New Approach based on Association Rules for Building and Optimizing OLAP Cubes on Graphs

Redouane LABZIOUI[1], Khadija LETRACHE[2], Mohammed RAMDANI[3]
Informatics Department, LIM Laboratory
Faculty of Sciences and Techniques of Mohammedia
University Hassan II, Casablanca, Morocco

***Abstract*—The expansion of data has prompted the creation of various NoSQL (Not only SQL) databases, including graph-oriented databases, which provide an understandable abstraction for modeling complex domains and managing highly connected data. However, to add graph data to existing decision support systems, new data warehouse systems that consider the special characteristics of graphs need to be developed. This work proposes a novel method for creating a data warehouse under a graph database and demonstrates how OLAP (Online Analytical Processing) structures created for reporting can be handled by graph databases. Additionally, the paper suggests using aggregation algorithms based association rules techniques to improve the efficiency of reporting and data analysis within a graph-based data warehouse. Finally, we provide a Cypher language implementation of the suggested approach to evaluate and validate our approach.**

***Keywords*—NoSQL; graph-oriented databases; data warehouse; OLAP; aggregation algorithms; association rules; cypher language**

I. Introduction

Modern databases have been considerably altered by the expansion of data. NoSQL databases have expanded as a result of these new demands and now come in a wide range of models [1], including key-value, document, column, and graph.

In particular, graph-oriented databases are one of the most-known various of NoSQL systems; they have attracted a lot of attention and popularity. Graph-oriented databases are a fundamental form that offer an understandable abstraction to model numerous complicated domains, manage highly connected data, and run sophisticated queries over them [2] [3].

Currently, many businesses and entrepreneurs are interested in developing business intelligence systems on graph databases to take advantage of its benefits. Enterprises are also interested in expanding their OLAP analysis to include the new forms of data because OLAP technology is currently widely used[4].However, despite the growing interest in graph-based data warehouses and their potential benefits, there is a lack of research addressing the challenge of creating an optimized OLAP model under graphs, specifically focusing on selecting the most relevant OLAP aggregations to effectively meet the diverse user's needs. This gap calls for further investigation and exploration to bridge the divide between the advantages of graph-based data warehousing and the need for efficient and user-centric aggregation selection techniques. The aim of this study is to create new data warehouse under graph database systems that consider the special characteristics of graphs, this work also aims to optimize the efficiency of reporting and data analysis within this graph-based data warehouse by employing user-centric aggregation techniques that select the most relevant OLAP aggregations to meet the diverse needs of users effectively. To address this issue, it is possible to use aggregation algorithms that automatically select the most relevant aggregations for the cube. These aggregation algorithms are often based on machine learning and data mining techniques to identify the best possible aggregations based on raw data. The use of these algorithms can help reduce cube build time and improve the accuracy of analysis results. In this context, we demonstrate how OLAP structures created for reporting can be handled by graph databases, additionally, We provide new approach to optimize OLAP cube using the association rules algorithm.

The remainder of this paper is organized as follows. In Section II, we present some works of the literature reviews related on graph data warehouse. In Section III, we give a background overview of our approach. In Section IV, we describe the implementation of our approach as well as a case study to assess it. Section V concludes this paper and suggest future research directions.

II. Related Work

Many approaches were proposed in the literature as a result of the growing interest in combining graph databases and business intelligence technology in recent years. In [5], The authors have proposed a new concept Graph Cube, a new data warehouse model that supports OLAP queries in large multidimensional networks, in the Graph Cube the dimensions are based on the attributes of the nodes, while the computed measures represent the aggregations of these node attributes. The Graph Cube approach have some limitations in analyzing dynamic or evolving graphs, where the structure of the graph changes over time. Moreover, the accuracy and reliability of the analysis results may be affected by data quality issues such as missing or erroneous data. In [6], The author introduce a GraphAware Framework for Neo4j, which enables the pre-calculation and storage of node information in graphs. For instance, the framework can compute the number of friends in a social network and store the result for efficient querying. The GraphAware Framework also supports the analysis of node degrees in the graph, which can be useful for identifying important or influential nodes. The GraphAware Framework is focused on precalculating and storing node information, which may be limited in scope and may not capture the

full complexity of the graph data. For example, some graph analytics tasks may require more sophisticated computations that go beyond simple node attributes, such as graph centrality measures or community detection. In [7], The authors proposed to use the graph structure as a basis for OLAP queries; this approach relies on using the performance and efficiency of the Neo4j graph database to store and query OLAP queries. In this model, dimensions and measures are transformed into nodes. The connection between dimensions and measures is done through arcs of the graph. For hierarchical dimensions are also stored in nodes and linked together by hierarchical relationships. The approach is limited to using only the snowflake model, which may not be suitable for all types of data. This may restrict the flexibility and adaptability of the approach. In [8], The author's goal is to compare the execution times of the identical OLAP queries in the relational and graph databases by using a MusicBrainz database data warehouse in a PostgreSQL relational environment and then implementing the same decision model in Neo4j. The authors only tested their approach on a single dataset (the MusicBrainz database), which may limit the generalizability of their results. Additionally, they only considered a specific decision model and did not explore the potential impact of different OLAP queries or decision models on the performance of the two database types. Furthermore, while the results of the study showed that the graph database outperformed the relational database in terms of execution time, the authors did not provide a detailed analysis of the factors that contributed to these results. This lack of analysis makes it difficult to fully understand the advantages and disadvantages of each database type for OLAP queries. In [9],The authors define a set of transformation rules that can transform conceptual models into graph-oriented models. They have defined four transformation rules, namely: fact transformation, Name and Identifier of Dimension Transformation, Hierarchies Transformation, Transformation of the relation between Fact and Dimension. Still within the framework of Datawarehouse modelling in a NoSQL graph database, the authors propose a conceptual mapping between a multidimensional schema and a graph-oriented NoSQL model. The authors chose to concentrate on proving the viability of their approach rather than providing any experimental campaign to validate it. In [10], the authors proposes to integrate NoSQL Graph-oriented Data into Data Warehouses as a solution to tackle Big Data challenges, The paper introduces a new approach called "Big-Parallel-ETL" that adapts the classical ETL (Extract-Transform-Load) process with Big Data technologies, leveraging the efficiency of the MapReduce concept for parallel processing. However, this work does not address OLAP aggregations. The authors in [11], suggest a set of guidelines to create a graph data model from a multidimensional data model (MDM2G). Then the authors compare the performance of the two star and snowflake designs in the graphs and relational databases, in terms of dimensionality and size. After doing this comparison, the authors concluded that a graph implementation of a data warehouse with multiple tables is more effective than a relational implementation, and that a star model performs similarly to a snowflake model in graph databases. The study does not provide a detailed description of the conversion rules and does not present any experimental results or validation of the proposed approach.In [12], the authors proposed employing two alternative logical models, equivalent to the ROLAP (Relational Online Analytical Processing) and MOLAP (Multidimensional Online Analytical Processing) models, to create OLAP engines within a graph database. They specify a set of guidelines for mapping these models from the multidimensional model. Additionally, they suggest an aggregation technique for constructing the lattice of cuboids from a data warehouse. However, the choice of aggregations is random and imprecise, which makes the model unoptimized and burdens the graph with several unnecessary nodes. The authors in [13], suggest an approach founded on a multi-version evolutionary schema model. Data instances corresponding to various schema versions are stored in a graph data warehouse. A meta-model is utilized to manage these warehouse schema versions. Additionally, they introduce evolution functions at the schema level. To validate their approach, they implement a software prototype and conduct a case study that demonstrates queries on schema versions, cross-queries, and the runtime performance of their approach. However, the impact of the multi-version approach on OLAP aggregations is not addressed in this study.

The Table I provides a comprehensive summary of the literature review, highlighting the key findings and identified gaps in the research.

All of the cited works provide an important context for the implementation of decision systems using graph databases. However, most of these works focus on converting relational data warehouses into graph databases or applying traditional business intelligence methods, which can limit the advantages of using graph databases. Our proposed approach is different and relies on the properties of graphs to implement data warehouses. It highlights the importance of studying a model that optimizes the choice of OLAP aggregations to enhance the graph cube's performance. By selecting the optimal set of aggregations for a graph cube, OLAP query performance can be significantly improved, resulting in faster query response times and more efficient use of system resources. This can enable users to analyze larger volumes of data more quickly and accurately, leading to more informed decision-making. Moreover, reducing the computational resources required to execute OLAP queries can result in cost savings for organizations that need to process large amounts of data.

III. Our Approach and Background Information

A. Background Information

Graph Oriented Database: Store data entities as nodes and entity relationships as edges. A periphery always has a start node, an end node, a type, and a direction. A node can describe relationships, actions, parent-child ownership, etc. The number and type of relationships a node can have are unlimited.

A property graph is defined as

TABLE I. LITERATURE REVIEW

Year	Authors	Findings	Gaps
2011	Zhao et al.	Graph Cube: A new paradigm for Data Warehouse (DW) that supports OLAP queries in large multidimensional networks, with dimensions based on node attributes and computed measures representing aggregations.	Limitations in analyzing dynamic or evolving graphs and potential data quality issues.
2013	Bachman	GraphAware Framework for Neo4j enables pre-calculation and storage of node information for efficient querying, but may not capture the full complexity of graph data.	Limited in handling more sophisticated computations beyond simple node attributes.
2014	Castelltort et al.	Proposes using graph structure as a basis for OLAP queries, but limited to the snowflake model, reducing flexibility.	May not be suitable for all types of data.
2019	Vaisman et al.	Comparing the execution timings of identical OLAP queries in relational and graph databases reveals that the graph database provides superior performance.	Lack of detailed analysis of factors contributing to the performance difference.
2020	Sellami et al.	Define transformation rules to convert conceptual models into graph-oriented models.	No experimental campaign to validate the approach.
2021	Soussi	Propose parallel loading based integration of NoSQL graph-oriented data into data warehouses.	Doesn't address OLAP aggregations.
2022	Akid et al.	creating graph data models based on multidimensional data and comparing star and snowflake designs in graphs and relational databases.	Lack of detailed conversion rules and experimental validation.
2022	Khalil et al.	Propose alternative logical models for OLAP engines within a graph database and an aggregation technique for constructing the lattice of cuboids from a data warehouse.	Unoptimized aggregations and unnecessary nodes in the graph.
2023	Benhissen et al.	Propose an approach based on a multi-version evolutionary schema model in a graph data warehouse.	Doesn't address the impact of multi-version approach on OLAP aggregations.

$G = (N, E, L^N, L^E, P^N, P^E)$,

where:

N is a set of finite nodes,

$E \subseteq N \times N$ represent edges between the nodes.

L^N describes the label of the nodes.

L^E describes the edges' label..

P^N is a set of characteristics that identify node.

P^E is a set of properties that describe an edge.

Conceptual Multi-Dimensional Schema: Before defining our model, we clarify the concepts of the conceptual model: (dimensions, hierarchies, and measures) [14].
The attributes of the multidimensional schema are :$(F^M, D^M, Star^M)$ where[12]:

- $F^M = F_1, \ldots, F_n$ is a set of facts.
- $D^M = D_1, \ldots, D_i$ is a set of finite dimensi
- $Star^M : F_i \to 2^{D}i$ maps each fact
corresponding dimensions D_i.

Measures are a group of properties that make
Each measure has an aggregate function attached t
Facts are determined by: $(N^F, M^F)$where:

- N^F represent the fact name.
- M^S is a collection of measures, every one of which has an aggregate function.

A dimension consists of a set of attributes representing different levels of granularity on the data to be analyzed (measures).

A dimension, denoted $D_i \in D^S$, This is characterized by (N^D, A^D, H^D) where:

- N^D is the name of the dimension.
- $A^D = A_1, \ldots, A_n$ is a set of dimension attributes.
- $H^D = H_1, \ldots, H_n$ is a set of hierarchies, arranging the properties in accordance with the level of granularity that each one represents.

The Fig. 1 illustrates our multidimensional use case model:

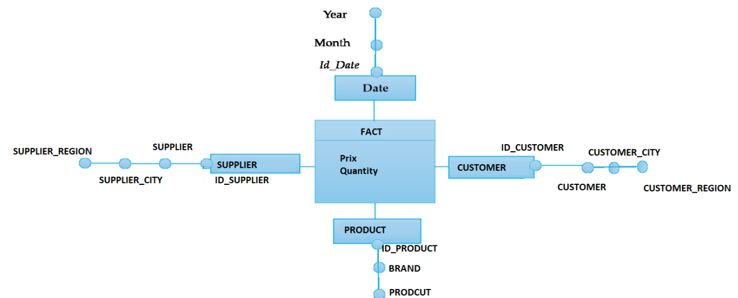

Fig. 1. The multi dimensional model.

[H]

B. Our Approach

Our approach involves leveraging the advantages of graph databases by creating the OLAP cube in the graph and using user queries to identify frequently used dimensions in OLAP analyses. To optimize the aggregations to be created, we use the Apriori algorithm to extract the most frequently associated sets of dimensions in OLAP queries, and then apply a rule-based association algorithm to identify the most relevant aggregations. The resulting aggregations are created in the OLAP cube, leading to improved OLAP query performance. The approach consists of four steps:

- Creation of the graph data warehouse.
- Extraction of the most frequently associated sets of dimensions in OLAP queries using the Apriori algorithm.
- Identification of the most relevant aggregations using a rule-based association algorithm.
- Creation of identified aggregations.

1) Graph Data Warehouse: When used with relational databases, a relational fact table is created for each fact in the multi dimensional conceptual model. Measures are columns in the fact table. Additionally, each dimension is transformed into a normalized dimension table with columns for each attribute (including parameters and weak attributes). Fact and dimension tables each have a unique row for storing each instance [16]. Similarly, we provide our rules, which define a graph DW, using the definitions of multi dimensional model and property graph ideas that were previously presented.

Dimension D^S in our model is created in node format identified by (L^N, P^N) where:

- $\mathrm{L^N}$ represents the label of the node, a node can have zero to many labels.
- $\mathrm{P^N}$ represents attributes of the dimension.

Hierarchies: In a graph data warehouse, a hierarchy can be represented using nodes and edges. Each level of the hierarchy can be represented as a node, with edges connecting nodes at different levels to indicate parent-child relationships.

Fact: A fact node in a graph data warehouse can be represented as a node with edges connecting it to dimension nodes. The fact node can also have properties that represent the measures, such as the actual values or aggregate functions applied to them [17].A fact node, is specified by (N^F, M^F) where:

- $\mathrm{N^F}$: represent the fact name.
- $\mathrm{M^F}$: It comprises a collection of measures as node attributes, with each measure linked to an aggregation function.

Relationship between fact and dimensions: The relationship between fact and its associated dimensions are represented as edges connecting nodes in the graph, the link is defined by (L^E, N^F, N^D, P^E), where:

- $\mathrm{L^E}$ is the label of relationship.
- $\mathrm{N^F}$ is the fact node.
- $\mathrm{N^D}$ is a node that represents the dimension linked to the fact.
- $\mathrm{P^E}$ represent the properties of the relationship, the properties are key-value pairs that are used for storing data on relationships.

2) Algorithm 1: Calculate Frequent Itemsets.: After creating our OLAP system, the second phase in our approach is to collect user queries from the OLAP system logs [18]. And after that, we determine common itemsets of predicates using the Apriori algorithm [19]. In the next phase we will use the generated itemsets as input to the second association rule algorithm to determine the most important aggregations to create. The algorithm starts by initializing an empty list to store frequent itemsets and another empty list to store previous frequent itemsets. Then, it loops on user queries predicates until there are no more frequent itemsets to explore. During each iteration of the loop, the algorithm generates candidates for new frequent itemsets by combining previous frequent itemsets. Then, it calculates the frequency of each candidate by scanning through all transactions. Candidates that have a frequency below a minimum support threshold are filtered out. Finally, the frequent itemsets are added to the list of frequent itemsets. The candidate generation and filtering functions are also defined in the algorithm.

Algorithm 1 Calculate frequent itemsets

```
// Initialization
frequent_itemsets ←
previous_frequent_itemsets ←
// Loop until there are no more frequent itemsets to explore
while (previous_frequent_itemsets is not empty) do
    // Generate candidates
    current_frequent_itemsets ← generate_candidates(previous_frequent_itemsets)
    // Calculate frequency of candidates
    for each aggregation in OLAP cube do
        for each candidate in current_frequent_itemsets do
            if candidate is used in aggregation then
                candidate.frequency ← candidate.frequency + 1
            end if
        end for
    end for
    // Filter candidates
    previous_frequent_itemsets ← current_frequent_itemsets
    current_frequent_itemsets ← filter_candidates(current_frequent_itemsets, min_sup)
    // Add frequent itemsets to the list
    frequent_itemsets ← frequent_itemsets ∪ current_frequent_itemsets
end while

FUNCTION generate_ candidates(itemsets)
candidates
for each itemset1 in itemsets do
    for each itemset2 in itemsets do
        if itemset1 ≠ itemset2 and all elements of itemset1 except the last one are the same as the corresponding elements of itemset2 then
            candidate ← union of itemset1 and itemset2, keeping only the unique elements
            if candidate is not already in candidates then
                candidates ← candidates ∪ candidate
            end if
        end if
    end for
end for
return candidates
End FUNCTION

FUNCTION filter_ candidates (candidates, min_ sup)
frequent_candidates ←
for each candidate in candidates do
    if candidate.frequency ≥ min_sup then
        frequent_candidates ← frequent_candidates ∪ candidate
    end if
end for
return frequent_candidates
End FUNCTION
```

3) *Algorithm2: Generate Association Rules.:* The second algorithm in our approach is used to generate association rules from frequent itemsets. It first loops through each frequent itemset, and for each itemset it generates all possible subsets. For each subset, it creates an association rule with the antecedent being the subset and the consequent being the complement of the subset in the frequent itemset.

Algorithm 2 Generate Association Rules

```
association_rules ←
for each frequent_itemset in frequent_itemsets do
    subsets ← generate_subsets(frequent_itemset)
    for each subset in subsets do
        antecedent ← subset
        consequent ← (frequent_itemset) − (subset)
        association_rule ← {antecedent : antecedent, consequent : consequent, confidence : 0}
        // Calculate confidence of the rule
        frequency_antecedent ← calculate_frequency(antecedent)
        frequency_frequent_itemset ← calculate_frequency(frequent_itemset)
        confidence_frequency_frequent_ ← itemset_frequency_antecedent
        if confidence ≥ min_conf then
            association_rule.confidence ← confidence
            association_rules ← association_rules ∪ association_rule
        end if
    end for
end for

FUNCTION generate_ subsets(itemset)
subsets ←
for i ← 1 to taille(itemset) do
    subset ←
    for j ← 1 to taille(itemset) do
        if j ≠ i then
            subset ← subset ∪ itemset[j]
        end if
    end for
    subsets ← subsets ∪ subset
end for
return subsets
End FUNCTION

FUNCTION calculate_ frequency (itemset)
frequency ← 0
for transaction in transactions do
    if transaction contains all elements of itemset then
        frequency ← frequency + 1
    end if
end for
return frequency
End FUNCTION
```

4) *Graph Aggregation:* The two optimization algorithms previously defined enable finding the best combinations of aggregations, For improving query performance and maximising data retrieval in a graph, OLAP aggregations must be created. Aggregations act as quick-query summaries of data that have

already been calculated, saving time and resources needed to obtain the raw data. Complex queries can be conducted in a fraction of the time it would take to scan the full data set by specifying and producing the relevant aggregations. In our approach, aggregations are stored both in the nodes and in the edges. since the graph allows to put the measures in properties of the links, the advantage of this model will allow us to minimize the number of nodes created in the graph, and also allows us to take advantage of the benefits of graphs. We will also put in the links between the dimensions the information of time dimension.
We will store aggregations in relationships when the aggregation combines only one dimension, two dimensions, or three dimensions (including time dimensions). Aggregations that combine more than three dimensions will be stored in nodes. The link-Aggregation is represented by an edge (L^E, N^S, N^T, P^E), where:

- L^E is the label of relationship.
- N^S is the start node.
- N^T is the target node.
- P^E represent the properties of the relationship, the properties are key-value pairs that are used for storing aggregation on relationships.

IV. Approach Implementation and Evaluation

A. Implementation of the Graph Warehouse

We implemented our graph warehouse using the Neo4j database (version 5.1.0)[1]. Neo4j is a graph database management system that uses graph structures with nodes, relationships, and properties to represent and store data[20], enabling efficient storage and querying of complex, interconnected data. Neo4j supports ACID-compliant transactions, offers a flexible data model, and provides a query language called Cypher for working with graph data[21].

In addition,We used the TPC-H benchmark database as a source file, We made use of a global flat CSV file that contains information from a flat meta-model. The TPC-H benchmark database has been used to provide support for Big Data technologies, including NoSQL and Hadoop file systems[22]. It generates data in various file formats (xml, jason, csv, tab, ...) following different data models.

The dimensions used in the model are as follows:

- Product dimension with the TypeProduct hierarchy.
- Customer dimension with the cityCustomer and RegionCustomer hierarchy.
- Supplier dimension with the citySupplier and RegionSupplier hierarchy.
- Year dimension with the Month hierarchy.

The script in Listing 1 is used to import data from a CSV file ("File.csv") into Neo4j database by creating nodes to represent different entities related to the (CUSTOMER) as well as the hierarchies associated with these entities (CityCustomer and RegionCustomer). The script uses the APOC (Awesome Procedures on Cypher) library's apoc.periodic.iterate procedure to efficiently manage the data import from the CSV file. It performs the import by iterating over batches of data, allowing it to process large amounts of data while avoiding overwhelming the system. The function 'LOAD csv WITH HEADERS FROM "file:///File.csv" as row FIELDTERMINATOR ";" RETURN row' loads the CSV data with headers into the "row" variable and uses the semicolon (;) as the field delimiter. The first function MERGE (creates or updates) a node with the "CUSTOMER" label having properties "CUSTOMER-ID" and "CUSTOMER-NAME" extracted from the corresponding columns in the CSV file. The options batchSize:10000, allow the data to be processed in batches of 10,000 rows in parallel for better performance. The second use of the MERGE function in the script creates or merges nodes with the "CityCust" label having properties "CUSTOMER-CITYID" and "CUSTOMER-CITY" extracted from the CSV file. Similarly, the third use of the MERGE function is also used for creating or matching nodes in the graph database for the "RegionCust" hierarchy.

Listing 1: The Customer Dimension

```

// Dimension Customer
  CALL apoc.periodic.iterate(
 'LOAD csv WITH HEADERS FROM ``file:///File.
    csv" as row FIELDTERMINATOR ``;"
  RETURN row',
 'MERGE (C:CUSTOMER [CUSTOMER_ID : row.
    CUSTOMER_ID,CUSTOMER_NAME: row.
    CUSTOMER_NAME])',
 [batchSize:10000, parallel:true]);
// Hierarchy CityCustomer
 CALL apoc.periodic.iterate (
 'LOAD CSV WITH HEADERS FROM ``file:///File.
    csv" as row FIELDTERMINATOR ``;"
  RETURN row',
 'MERGE (CC:CityCust [CUTOMER_CITYID : row.
    CUSTOMER_CITYID,CUSTOMER_CITY: row.
    CUSTOMER_CITY])',
 [batchSize:10000, parallel:true]);
// Hierarchy RegionCustomer
 CALL apoc.periodic.iterate(
 'LOAD CSV WITH HEADERS FROM ``file:///File.
    csv" as row FIELDTERMINATOR ``;"
  RETURN row',
 'MERGE (RC:RegionCust [CUSTOMER_REGIONID :
    row.CUSTOMER_REGIONID,CUSTOMER_REGION:
    row.CUSTOMER_REGION])',
 [batchSize:10000, parallel:true]);
```

The provided script in Listing 1 only facilitates the creation of nodes and hierarchies but does not establish connections between them. The script in Listing 2 establishes relationships between nodes in the Neo4j database for the "Customer," "CityCustomer," and "RegionCustomer" hierarchies based on the data imported from the CSV file. For each row, the script looks for the "CUSTOMER" node with the matching "CUSTOMER-ID" property, and the "CityCust" node with the

[1] https://neo4j.com/product/neo4j-graph-database/

matching “CUSTOMER-CITYID” property using the MATCH clauses. The First MERGE clause creates a relationship of type CITY-CUSTOMER between the matched “CUSTOMER” and “CityCust” nodes. The second use of the MERGE clause creates a relationship of type REGION-CUSTOMER between the matched “CityCust” and “RegionCust” nodes.

Listing 2: The relationship between the Customer dimension and its hierarchies

```
1  // Relationship between Customer and
       CityCustomer
2  CALL apoc.periodic.iterate('LOAD CSV WITH
       HEADERS FROM ``file:///File.csv" as row
       FIELDTERMINATOR ``;"
3  RETURN row'
4  'MATCH (C:CUSTOMER [CUSTOMER_ID: row.
       CUTOMER_ID])
5  MATCH (CC:CityCust [CUSTOMER_CITYID: row.
       CUSTOMER_CITYID])
6  MERGE (C)-[:CITY_CUSTOMER]->(CC)',
7  [batchSize:2000, iteratelist:true]);
8
9  // Relationship between CityCustomer and
       RegionCustomer
10 CALL apoc.periodic.iterate(
11 'LOAD CSV WITH HEADERS FROM ``file:///File.
       csv" AS row FIELDTERMINATOR ``;"
12 RETURN row',
13 'MATCH (RC:RegionCust [CUSTOMER_REGIONID: row
       .CUSTOMER_REGIONID])
14 MATCH (CC:CityCust [CUSTOMER_CITYID: row.
       CUSTOMER_CITYID])
15 MERGE (CC)-[:REGION_CUSTOMER]->(RC)',
16 [batchSize:2000, iteratelist:true]);
```

To create the dimensions “PRODUCT”, “SUPPLIER”, and “Time”, we use a similar approach as employed for the “CUSTOMER” dimension.

After creating all the dimension nodes and their hierarchies in the same way, the next step is to create the fact node that contains the measures, and relationships between the fact and dimension nodes. The following script in Listing 3, demonstrates the creation of the fact node in Neo4j and the relationships between the fact and a dimension node.

The script uses the MERGE clause to create a node labeled as “FACT” with the specified properties (“ID”, “Price”, and “QUANTITY”) taken from the corresponding columns in the CSV file.

For each row, the script uses the MATCH clauses to find the “CUSTOMER” node with the matching “CUSTOMER-ID” and the “FACT” node with the matching “ID” property.

The MERGE clause creates a relationship labeled as “FACT-CUSTOMER” between the matched “FACT” and “CUSTOMER” nodes.

Listing 3: The Fact Node

```
1
2  // FACT NODE
3
4  CALL apoc.periodic.iterate(
5    'LOAD CSV WITH HEADERS FROM ``file:///File.
         csv" AS row FIELDTERMINATOR ``;"
6     RETURN row',
7    'MERGE (FCT:FACT [ID: row.INTEGRATION_ID,
         Price: row.O_TOTALPRICE, QUANTITY:
         toInteger(row.L_QUANTITY)])',
8    [batchSize:10000, parallel:true]);
9
10
11
12 // Relationship FACT/CUSTOMER
13
14 CALL apoc.periodic.iterate(
15 'LOAD CSV WITH HEADERS FROM ``file:///File.
       csv" AS row FIELDTERMINATOR ``;"
16 RETURN row',
17 'MATCH (C:CUSTOMER [CUSTOMER_ID: row.
       CUSTOMER_ID])
18 MATCH (FCT:FACT [ID: row.INTEGRATION_ID])
19 MERGE (FCT)-[:FACT_CUSTOMER]->(C)',
20 [batchSize:20000, iteratelist:true]);
```

In the same way, we create relationships between the fact node and the other dimensions (PRODUCT, SUPPLIER, TIME). Using also the apoc.periodic.iterate procedure along with MATCH and MERGE statements to efficiently import data from the CSV file and establish the relationships between the ”FACT” nodes and the corresponding nodes in the “PRODUCT”, “SUPPLIER”, and “TIME” dimensions in the Neo4j graph database.

The Fig. 2 represents the implementation of graph warehouse in Neo4j.

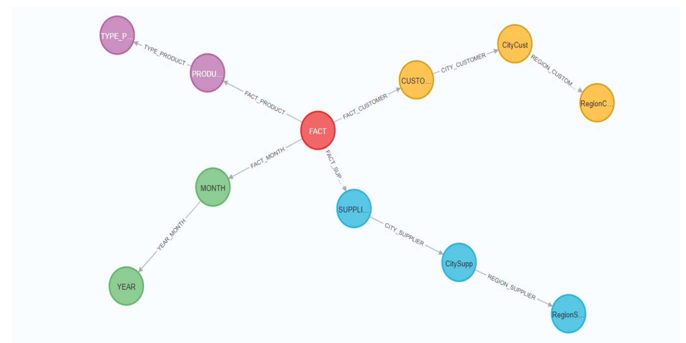

Fig. 2. The graph warehouse of our case study.

B. OLAP Operators

Slice

The slice operator in OLAP enables the selection of slices from the data based on a condition on the dimension values [7].

In Listing 4 the slice operator is applied to the Customer Region dimension using the filter condition “AFRICA”, which allows for selecting the data related to the AFRICA.

Listing 4: Selecting Price and Quantity Results from Africa.

```
1 MATCH (RC:RegionCust [CUSTOMER_REGION: '
    AFRICA'])<-[*3]-(m:FACT)
2 RETURN RC.CUSTOMER_REGION, sum(tofloat(m.
    Price)), sum(tofloat(m.QUANTITY))
```

Dice

The Dice operator is used in OLAP to select a subset of data based on two or more conditions on dimensions. It is similar to the Slice operator, but allows for finer selection by applying multiple criteria on dimensions simultaneously.

In Listing 5 the Dice operator is applied to the Customer Region and Year dimensions using the filter condition "AFRICA" and "1997".

Listing 5: Dice-Selecting Quantity Results with a Dice Operation.

```
1 MATCH (RC:RegionCust [CUSTOMER_REGION : '
    AFRICA'])<- [*3]-(m:FACT)
2 MATCH  (Y:YEAR [YEAR : 1994])<- [*2]-(m:FACT)
3 RETURN RC.CUSTOMER_REGION, Y.YEAR, sum(
    tofloat(m.QUANTITY)) as QUANTITY
```

Roll Up

In OLAP, [23]the Roll-Up operation is used to aggregate data at a higher level of hierarchy than the current level[24].It involves moving from a detailed level to a higher-level concept [25].The Roll-Up operation is performed by grouping the data based on the dimensions and then performing the aggregation function on the measures. The result is a summarized view of the data at a higher level of abstraction. In Listing 6, the Roll Up operation is carried out by moving up the Product dimension's concept hierarchy (Product → Product Type) and the hierarchy of dimension Time(Month → Year). This query creates a relationship between the two dimensions that contains the aggregated measures.

Listing 6: Roll Up- Price and Quantity summarized by Product Type and Year.

```
1 MATCH (TP:TYPE_PRODUCT)<-[*2]-(FCT:FACT)
2 MATCH (Y:YEAR)<-[*2]-(FCT:FACT)
3 WITH  distinct TP,Y,  sum(tofloat(FCT.Price))
     as Price, SUM(tofloat(FCT.QUANTITY)) As
    QUANTITY
4 create (TP)<-[:TYPRD_YEAR_AGG [Price: Price,
    QUANTITY: QUANTITY]]-(Y)
```

C. Graph Aggregations

After generating the most frequently user queries, the Apriori Algorithm was then used to determine the most common dimensions, and then we executed the second algorithm to generate the most commonly used combinations to create the aggregations. We set the support to 0.4 and the confidence to 0.7. The Fig. 3 shows the combinations of the most commonly used dimensions.

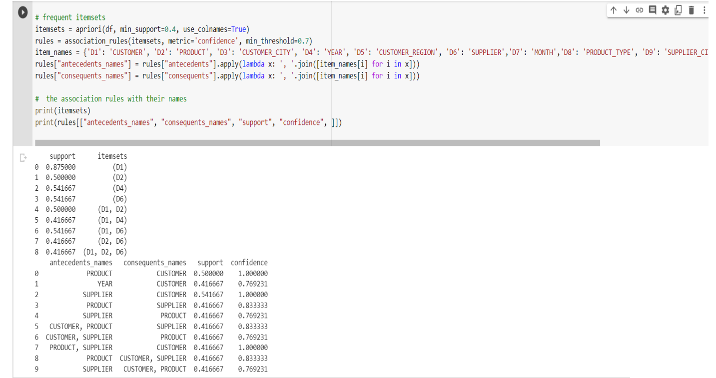

Fig. 3. The combinations of the most commonly used dimensions.

To create aggregations in Neo4j, we use the following script in Listing 7 that stores the aggregations in relationships:

Listing 7: Aggregation Customer-Year

```
1 CALL apoc.periodic.iterate(
2 'LOAD CSV WITH HEADERS FROM ''file:///File.
    csv" AS row FIELDTERMINATOR '';"
3 RETURN row',
4 'MATCH (C:CUSTOMER)<-[]-(FCT:FACT)
5 MATCH (Y:YEAR)<-[*2]-(FCT:FACT)
6 CREATE (C)<-[:CUST_YEAR_AGG [Price: toFloat(
    row.O_TOTALPRICE),QUANTITY: toFloat(row.
    L_QUANTITY)]]-(A)',
7 [batchSize:10000, parallel:true]);
```

In Listing 8, the script is used to create aggregations that are stored in nodes, not in relationships.

Listing 8: Aggregation Customer-Product-Supplier

```
1 CALL apoc.periodic.iterate(
2 'LOAD CSV WITH HEADERS FROM ''file:///File.
    csv" AS row FIELDTERMINATOR '';"
3 RETURN row',
4 'MATCH (P:PRODUCT)<-[]-(FCT:FACT)
5 MATCH (C:CUSTOMER)<-[]-(FCT:FACT)
6 MATCH (S:SUPPLIER) <-[]-(FCT:FACT)
7 create (PCS:PROD_CUST_SUPP [ PRICE:  toFloat(
    row.O_TOTALPRICE),QUANTITY: toFloat(row.
    L_QUANTITY)])
8 create (P)<-[:PROD_3]-(PCS)
9 create(C)<-[:CUST_3]-(PCS)
10 create (S)<-[:SUPP_3]-(PCS)',
11 [batchSize:10000, parallel:true]);
```

The Fig. 4 shows the creation of aggregations using in approach.

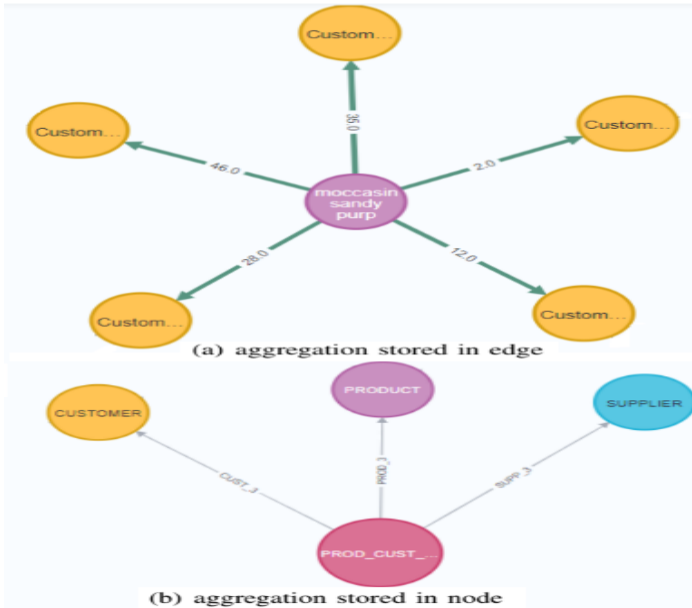

Fig. 4. Stored aggregations.

D. Experimental Results and Evaluation

To validate our approach and measure the effectiveness of optimizing the OLAP cube in the graph, we conducted a series of experiments in which we evaluated performance before and after using our optimization approach. We measured the query execution time before and after adding optimized aggregations, and compared the execution times to determine if adding the optimized aggregations led to a significant improvement in performance. We conducted our test on an i7 processor machine with 16GB of RAM and 1TB of storage memory. Additionally, we used the TPC-H database with a scale factor of SF1 (1GB). We use in Table II, Cypher queries before and after optimization.

TABLE II. CYPHER QUERIES BEFORE AND AFTER OPTIMIZATION

Query	Before Optimization	After Optimization
Q 1	MATCH (RS:RegionSupp) <-[*3]-(FCT:FACT) return SUM(tofloat(FCT.Price)) AS Price, RS.SUPPLIER_REGION	MATCH (RS:RegionSupp) <-[:REGION_SUPP_AG]- (RA:Region_Supp_AGG) return RA.Price ,RS.SUPPLIER_REGION
Q 2	MATCH(Y:YEAR)<- [*2]-(m:FACT) MATCH(TP:TYPE_PRODUCT) <-[*2]-(m:FACT) return Y.YEAR, TP. BRAND ,sum(tofloat(m.Price))as Price	MATCH(TP:TYPE_PRODUCT) <-[r:TYPRD_YEAR_AGG] -(Y:YEAR) return Y.YEAR, TP. BRAND ,sum(tofloat(r.Price)) as Price
Q 3	MATCH (C:CUSTOMER) <-[] -(FCT:FACT) MATCH (P:PRODUCT)<- []-(FCT:FACT) return sum(tofloat(FCT.Price)) as Price ,C.CUSTOMER_NAME,P. PRODUCT_NAME	MATCH (P:PRODUCT)- [r:AGG_CUST_PROD]->(C :CUSTOMER) return r.Price,C. CUSTOMER_NAME ,P.PRODUCT_NAME
Q 4	MATCH (TP: TYPE_PRODUCT) <-[*2]-(FCT:FACT) return SUM(tofloat(FCT.QUANTITY)) As QUANTITY, TP.BRAND	MATCH (TP: TYPE_PRODUCT) <-[:TYPE_PROD_AG] -(TPAGG:TYPE_PROD_AGG) return TPAGG.QUANTITY , TP.BRAND
Q 5	MATCH (P:PRODUCT)<- [:FACT_PRODUCT]-(FCT: FACT) MATCH (C:CUSTOMER)<- [:FACT_CUSTOMER]-(FCT :FACT) MATCH (S:SUPPLIER)<- [:FACT_SUPPLIER]-(FCT :FACT) return SUM(tofloat(FCT.Price)) AS PRICE, P.PRODUCT_NAME,C. CUSTOMER_NAME, S.SUPPLIER_NAME LIMIT 43	MATCH (P:PRODUCT)<-[: PROD_3] -(PCS:PROD_CUST_SUPP) MATCH (C:CUSTOMER) <-[:CUST_3] -(PCS:PROD_CUST_SUPP) MATCH (S:SUPPLIER) <-[:SUPP_3] -(PCS:PROD_CUST_SUPP) return PCS.PRICE,P. PRODUCT_NAME, C.CUSTOMER_NAME,S. SUPPLIER_NAME LIMIT 43

The Fig. 5 shows the query execution time before and after the optimization of the model.

Fig. 5. Query execution time.

The results demonstrate that the execution time of the queries decreased after the optimization and usage of OLAP aggregations, although the execution time may vary depending on the complexity of the query, with complex queries requiring traversal of numerous relationships taking more time, while simple queries involving only a few relationships having relatively short execution times.

For instance, in the first query, the execution time before optimization was approximately 14266 milliseconds, and after optimization, it reduced to 3 milliseconds. This translates to an impressive percentage improvement of approximately 99%. Furthermore, in the query 4, we achieved considerable improvements as well, the execution time decreased from 3,978 milliseconds to 2 milliseconds,substantial gains in performance clearly demonstrate the effectiveness of our approach in making the system nearly 2,000 times faster than its previous state.

The significant reduction in execution time showcases how this approach can make the system multiple times faster than its previous state, enhancing the efficiency of reporting and data analysis within the graph-based data warehouse.

We also compared our model implemented in the graph and the ROLAP model, comparing the execution time of the same multidimensional queries in Neo4j and Oracle.

Table III shows the queries used in the comparison between Graph OLAP and ROLAP.

TABLE III. RELATIONAL QUERY VS GRAPH QUERY

Query	Relational Query	Graph Query
Q 1	SELECT **sum**(o_totalprice) as Price, supplier_region **from** w_fact_1gb_f3 Group by supplier_region	MATCH (RS:RegionSupp) <-[:REGION_SUPP_AG]- (RA:Region_Supp_AGG) **return** RA.Price ,RS.SUPPLIER_REGION
Q 2	SELECT **sum**(o_totalprice) as Price, orderyear AS YEAR, brand as PRODCUT_TYPE **from** w_fact_1gb_f3 GROUP BY orderyear, brand	MATCH(TP:TYPE_PRODUCT) <-[r:TYPRD_YEAR_AGG] -(Y:YEAR) **return** Y.YEAR, TP. BRAND ,**sum**(tofloat(r.Price)) as Price
Q 3	SELECT **sum**(o_totalprice) as Price, product_name, customer_name **from** w_fact_1gb_f3 GROUP BY product_name, customer_name	MATCH (P:PRODUCT)- [r:AGG_CUST_PROD]->(C :CUSTOMER) **return** r.Price,C. CUSTOMER_NAME ,P.PRODUCT_NAME
Q 4	SELECT **sum**(l_quantity) as QUANTITY, brand as PRODUCT_TYPE FROM w_fact_1gb_f3 GROUP BY brand	MATCH (TP: TYPE_PRODUCT) <-[:TYPE_PROD_AG] -(TPAGG:TYPE_PROD_AGG) **return** TPAGG.QUANTITY , TP.BRAND
Q 5	select * **from** (SELECT **sum**(o_totalprice) as Price, product_name, customer_name, supplier_name FROM w_fact_1gb_f3 GROUP BY product_name, customer_name, supplier_name) where ROWNUM <= 43;	MATCH (P:PRODUCT)<-[: PROD_3] -(PCS:PROD_CUST_SUPP) MATCH (C:CUSTOMER) <-[:CUST_3] -(PCS:PROD_CUST_SUPP) MATCH (S:SUPPLIER) <-[:SUPP_3] -(PCS:PROD_CUST_SUPP) **return** PCS.PRICE,P. PRODUCT_NAME, C.CUSTOMER_NAME,S. SUPPLIER_NAME LIMIT 43

The Figure 6 shows the query execution time in the Graph OLAP and ROLAP.

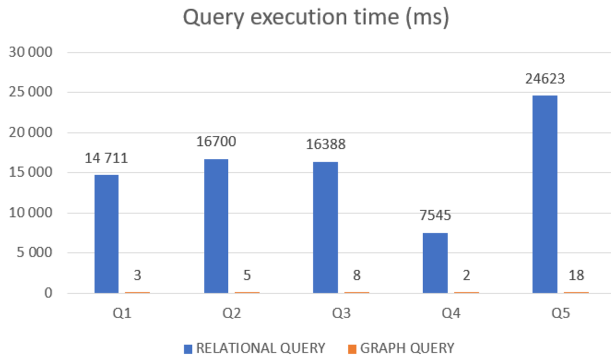

Fig. 6. Query execution time in Oracle and Neo4j.

The results also show that the Graph cube delivers performance levels that are better than those of the ROLAP model.we also notice that Graph databases provide a great degree of flexibility for searching through data by conducting more complex pathways or by following direct linkages. While SQL queries that use joins to mix data from various tables provide the basis for data traversal in a relational architecture. While joins can be effective, they can also be less flexible and intuitive when navigating complex relationships.

V. Conclusion

Graph-Oriented Databases offers a clear abstraction for managing heavily connected data and modelling complicated domains. In this Paper we present our contribution for developing a data warehouse under a graph database, our approach relies on the properties of graphs to implement graph data warehouse. To enhance the graph cube's performance we provide a new technique that optimizes the choice of OLAP aggregations by using the association rules algorithm.

To validate our approach and measure the effectiveness of OLAP cube optimization in the graph, we conducted a series of experiments in which we evaluated the performance before and after the optimization, we also compared our model with the relational model in terms of query performance. The experiment's findings demonstrate the benefits of creating OLAP systems under graph oriented databases when using a large amount of data. In our future research works we will concentrate on the implementation of decision systems in other Nosql databases such as document and column databases using new approaches.

References

[1] Aqib Ali, Samreen Naeem, Sania Anam, and Muhammad Munawar Ahmed. A state of art survey for big data processing and nosql database architecture. *IJCDS Journal*, May 2023.

[2] Amine Ghrab, Oscar Romero, Sabri Skhiri, and Esteban Zimányi. Topograph: an end-to-end framework to build and analyze graph cubes. *Information Systems Frontiers*, 23(1):203–226, 2021.

[3] Matteo Kamm, Manuel Rigger, Chengyu Zhang, and Zhendong Su. Testing graph database engines via query partitioning. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 140–149, 2023.

[4] Amine Ghrab, Oscar Romero, Sabri Skhiri, Alejandro Vaisman, and Esteban Zimányi. A framework for building olap cubes on graphs. In *Advances in Databases and Information Systems: 19th East European Conference, ADBIS 2015, Poitiers, France, September 8-11, 2015, Proceedings 19*, pages 92–105. Springer, 2015.

[5] Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. Graph cube: on warehousing and OLAP multidimensional networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 853–864, Athens Greece, June 2011. ACM.

[6] Michal Bachman. Graphaware: Towards online analytical processing in graph databases. *Department of Computing, Master*, 2013.

[7] Arnaud Castelltort and Anne Laurent. Fuzzy queries over nosql graph databases: perspectives for extending the cypher language. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems: 15th International Conference, IPMU 2014, Montpellier, France, July 15-19, 2014, Proceedings, Part III 15*, pages 384–395. Springer, 2014.

[8] Alejandro Vaisman, Florencia Besteiro, and Maximiliano Valverde. Modelling and querying star and snowflake warehouses using graph databases. In *New Trends in Databases and Information Systems: ADBIS 2019 Short Papers, Workshops BBIGAP, QAUCA, SemBDM, SIMPDA, M2P, MADEISD, and Doctoral Consortium, Bled, Slovenia, September 8–11, 2019, Proceedings 23*, pages 144–152. Springer, 2019.

[9] Amal Sellami, Ahlem Nabli, and Faiez Gargouri. Transformation of data warehouse schema to nosql graph data base. In *Intelligent Systems Design and Applications: 18th International Conference on Intelligent Systems Design and Applications (ISDA 2018) held in Vellore, India, December 6-8, 2018, Volume 2*, pages 410–420. Springer, 2020.

[10] Nassima Soussi. Big-Parallel-ETL: New ETL for Multidimensional NoSQL Graph Oriented Data. *Journal of Physics: Conference Series*, 1743(1):012037, January 2021.

[11] Hajer Akid, Gabriel Frey, Mounir Ben Ayed, and Nicolas Lachiche. Performance of nosql graph implementations of star vs. snowflake schemas. *IEEE Access*, 10:48603–48614, 2022.

[12] Abdelhak Khalil and Mustapha Belaissaoui. A Graph-oriented Framework for Online Analytical Processing. *International Journal of Advanced Computer Science and Applications*, 13(5), 2022.

[13] Redha Benhissen, Fadila Bentayeb, and Omar Boussaid. GAMM: graph-based agile multidimensional model. In Enrico Gallinucci and Lukasz Golab, editors, *Proceedings of the 25th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) co-located with the 26th International Conference on Extending Database Technology and the 26th International Conference on Database Theory (EDBT/ICDT 2023), Ioannina, Greece, March 28,*

2023, volume 3369 of *CEUR Workshop Proceedings*, pages 23–32. CEUR-WS.org, 2023.

[14] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2011.

[15] Alejandro Vaisman and Esteban Zimányi. Data warehouse systems. *Data-Centric Systems and Applications*, 2014.

[16] Yiming Lin, Yeye He, and Surajit Chaudhuri. Auto-bi: Automatically build bi-models leveraging local join prediction and global schema graph. *arXiv preprint arXiv:2306.12515*, 2023.

[17] Ajith Abraham, Aswani Kumar Cherukuri, Patricia Melin, and Niketa Gandhi. *Intelligent Systems Design and Applications: 18th International Conference on Intelligent Systems Design and Applications (ISDA 2018) Held in Vellore, India, December 6-8, 2018, Volume 1*. Springer, 2020.

[18] Khadija Letrache, Omar El Beggar, and Mohammed Ramdani. Olap cube partitioning based on association rules method. *Applied Intelligence*, 49:420–434, 2019.

[19] Rakesh Agarwal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, volume 487, page 499, 1994.

[20] Faaiz Hussain Shah. *Gradual Pattern Extraction from Property Graphs*. PhD thesis, Université Montpellier, 2019.

[21] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases: new opportunities for connected data*. " O'Reilly Media, Inc.", 2015.

[22] Mohammed El Malki, Arlind Kopliku, Essaid Sabir, and Olivier Teste. Benchmarking big data olap nosql databases. In *Ubiquitous Networking: 4th International Symposium, UNet 2018, Hammamet, Tunisia, May 2–5, 2018, Revised Selected Papers 4*, pages 82–94. Springer, 2018.

[23] Adriana P Matei. *An integrated approach to deliver OLAP for multidimensional Semantic Web Databases*. PhD thesis, Coventry University, 2015.

[24] Elaheh Pourabbas and Maurizio Rafanelli. Characterization of hierarchies and some operators in olap environment. In *Proceedings of the 2nd ACM International Workshop on Data Warehousing and OLAP*, pages 54–59, 1999.

[25] Hans-Joachim Lenz and Bernhard Thalheim. A formal framework of aggregation for the olap-oltp model. *J. Univers. Comput. Sci.*, 15(1):273–303, 2009.