 4. Query execution time for 5 GB and 10 GB dataset

When using a scale of 5 GB for the same query 1, there is a notable percentage enhancement of about 46%. Similarly, when using a scale of 10 GB, the execution time for the initial query was around 30 ms in the original OLAP cube. However, with the same query, it dropped to 14 ms when using the partitioned cube. This signifies a notable percentage enhancement of about 53%. The considerable decrease in execution time demonstrates the ability of this approach to substantially improve the system's speed compared to its previous state. This enhancement contributes to more efficient data analysis within the graph warehouse. Furthermore, the adoption of partitioning in the graph proves to be a crucial factor in achieving these significant improvements. By strategically organizing and managing data through partitions, the system not only experiences a substantial boost in speed but also lays the groundwork for streamlined and optimized data processing.

6. CONCLUSION

In our paper, we present our contribution to constructing a data warehouse within a graph database. We discuss the significance of data historization and introduce a novel method for partitioning our graph warehouse, utilizing the association rules algorithm. In a series of studies, we compared the performance before and after partition implementation in order to confirm our methodology and assess the benefits of using partitions in the graph warehouse. The experiment's outcomes show the advantages of partitioning graph warehouse systems, particularly when handling big data sets. In our upcoming research projects, we will focus on developing novel techniques for partitioning data warehouses in various NoSQL databases, like document and column databases.




REFERENCES

- [1] Z. A. El Mouden and A. Jakimi, "A new algorithm for storing and migrating data modelled by graphs," *International Journal of Online and Biomedical Engineering (iJOE)*, vol. 16, no. 11, pp. 137–152, 2020, doi: 10.3991/ijoe.v16i11.15545.
- [2] M. El Malki, A. Kopliku, E. Sabir, and O. Teste, "Benchmarking big data olap nosql databases," in *4th International Symposium on Ubiquitous Networking (UNet 2018)*, Hammamet, Tunisia, 2018, pp. 82–94, doi: 10.1007/978-3-030-02849-7_8.
- [3] A. Ghrab, O. Romero, S. Skhiri, and E. Zimányi, "TopoGraph: an end-to-end framework to build and analyze graph cubes," *Information Systems Frontiers*, vol. 23, no. 1, pp. 203–226, 2021, doi: 10.1007/s10796-020-10000-z.
- [4] M. Kamm, M. Rigger, C. Zhang, and Z. Su, "Testing graph database engines via query partitioning," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, USA: ACM, 2023, pp. 140–149, doi: 10.1145/3597926.3598044.
- [5] A. Ghrab, O. Romero, S. Skhiri, A. Vaisman, and E. Zimányi, "A framework for building olap cubes on graphs," in *Advances in Databases and Information Systems*, Poitiers, France, 2015, pp. 92–105, doi: 10.1007/978-3-319-23135-8_7.
- [6] S. Ahmadi, "Optimizing data warehousing performance through machine learning algorithms in the cloud," *International Journal of Science and Research (IJSR)*, vol. 12, no. 12, pp. 1859–1867, 2023, doi: 10.21275/sr231224074241.
- [7] A. Castelltort and A. Laurent, "Representing history in graph-oriented nosql databases: a versioning system," in *8th International Conference on Digital Information Management, ICDIM 2013*, 2013, pp. 228–234, doi: 10.1109/ICDIM.2013.6694022.
- [8] A. Castelltort and A. Laurent, "Fuzzy historical graph pattern matching a nosql graph database approach for fraud ring resolution," *IFIP Advances in Information and Communication Technology*, vol. 458, pp. 151–167, 2015, doi: 10.1007/978-3-319-23868-5_11.
- [9] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov, "Streaming graph partitioning: an experimental study," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1590–1603, 2018, doi: 10.14778/3236187.3236208.
- [10] M. H. Mofrad, R. Melhem, and M. Hammoud, "Partitioning graphs for the cloud using reinforcement learning," *arXiv-Computer Science*, pp. 1–9, 2019.
- [11] T. A. Ayall *et al.*, "Graph computing systems and partitioning techniques: a survey," *IEEE Access*, vol. 10, pp. 118523–118550, 2022, doi: 10.1109/ACCESS.2022.3219422.
- [12] P. Zhao, X. Li, D. Xin, and J. Han, "Graph cube: on warehousing and olap multidimensional networks," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2011, pp. 853–864, doi: 10.1145/1989323.1989413.
- [13] A. Castelltort and A. Laurent, "Fuzzy queries over nosql graph databases: perspectives for extending the cypher language," *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pp. 384–395, 2014, doi: 10.1007/978-3-319-08852-5_40.
- [14] A. S. Muttipati and P. Padmaja, "Analysis of large graph partitioning and frequent subgraph mining on graph data," *International Journal of Advanced Research in Computer Science*, vol. 6, no. 7, pp. 29–40, 2015.
- [15] D. Dai, W. Zhang, and Y. Chen, "IOGP: an incremental online graph partitioning algorithm for distributed graph databases," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, New York, USA: ACM, 2017, pp. 219–230, doi: 10.1145/3078597.3078606.
- [16] H. Akid, G. Frey, M. B. Ayed, and N. Lachiche, "Performance of nosql graph implementations of star vs. snowflake schemas," *IEEE Access*, vol. 10, pp. 48603–48614, 2022, doi: 10.1109/ACCESS.2022.3171256.
- [17] L. Andriamampianina, F. Ravat, J. Song, and N. Vallès-Parlangeau, "Graph data temporal evolutions: from conceptual modelling to implementation," *Data and Knowledge Engineering*, vol. 139, 2022, doi: 10.1016/j.datak.2022.102017.
- [18] X. Zhou, G. Li, J. Feng, L. Liu, and W. Guo, "Grep: a graph learning based database partitioning system," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–24, 2023, doi: 10.1145/3588948.
- [19] R. Benhissen, F. Bentayeb, and O. Boussaid, "GAMM: graph-based agile multidimensional model," *CEUR Workshop Proceedings*, vol. 3369, pp. 23–32, 2023.
- [20] A. Y. H. Chou and F. S. C. Tseng, "A theoretical framework for temporal graph warehousing with applications," *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 6, pp. 260–270, 2024, doi: 10.14569/IJACSA.2024.0150628.
- [21] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 487–499.
- [22] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 207–216, 1993, doi: 10.1145/170036.170072.
- [23] D. Martinez-Mosquera, R. Navarrete, S. Luján-Mora, L. Recalde, and A. Andrade-Cabrera, "Integrating olap with NoSQL databases in big data environments: systematic mapping," *Big Data and Cognitive Computing*, vol. 8, no. 6, 2024, doi: 10.3390/bdcc8060064.
- [24] R. Labzioui, K. Letrache, and M. Ramdani, "New approach based on association rules for building and optimizing olap cubes on graphs," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 7, pp. 997–1008, 2023, doi: 10.14569/IJACSA.2023.01407108.
- [25] A. Sellami, A. Nabli, and F. Gargouri, "Transformation of data warehouse schema to nosql graph data base," *Advances in Intelligent Systems and Computing*, vol. 941, pp. 410–420, 2020, doi: 10.1007/978-3-030-16660-1_41.
- [26] R. Labzioui, K. Letrache, and M. Ramdani, "New strategy for developing and enhancing online analytical processing cubes on graphs," in *2023 14th International Conference on Intelligent Systems: Theories and Applications (SITA)*, IEEE, 2023, pp. 1–8, doi: 10.1109/SITA60746.2023.10373687.
- [27] A. Khalil and M. Belaissaoui, "A graph-oriented framework for online analytical processing," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 5, pp. 547–555, 2022, doi: 10.14569/IJACSA.2022.0130564.




- [28] A. Sellami, A. Nabli, and F. Gargouri, "Graph nosql data warehouse creation," in *Proceedings of the 22nd International Conference on Information Integration and Web-based Applications & Services*, New York, USA: ACM, 2020, pp. 34–38, doi: 10.1145/3428757.3429141.
- [29] M. Goller and S. Berger, "Handling measurement function changes with slowly changing measures," *Information Systems*, vol. 53, pp. 107–123, 2015, doi: 10.1016/j.is.2014.12.009.
- [30] T. Phungtua-Eng and S. Chittayasothorn, "Slowly changing dimension handling in data warehouses using temporal database features," in *Intelligent Information and Database Systems*, 2019, pp. 675–687, doi: 10.1007/978-3-030-14799-0_58.
- [31] M. Kromer, "Slowly changing dimensions," in *Mapping Data Flows in Azure Data Factory*, California, USA: Apress Berkeley, 2022, pp. 79–92, doi: 10.1007/978-1-4842-0082-7_13.
- [32] K. Letrache, O. El Beggar, and M. Ramdani, "OLAP cube partitioning based on association rules method," *Applied Intelligence*, vol. 49, no. 2, pp. 420–434, 2019, doi: 10.1007/s10489-018-1275-2.
- [33] M. Friedrichs, "BioDWH2: an automated graph-based data warehouse and mapping tool," *Journal of integrative bioinformatics*, vol. 18, no. 2, pp. 167–176, 2021, doi: 10.1515/jib-2020-0033.
- [34] A. Vaisman, F. Besteiro, and M. Valverde, "Modelling and querying star and snowflake warehouses using graph databases," *Communications in Computer and Information Science*, vol. 1064, pp. 144–152, 2019, doi: 10.1007/978-3-030-30278-8_18.
- [35] T. M. Allam, "Estimate the performance of cloudera decision support queries," *International Journal of Online and Biomedical Engineering*, vol. 18, no. 1, pp. 127–138, 2022, doi: 10.3991/ijoe.v18i01.27877.

BIOGRAPHIES OF AUTHORS






Redouane Labzioui    is a Ph.D. student at Faculty of Sciences and Techniques of Mohammedia (FSTM), University Hassan II, Casablanca, Morocco. He is working on the issue of the business intelligence system based on NoSQL. He can be contacted at email: redouane.labzioui-etu@etu.univh2c.ma.



Dr. Khadija Letrache    received her Ph.D. degree in computer science from Faculty of Sciences and Techniques of Mohammedia (FSTM), University Hassan II, Casablanca, Morocco in 2019. She is currently a Professor of computer engineering at the Faculty of Sciences and Techniques of Mohammedia (FSTM), University Hassan II. She is an author of several papers in international journals and conferences. Her research interests include business intelligence and MDA architecture. She can be contacted at email: khadija.lettrache@fstm.ac.ma.



Dr. Mohammed Ramdani    received his Ph.D. in fuzzy machine learning in 1994, and his HDR in perceptual computation in 2001, at University Paris VI, France. Since 1996, he is a full Professor at the FSTM, University Hassan II of Casablanca, Morocco. In the same faculty, for the periods 1996–1998 and 2003–2005 he held the position of head of Department of Computer Science. Between 2008 and 2014, he was Pedagogical Director of the Department of Engineering "Software Engineering and Systems Integration" (ILIS). Since 2006, he is Director of the Computer Science Lab. His research interests include explanation in machine learning, perceptual computation with fuzzy logic, and big datamining. He is author of several articles in many indexed journals. He can be contacted at email: mohammed.ramdani@fstm.ac.ma.