Performance of NoSQL Graph Implementations of Star vs. Snowflake Schemas

Ajer Akid, Gabriel Frey, Mounir Ben Ayed, Nicolas Lachiche

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 1, 2022, accepted April 20, 2022, date of publication May 5, 2022, date of current version May 11, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3171256

Performance of NoSQL Graph Implementations of Star vs. Snowflake Schemas

HAJER AKID[1,2], (Member, IEEE), GABRIEL FREY[1], MOUNIR BEN AYED[2], (Senior Member, IEEE), AND NICOLAS LACHICHE[1]

[1]ICube—Laboratory of Engineering, Computer Science and Imagery, University of Strasbourg, Illkirch, 67412 Cedex, France
[2]REGIM-Lab—REsearch Groups in Intelligent Machines, National Engineering School of Sfax (ENIS), University of Sfax, Sfax 3038, Tunisia

Corresponding author: Hajer Akid (akid.hajer@ieee.org)

This work was supported by the Ministry of Higher Education and Scientific Research of Tunisia under Grant LR11ES48.

ABSTRACT Nowadays, the data used for decision-making come from a wide variety of sources which are difficult to manage using relational databases. To address this problem, many researchers have turned to Not only SQL (NoSQL) databases to provide scalability and flexibility for On-Line Analytical Processing (OLAP) systems. In this paper, we propose a set of formal rules to convert a multidimensional data model into a graph data model (MDM2G). These rules allow conventional star and snowflake schemas to fit into NoSQL graph databases. We apply the proposed rules to implement star-like and snowflake-like graph data warehouses. We compare their performances to similar relational ones focusing on the data model, dimensionality, and size. The experimental results show large differences between relational and graph implementations of a data warehouse. A relational implementation performs better for queries on a couple of tables, but conversely, a graph implementation is better when queries involve many tables. Surprisingly the performances of a star-like and snowflake-like graph data warehouses are very close. Hence a snowflake schema could be used in order to easily consider new sub-dimensions in a graph data warehouse.

INDEX TERMS Data model, graph data warehouse, NoSQL, performance, relational data warehouse.

I. INTRODUCTION

The amount of digital data generated every day is expanding rapidly. This phenomenon is labeled as ''Big Data'' which refers to large volumes of high velocity, complex and heterogeneous data which require advanced techniques and technologies to enable the capture, storage, distribution, management, and analysis of the information [1]. Today, a challenging issue is to design and build a decision support system (DSS) that enables access to big data and provides correct and fast answers to complex analytical queries. Consequently, nowadays, many researchers from different fields are working on the improvement of conventional decision-making systems to address big data requirements.

Traditionally, a DSS incorporates all data relevant to the management of an organization into a specific repository used for analytical purposes named data warehouse. As defined in [2], a data warehouse is a ''subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision-making process and business intelligence''. Generally, a data warehouse is designed using a multidimensional data model that provides an understandable business view of the database [3]. The implementation of a data warehouse involves applying a specific approach to convert its conceptual multidimensional data model into a target logic model [4]. Several approaches have been proposed [3], [5]–[7], the most popular being Relational Online Analytical Processing (R-OLAP) which converts the conceptual multidimensional model into a relational one [8] using a star schema or a snowflake schema. In the context of relational databases, the snowflake schema is known for being less efficient than the star schema due to the high cost of join operators [9].

Undeniably, relational database management systems (RDBMS) have dominated the database management landscape since the 1970s mainly for storing and retrieving structured data. However, despite their maturity, the relational databases are currently facing many challenges as they were designed neither to provide good scalability and deal efficiently with a huge amount of data [10], nor to cope with unstructured data [11]. Hence, to meet these needs, a new

The associate editor coordinating the review of this manuscript and approving it for publication was Genoveffa Tortora.

range of database management systems labeled as NoSQL (Not Only SQL), not based on relational models, has arisen. NoSQL systems have been mainly introduced to integrate large, unstructured and complex data generated from multiple sources such as social networks, interconnected devices, and sensors in order to make better decisions. Generally, NoSQL databases are defined through a set of features which are mainly flexibility, high availability, scalability, and low-cost requirements [12], [13]. NoSQL systems are commonly classified into four main types depending on their logical model: key-value oriented stores, column-oriented stores, document-oriented stores, and graph-oriented stores [14]–[16]. The emergence of NoSQL systems has enriched the database management landscape. Consequently, the choice of which database to use has become harder than before. Over the last few years, many insightful research works have studied the effectiveness of using NoSQL systems to implement big data warehouses [17], [18]. Three major categories of approaches have been considered: column-based approaches, document-based approaches, and graph-based approaches. These approaches allow transforming the conceptual multidimensional model of a data warehouse to a target NoSQL logical model using a set of transformation rules. Most of the proposed approaches focused on column-oriented [19]–[26] and document-oriented [27]–[29] NoSQL models. Some of them provided a performance evaluation based on some criteria such as read latency [19], [27] and write latency [27], [28]. However, only few and recent studies considered the NoSQL graph-oriented model. They focused either on the performances of graph versus relational databases under various uses [30], [31], or on designing graph data warehouses and defining graph OLAP operators (G-OLAP) [32]–[35]. However, to the best of our knowledge, the respective performances of graph implementations of normalized (snowflake) versus denormalized (star) data warehouses have not been evaluated yet.

In this paper, we present a new approach to convert a multidimensional data model to a graph database (MDM2G) which encompasses a set of transformation rules to convert star and snowflake relational multidimensional models to star-like and snowflake-like graph data models. We provide a formal definition for each model and we evaluate their performance to figure out whether a snowflake-like model would be highly time-consuming in the context of a graph database as it is in a relational database. In addition, we compare the performance of graph data warehouses to analogous relational star and snowflake logical models to determine whether a graph data warehouse could be more efficient than a relational one. Our motivation for investigating a graph database is its performance when dealing with connected data compared to relational databases and other NoSQL logical models [36]. In fact, the join mechanism of relational databases is time consuming. Also, column-oriented and document-oriented databases lack relationships and require adapting their models to store and query complex data. On the contrary, graph databases store physical relationships that facilitate graph traversals between entities.

The remaining of the paper is organized as follows: section 2 gives an overview of the proposed approaches in the literature for implementing NoSQL data warehouses; section 3 describes our approach which enables modeling data warehouses using graphs; section 4 details our experiments; section 5 reports and analyses our results. Our conclusions and research perspectives are presented in section 6.

II. RELATED WORKS: NOSQL DATA WAREHOUSES

Most decision support systems are based on data warehousing techniques to take advantage of data collected from heterogeneous sources. Data warehouses allow decision-makers to have a global and synthetic view of the information circulating in their companies. Generally, data warehouses organize data according to a multidimensional conceptual model considering an analyzed subject as a point in a space which could be observed through several dimensions [3]. Conceptually, a multidimensional model is composed of the concepts of fact, dimensions and hierarchies. The fact is the entity being analyzed. It consists of one or more measures. The dimensions are the axis of analysis which allow the evaluation of the fact. They contain one or more attributes that are used to vary the measures of the analyzed activity. One distinguishes between the parameters which are attributes defining the levels of granularity and the weak attributes which are informational attributes related to the parameters. These different levels make it possible to respond to different queries, depending on the analytical needs. A hierarchy allows ordering the parameters of a dimension according to their level of granularity or detail. Three approaches were proposed to build logical models suitable for a data warehouse: R-OLAP (Relational OLAP) [3], [5], M-OLAP (Multidimensional OLAP) [6] and H-OLAP (Hybrid OLAP) [7] approaches. R-OLAP is the oldest and predominant storage strategy. It makes it possible to transform the concepts of fact and dimension of a multidimensional conceptual model into relational tables.

Three multidimensional designs have been defined in this approach to simulate a multidimensional structure in a relational database, namely: star, snowflake, and constellation schema [37], [38]. A star schema includes a central fact table and many dimensions tables. This model represents the dimensions in a denormalized way. Each dimension table is joined to the fact table using its primary key, transformed in foreign key in the fact table. However, the dimensions are not joined together. A snowflake schema is an extension of the star schema in which some dimensions are hierarchical. It consists of keeping the same fact table and normalizing the dimension tables into sub-dimensions in order to allow a more explicit representation of the hierarchy. So, the dimensions are described through a succession of tables using foreign keys. A constellation schema involves several star schemas. Therefore, it contains many tables of fact and dimensions

which could be shared or not. Obviously, a snowflake data model is more complex than a star data model. In most cases, this complexity impacts the performance of the data warehouse as more join operations are required to answer queries.

Since the arrival of NoSQL systems, many researchers have compared it to relational systems based on different requirements such as scalability [10], [30], [38], [39]. Further research works have focused on proposing approaches to allow data migration from relational databases to column NoSQL stores [40], document NoSQL stores [41]–[43] or graph NoSQL stores [30], [44], [45]. Recently, using NoSQL database management systems to implement big data warehouses able to gather voluminous and heterogeneous data to take better decisions have attracted researchers and organization. When looking at all the proposed approaches, three major categories can be identified: column-based approaches, document-based approaches, and graph-based approaches.

A. COLUMN-BASED APPROACHES

These approaches allow data warehouses to be implemented under column-oriented NoSQL systems. In [20], [24] authors have proposed a set of transformation rules to convert facts, measures, dimensions, and attributes to columnar concepts. More precisely, facts and dimensions are transformed into column families where measures and attributes are stored in columns. These studies considered the case of a star schema and did not consider hierarchies. In [19], the authors proposed three methods to enable the implementation of the columnar data warehouse. The first method allows the storage of facts and dimensions in the same column family. The second method stores facts and dimensions separately. Each fact table is transformed into a column family that contains measures as columns. The dimensions are also transformed into column families having attributes as columns. This method models and stores a star schema. The third method considers hierarchies. Each attribute of a dimension is stored in a separate column family. The results of this work showed that the storage of hierarchies in column-oriented stores is highly time-consuming. The findings of [19] demonstrate also that splitting the attributes of dimensions in different column families affects the performance of the columnar data warehouse. In [46], authors focus on building OLAP columnar NoSQL cubes and evaluate their performances.

B. DOCUMENT-BASED APPROACHES

Many approaches have been proposed to transform the concepts of multidimensional conceptual model into document-oriented model concepts. In [20], the authors proposed to convert each fact into a collection of documents that contains measures. Each dimension is also transformed into a collection of documents that contains the different attributes (parameter and weak attributes) in forms of documents. In this work, hierarchies were not studied. In [47], the authors proposed three methods. In the first method, facts and dimensions are stored in the same collection of documents. In the second method, each fact and related measures is stored in a collection of documents. Each dimension and its related attributes are stored a separate collection of documents. Hierarchies were not studied in both these methods. The last set of transformation rules enables the storage of fact and measures in a collection of documents. Parameters of dimensions are normalized in different collections of documents having the weak attributes as embedded documents. This study revealed that modeling and storing hierarchies using the concept of embedded documents decreases significantly the performance of queries that perform many joins. Regardless of the data warehousing context, another research work [48] reported a study on the impact of structuring data in forms of embedded documents. Experiments demonstrated that querying data stored at different levels in a collection of documents require complex manipulation and more time to be executed.

C. GRAPH-BASED APPROACHES

Graph databases are composed of nodes and edges tagged with labels. Both nodes and edges can store properties by means of key/value pairs. In order to implement a graph data warehouse, [32] proposed to transform facts into nodes. The measures of each fact are stored as properties in the same node. Also, dimensions are transformed into nodes. There are two types of relations between nodes. The first type of relationship is labeled FACT which links fact to dimensions. The second type is labeled HIER which links the attributes of dimensions. This work focused on adapting Cypher query language to support OLAP operators mainly Slice, Dice and Roll up ones. Some experimental tests have been conducted to validate the proposed approach. However, the authors considered only the case of snowflake schema and did not study the effectiveness of the graph data warehouse especially when queries get more complex or the database gets larger. In [33], [34], the authors provide formal transformation rules to convert a multidimensional conceptual model into NoSQL graph-oriented model. Yet, the proposed data warehouses were not evaluated. The performances of relational versus graph databases were evaluated in [30], [31] but not from an OLAP perspective with respect to normalized versus denormalized schemas.

In the absence of performance evaluation of graph data warehouses and with increasing interests to graphs as a native tool to answer complex queries, we provide in this paper a new approach to convert a multidimensional data model to graph database (MDM2G) that we evaluate based on two metrics: write latency and read latency.

III. PROPOSED APPROACH: MULTIDIMENSIONAL DATA MODEL TO GRAPH DATABASE (MDM2G)

The R-OLAP approach allows transforming the multidimensional data model of a data warehouse into relational logical models in the form of star or snowflake schemas. These relational logical models are automatically generated from conceptual models by applying a set of rules [49]. Using these transformation rules in the context of big data has

many weaknesses ascribed to the limitations of the relational data model mainly when queries require multiple complex aggregations. To address this problem, we propose converting the multidimensional data model of a data warehouse to a graph database (MDM2G) by mapping the concepts of the multidimensional data model (facts, dimensions, etc.) into graph concepts. We provide in this section a formal definition of MDM2G transformation rules. These rules enable the definition of two graph data warehouses having a star-like schema or a snowflake-like schema.

A. MULTIDIMENSIONAL DATA MODEL

In order to define our rules, we first define the concepts of the source data model which is the multidimensional conceptual model.

Definition 1: A multidimensional model denoted MDM, is formally defined [20], [50] by the triplet (F^{MDM}, D^{MDM}, $Star^{MDM}$) where:

- $F^{MDM}=\{f_1,\ldots,f_n\}$ is a finite set of facts,
- $D^{MDM}=\{d_1,\ldots,d_m\}$ is a finite set of dimensions,
- $Star^{MDM}$: $F^{MDM}\longrightarrow 2^{D^{MDM}}$ is a function that associates each fact $F_i\in F^{MDM}$ to a set of dimensions $D_i\in D^{MDM}$.

Definition 2: A fact, denoted $F_i \in F^{MDM}$, is defined by ($Name^{F_i}$, M^{F_i}) where:

- $Name^{F_i}$ is the name of the fact,
- $M^{F_i}=\{m_1,\ldots,m_{p_i}\}$ is a set of measures.

Definition 3: A dimension, denoted $D_i \in D^{MDM}$ is defined by ($Name^{D_i}$, A^{D_i}, H^{D_i}) where:

- $Name^{D_i}$ is the name of the dimension,
- $A^{D_i}=\{a_1,\ldots,a_{r_i}\}$ is a set of dimension attributes,
- $H^{D_i}=\{h_1,\ldots,h_{s_i}\}$ is a set of hierarchies.

Definition 4: A hierarchy of the dimension D_i, denoted $H_j \in H^{D_i}$ is defined by ($Name^{H_j}$, $Param^{H_j}$, $Weak^{H_j}$) where:

- $Name^{H_j}$ is the name of the hierarchy,
- $Param^{H_j}=\{param_1{}^{H_j},\ldots,param_{q_j}{}^{H_j}\}$ is a set of attributes called parameters of hierarchy,
- $Weak^{H_j}$ is a function associating with each parameter zero or more weak attributes.

For instance, in Figure 2, ''Store_Sales'' is a fact having ''ss_quantity'' as a measure. ''Date_Dim'' and ''Customer'' are the dimensions of ''Store_Sales''. The dimension ''Customer'' has a hierarchical structure made up of ''Customer_Demographics'' and ''Income_Band''.

B. PROPERTY GRAPH DATA MODEL

The target model of our transformation rules is a property graph model. Graph data models have arisen since the eighties, but their popularity gradually decreased with the emergence of other data models, especially the geographical, spatial, semi-structured and XML [51]. Recently, graph databases have regained the attention of both academics and business entities due to the ever-increasing need to store, process, manage and analyze graph-like structures such as social networks [52], [53], biological networks [54]–[56], and document networks [57], [58]. Indeed, graph databases are considered as one of the most useful structures and natural ways for modeling interactions between the objects of a network [9]. Many graph database management systems are available today such as Neo4j [36] and GraphDB [59]. A database schema as well as instances in this model are a labeled directed graph, where the nodes represent objects and edges represent the connections between them. Whereas relational databases require expensive join operations to answer complex queries, graph databases consider the relationships between entities as important as the entities themselves [60] which facilitates the navigation between entities.

From a conceptual view, there are two graph data models: the property graph (PG) allowing both nodes (vertices) and edges to have any number of arbitrary properties and the Resource Description Framework (RDF) originally designed to represent information about resources on the World Wide Web. The most used model is the property graph model [36]. Informally, a PG is a directed labeled graph where data is represented by means of nodes, edges, and properties (key-value pairs). The nodes represent entities and the edges represent relationships between them. Both nodes and edges can be tagged with one or more labels and contain properties which represent their features.

Let us define L, P and V such as:

- $L=\{l_1,\ldots,l_a\}$ is an infinite set of labels,
- $P=\{p_1,\ldots,p_b\}$ is an infinite set of property names,
- $V=\{v_1,\ldots,v_c\}$ is a finite set of atomic values.

Definition 5: A property graph data model, namely G, is formally defined [61] by (N^G, E^G, ρ^G, λ^G, σ^G) where:

- $N^G=\{n_1,\ldots,n_j\}$ is a finite set of nodes,
- $E^G=\{e_1,\ldots,e_k\}$ is a finite set of edges,
- ρ^G: $E^G\longrightarrow (N^G\times N^G)$ is a total function that associates each edge in E^G with a pair of nodes (source and target nodes) in N^G,
- λ^G: $(N^G\cup E^G)\longrightarrow L$ is a partial function that associates nodes and edges to a set of labels from L,
- σ: $(N^G\cup E^G)\times P\longrightarrow V$ is a partial function that associates nodes and edges with properties, and for each property it assigns a value from V.

C. MDM2G: STAR-LIKE SCHEMA

In the context of relational databases, the star design transforms each fact of the multidimensional conceptual model to a relational fact table. The fact table contains measures as columns. In addition, each dimension is converted to a denormalized dimension table which contains all the attributes (parameters and weak attributes) as columns. Each instance of fact and dimension tables is stored in a specific row. In the same way, we use the previously mentioned definitions of multidimensional model and property graph concepts to propose our transformation rules which define a star-like graph schema.

Transformation 1: Each multidimensional data model MDM(F^{MDM}, D^{MDM}, $Star^{MDM}$) is transformed into

a multidimensional graph data model MGD(N^{MGD}, E^{MGD}, ρ^{MGD}, λ^{MGD}, σ^{MGD}) where:

- $N^{MGD}=\{n_1,\ldots,n_j\}$ is a finite set of facts and dimensions nodes,
- ρ^{MGD}: $E^{MGD}\longrightarrow (N^{MGD}\times N^{MGD})$ is a total function that associates each edge in E^{MGD} with a source fact node and a target parameter nodes in N^{MGD},
- λ^{MGD}: $(N^{MGD}\cup E^{MGD})\longrightarrow L$ is a partial function that associates facts and dimensions nodes and edges to a set of labels from L,
- σ: $(N^{MGD}\cup E^{MGD})\times P\longrightarrow V$ is a partial function that associates facts and dimensions nodes and edges with properties, and for each property it assigns a value from V.

Transformation 2: Each fact $F_i(Name^{F_i}, M^{F_i})\in F^{MDM}$ is transformed into a set of fact nodes $N_{F_i}\in N^{MGD}$ defined by ($Name^{F_i{}^{MGD}}$, $M^{F_i{}^{MGD}}$) where:

- $Name^{F_i{}^{MGD}}$ is the name of the fact F_i associated with the function λ^{MGD} as a label to the fact nodes N_{F_i},
- $M^{F_i{}^{MGD}}$ is a set of measures of the fact F_i associated with the function σ^{MGD} to the fact nodes N_{F_i} as properties. The value of the measure is stored as a value of the property.

  This rule creates as many fact nodes as instances of the fact. Figure 1 illustrates this transformation rule. In our example, the fact ''Store_Sales'' turns into a set of nodes with the same fact label ''Store_Sales'' having ''ss_ticket_number'' and ''ss_quantity'' as measure properties.

Transformation 3: Each dimension $D_i(Name^{D_i}, A^{D_i}, H^{D_i})\in D^{MDM}$ is transformed into a set of dimension nodes $N_{D_i}\in N^{MGD}$ defined by ($Name^{D_i{}^{MGD}}$, $A^{D_i{}^{MGD}}$) where:

- $Name^{D_i{}^{MGD}}$ is a the name of the dimension D_i associated with the function λ^{MGD} as a label to the dimension nodes N_{D_i},
- $A^{D_i{}^{MGD}}$ is a set of attributes (parameters and weak attributes) of the dimension D_i associated with the function σ^{MGD} as properties in the parameter nodesN_{D_i}. Hence, hierarchies are not taken into consideration,
- An edge is defined between each source fact node N_{F_i} and target parameter nodesN_{D_i} using the the function ρ^{MGD}.

This rule creates for each dimension as many nodes as its instances. The figure 1 illustrates the transformation of dimensions and their attributes. In our example, the dimension ''Customer'' is transformed into a set of nodes having the same label ''Customer''. All the attributes which give details about customers are transformed into properties in the ''Customer'' nodes. In this transformation, all the nodes of the dimensions are directly linked to the fact nodes using edges. Hence, the star-like schema allows querying the multidimensional graph data model using one-level graph traversals. In this case, the depth, which is the number of paths between a fact node and a dimension node, is equal to one. Figure 1 shows the transformation of the joins between the fact table

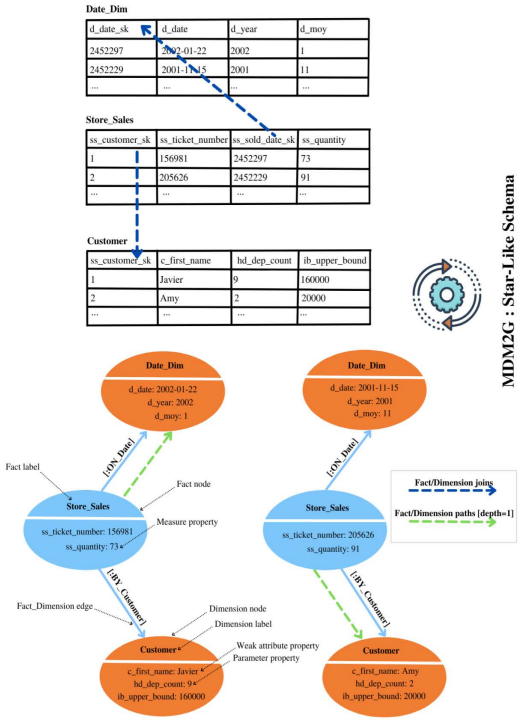


FIGURE 1. MDM2G: Star-Like schema.

''Store_Sales'' and dimension table ''Customer'' into a set of edges labelled '':BY_Customer''.

D. MDM2G: SNOWFLAKE-LIKE SCHEMA

In contrast to the star data model where all the parameters are grouped in a single denormalized dimension table, the snowflake data model allows representing hierarchies using several sub-dimensions which are smaller and normalized relational tables. Hence, we propose a snowflake-like multidimensional schema based on graph databases. We keep the first two transformation rules mentioned above and we add two other rules which allow representing hierarchies in graph databases as follows:

Transformation 4: Each dimension $D_i(Name^{D_i}, A^{D_i}, H^{D_i})\in D^{MDM}$ is transformed into a set of dimension nodes defined by ($Name^{D_i{}^{MGD}}$, $A^{D_i{}^{MGD}}$, $H^{D_i{}^{MGD}}$) where:

- $Name^{D_i{}^{MGD}}$ is the name of the dimension D_i associated with the function λ^{MGD} as a label to the dimension nodes N_{D_i},
- $A^{D_i{}^{MGD}}$ is a set parameters and weak attributes of the dimension D_i. Each parameter is transformed into a set of nodes to allow representing hierarchies. Each weak attribute of a parameter is transformed into a property in parameter nodes.
- $H^{D_i{}^{MGD}}$ is a set of nodes representing hierarchies of the dimension D_i.

Transformation 5: Hierarchies ($Name^{H_j}$, $Param^{H_j}$, $Weak^{H_j}$) are transformed into a set of linked nodes ($Param^{H_j{}^{MGD}}$, $Weak^{H_j{}^{MGD}}$) where:

- $\text{Param}^{\text{H}_j^{\text{MGD}}}$ is a set of parameters nodes. The function λ^{MGD} associates to these parameters nodes the name of $\text{Param}^{\text{H}_j}$ as a label. For example, in Figure 2, the "Customer", "Household_Demographics" and "Income_Band" turns into separate nodes.
- $\text{Weak}^{\text{H}_j^{\text{MGD}}}$ is a set of properties associated to the parameters nodes using the function σ^{MGD}.
- An edge is defined between the fact nodes N_{F_i} and the lowest related parameter Param_k of each dimension using the function ρ^{MGD}.
- Edges are defined between the neighbouring parameters of the same hierarchy using the function ρ^{MGD}. For example, "Customer" and "Household_Demographics" are related using the relationship ":Current_HDemo", and "Household_Demographics" nodes are connected to "Income_Band" nodes through ":Has" edges.

In this case, the depth, which is the number of edges relating the fact node to parameter nodes, is greater than one.

IV. EXPERIMENTS

Our experiments have mostly three goals. The first one is to validate our approach by applying the proposed transformation rules to implement a star-like and snowflake-like data warehouses. The second goal is to compare the performance of the proposed graph data warehouses to analogous relational data warehouses implemented using the traditional R-OLAP approach: Intra-Model comparison. The third goal is to evaluate the effectiveness of the star and snowflake data designs in the context of graph warehouses to find out whether a snowflake-like graph data model would be less efficient than a star-like data model: Inter-Model comparison. Our comparison is made while taking into account the data model, data dimensionality and data size.

To achieve the above-mentioned goals, we use Neo4j (version 3.5.0), a graph database written in Java. It is queried through the cypher query language. We use Neo4j to write our transformation rules and implement the star-like and snowflake-like graph data warehouses. To compare these latter to relational data warehouses, we use MariaDB (version 10.1.38) as a relational database. These data warehouses were deployed under a virtual machine with 32 GB of RAM and 8TB disk. The virtual machine runs under the 64-bit Ubuntu-18.04.01 LTS operating system. No index was added, in any DBMS, because we assume filtering can concern all columns in an OLAP context, where the users make new queries regularly. The caches were cleared before each query in order to make sure the run time corresponds to the first time a query is asked. In an OLAP context, the users run new queries rather than repeating the same ones.

The evaluation between the graph data warehouses and star data warehouses is based mainly upon two criteria which are: write latency and read latency. These criteria have been chosen to decide objectively, which DBMS is more efficient when data get larger or queries get more complex.

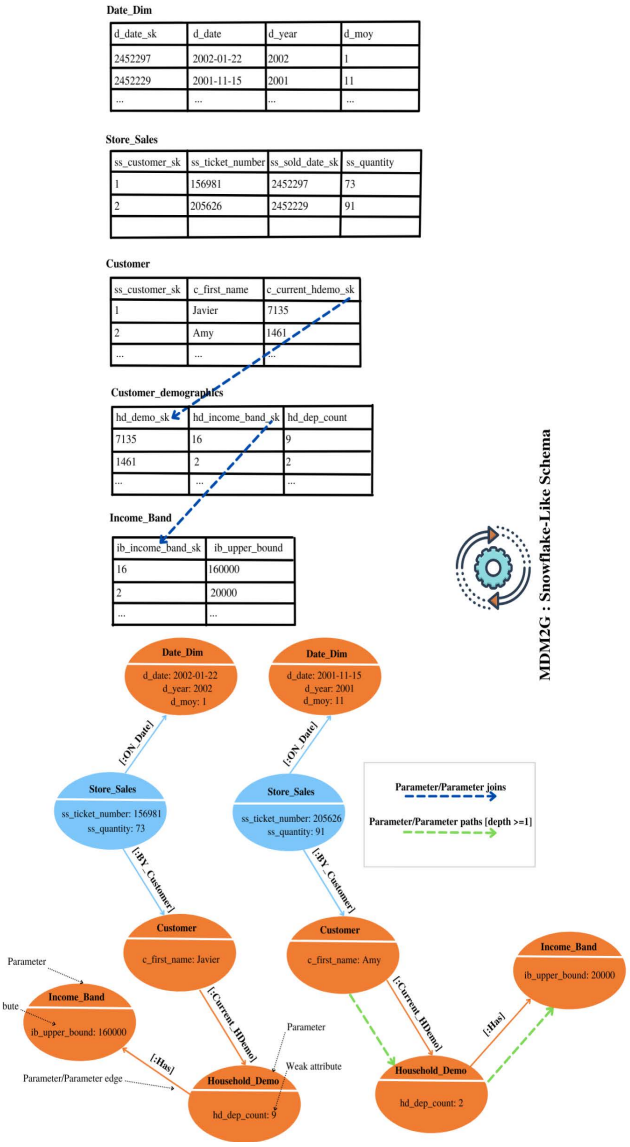


FIGURE 2. MDM2G: Snowflake-like schema.

A. DATA GENERATION

The data has been generated from the reference benchmark TPC-DS which has been proposed to evaluate the performance of DSS [62]. TPC-DS encompasses multiple snowflakes schemas that model the activities of a product supplier selling goods through three distribution channels: store, catalog, and internet [63]. TPC-DS data model is composed of 7 fact tables and 17 shared dimension tables. Each fact table has a snowflake schema. One distinguishing characteristic of the TPC-DS data model is the number of columns in each table. The average number of columns is 18 [63], which makes it possible to generate complex queries with predicates applied on many columns.

In this work, we focus on the most used snowflake schema [47], [64] which involves the fact table Store_Sales of the store channel and its 10 dimensions: date, time, store,

TABLE 1. Row counts per scale factor.

Tables	SF1	SF3	SF5	SF7
Store_Sales	2 880 404	8 639 377	14 400 052	20 159 325
Customer_	100 000	188 000	277 000	366 000
Customer_Demo	94 215	171 3197	249 626	322 762
Customer_Address	43 282	81 261	119 432	158 478
Date_Dim	73 049	73 049	73 049	73 049
Household_Demo	7 200	7 200	7 200	7 200
Income_Band	20	20	20	20
Item	18 000	36 000	54 000	74 000
Promotion	300	344	388	433
Store	12	32	52	72
Time_Dim	86 400	86 400	86 400	86 400

promotion, item, customer, customer demographics, household demographics, income band, and customer address. The TPC-DS data generator named DSDGEN generates for each entity (fact or dimension) a separate data file. These data files scale by means of scale factors (SF) that represent the data size in Gigabyte. In this work, we generated data according to four different scale factors SF1, SF3, SF5 and SF7 which are respectively 1GB, 3GB, 5GB, and 7GB. Whereas the fact table scales linearly with the scale factor, the non-static dimension tables scale sub-linearly. However, the data in static dimension tables such as date and time dimensions are loaded once and are not updated during the data maintenance phase [63]. Table 1 shows the number of rows generated for each table of the chosen snowflake schema.

B. DATA MODEL

As mentioned earlier, TPC-DS involves multiple snowflake schemas. In the chosen snowflake schema, data related to customers are hierarchically decomposed into different tables related with one-to-many relationships. In order to compare the performance of the snowflake design to the star design in the context of a graph data warehouse, we denormalized the dimension customer and its related tables using many left-joins to obtain a large table named "Customer_Details" which contains all the details about customers (customer demographics, household demographics, income band and customer address). In addition, we slightly modified the data model of TPC-DS to get pure snowflake and star designs as shown in Figure 3. More precisely, we deleted the columns that reference customer address, customer demographics and household demographics in the table Store_Sales. For example, we deleted the customer address at the time of sales transactions, and we keep only the current address. Also, we removed the columns that reference the date_dim dimension in the tables store, promotion, and customer. We dropped also the reference of Item in the table Promotion.

C. QUERIES

TPC-DS query generator QGEN allows generating queries according to different templates. In our experiments, we selected nine different queries that belong to the chosen snowflake (store sales channel). Theses nine queries can be grouped into three main categories as presented in Table 2.

TABLE 2. Characteristics of the query set.

Query type	Query name	Dimensions	Hierarchies	Returned rows
Non-hierarchical	Q3	2	no	61
	Q28	0	no	1
	Q42	2	no	10
	Q52	2	no	100
	Q55	2	no	69
Hierarchical	Q7	7	yes	100
	Q27	5	yes	100
Hierarchical and cumulative	Q13	6	yes	1
	Q48	5	yes	1

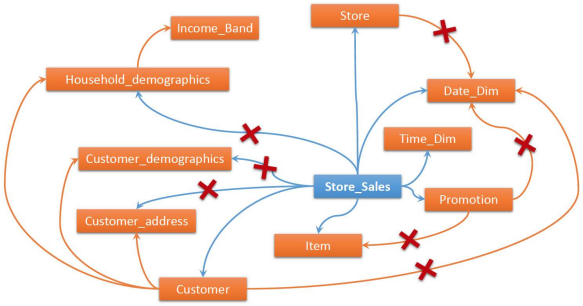


FIGURE 3. TPC-DS data model.

TABLE 3. Q3 - Non-hierarchical query.

SQL	Cypher
SELECT dt.d_year,	
item.i_brand_id brand_id,	**MATCH**(d:date_dim{d_moy:11})
item.i_brand brand,	WITH d
Sum(ss_ext_discount_amt) sum_agg	**MATCH**(d)<-[:ON_Date]-(ss:Store_Sales)
FROM date_dim dt,	WITH ss, d
Store_Sales ss,	**MATCH**(i:item{i_manufact_id:427})
item i	WITH i, ss, d
WHERE dt.d_date_sk = ss.ss_sold_date_sk	**MATCH**(ss)-[:OF_Item]->(i)
AND ss.ss_item_sk = i.i_item_sk	WITH i, ss, d
AND i.i_manufact_id = 427	**RETURN** d.d_year,
AND dt.d_moy = 11	i.i_brand_id AS brand_id,
GROUP BY dt.d_year,	i.i_brand AS brand,
i.i_brand,	**Sum**(ss.ss_ext_discount_amt) as sum_agg
ii_brand_id	**ORDER BY** d.d_year,
ORDER BY dt.d_year,	sum_agg **DESC**,
sum_agg **DESC**,	brand_id
brand_id	**LIMIT** 100;
LIMIT 100;	

The first category is made up of five *non-hierarchical queries* that do not involve hierarchies. Therefore, it is obvious that the customer table is not used in these queries. More precisely, all the tables queried are directly related to the fact table Store_Sales. For instance, the query Q3, cf. Table 3, computes the total rebate amount per item brand of the manufacturer 427 for all sales that took place in November. From a relational database perspective, this query requires different joins to get data from the tables item, Store_Sales and date_dim. However, in a graph database all these joins are replaced with relationships.

In the same category, we also distinguish the query Q28 which is not only non-hierarchical but also *flat* since only the fact table is used in this query and no dimension is queried. As shown in Table 4, Q28 calculates the average list price, the number of list prices and the number of distinct list prices of six different sales buckets of the store sales channel. Each bucket is defined by a range of distinct items and information about list price, coupon amount and wholesale cost.

TABLE 4. Q28 - Non-hierarchical query.

SQL	Cypher
SELECT * **FROM** (**SELECT Avg**(ss_list_price) B1_LP, **Count**(ss_list_price) B1_CNT, **Count**(**DISTINCT** ss_list_price) B1_CNTD **FROM** Store_Sales **WHERE** ss_quantity **BETWEEN** 0 **AND** 5 **AND** (ss_list_price **BETWEEN** 18 **AND** 18 + 10 **OR** ss_coupon_amt **BETWEEN** 1939 **AND** 1939 + 1000 **OR** ss_wholesale_cost **BETWEEN** 34 **AND** 34 + 20)) B1, (**SELECT Avg**(ss_list_price) B2_LP, **Count**(ss_list_price) B2_CNT, **Count**(**DISTINCT** ss_list_price) B2_CNTD **FROM** Store_Sales **WHERE** ss_quantity **BETWEEN** 6 **AND** 10 **AND** (ss_list_price **BETWEEN** 1 **AND** 1 + 10 **OR** ss_coupon_amt **BETWEEN** 35 **AND** 35 + 1000 **OR** ss_wholesale_cost **BETWEEN** 50 **AND** 50 + 20)) B2, (**SELECT Avg**(ss_list_price) B3_LP, **Count**(ss_list_price) B3_CNT, **Count**(**DISTINCT** ss_list_price) B3_CNTD **FROM** Store_Sales **WHERE** ss_quantity **BETWEEN** 11 **AND** 15 **AND** (ss_list_price **BETWEEN** 91 **AND** 91 + 10 **OR** ss_coupon_amt **BETWEEN** 1412 **AND** 1412 + 1000 **OR** ss_wholesale_cost **BETWEEN** 17 **AND** 17 + 20)) B3, (**SELECT Avg**(ss_list_price) B4_LP, **Count** (ss_list_price) B4_CNT, **Count**(DISTINCT ss_list_price) B4_CNTD FROM Store_Sales textbfWHERE ss_quantity **BETWEEN** 16 **AND** 20 **AND** (ss_list_price **BETWEEN** 9 **AND** 9 + 10 **OR** ss_coupon_amt **BETWEEN** 527 0 **AND** 5270 + 1000 **OR** ss_wholesale_cost **BETWEEN** 29 **AND** 29 + 20)) B4, (**SELECT Avg**(ss_list_price) B5_LP, **Count**(ss_list_price) B5_CNT, **Count**(DISTINCT ss_list_price) B5_CNTD **FROM** Store_Sales **WHERE** ss_quantity **BETWEEN** 21 **AND** 25 **AND** (ss_list_price **BETWEEN** 45 **AND** 45 + 10 **OR** ss_coupon_amt **BETWEEN** 826 **AND** 826 + 1000 **OR** ss_wholesale_cost **BETWEEN** 5 **AND** 5 + 20)) B5, (**SELECT Avg**(ss_list_price) B6_LP, **Count**(ss_list_price) B6_CNT, **Count**(**DISTINCT** ss_list_price) B6_CNTD **FROM** Store_Sales **WHERE** ss_quantity **BETWEEN** 26 **AND** 30 **AND** (ss_list_price **BETWEEN** 174 **AND** 174 + 10 **OR** ss_coupon_amt **BETWEEN** 5548 **AND** 5548 + 1000 **OR** ss_wholesale_cost **BETWEEN** 42 **AND** 42 + 20)) B6 **LIMIT** 100;	**MATCH**(ss:Store_Sales) **WHERE** ss.ss_quantity>=0 **AND** ss.ss_quantity<=5 **AND** ((ss.ss_list_price>=18 **AND** ss.ss_list_price<=18 + 10) **OR** (ss.ss_coupon_amt>=1939 **AND** ss.ss_coupon_amt<=1939 + 1000) **OR** (ss.ss_wholesale_cost>=34 **AND** ss.ss_wholesale_cost<=34 + 20)) WITH **Avg**(ss.ss_list_price) as B1_LP, **Count**(ss.ss_list_price) as B1_CNT, **Count**(**DISTINCT** ss.ss_list_price) as B1_CNTD **MATCH**(ss:Store_Sales) **WHERE** ss.ss_quantity>=6 **AND** ss.ss_quantity<=10 **AND** (ss.ss_list_price>=1 **AND** ss.ss_list_price<=1 + 10 **OR** ss.ss_coupon_amt>=35 **AND** ss.ss_coupon_amt<=35+1000 **OR** ss.ss_wholesale_cost>=50 **AND** ss.ss_wholesale_cost<= 50+20) WITH **Avg**(ss.ss_list_price) as B2_LP, **Count**(ss.ss_list_price) as B2_CNT, **Count**(**DISTINCT** ss.ss_list_price) AS B2_CNTD, B1_LP, B1_CNT, B1_CNTD **MATCH**(ss:Store_Sales) **WHERE** ss.ss_quantity>=11 **AND** ss.ss_quantity<=15 **AND** (ss.ss_list_price>=91 **AND** ss.ss_list_price<=91 + 10 **OR** ss.ss_coupon_amt>=1412 **AND** ss.ss_coupon_amt<=1412 + 1000 **OR** ss.ss_wholesale_cost>=17 **AND** ss.ss_wholesale_cost<= 17 +20) WITH **Avg**(ss.ss_list_price) as B3_LP, **Count**(ss.ss_list_price) AS B3_CNT, **Count**(**DISTINCT** ss.ss_list_price) AS B3_CNTD, B2_LP, B2_CNT, B2_CNTD, B1_LP, B1_CNT, B1_CNTD **MATCH**(ss:Store_Sales) **WHERE** ss.ss_quantity>=16 **AND** ss.ss_quantity<=20 **AND** (ss.ss_list_price>=9 **AND** ss.ss_list_pric<=9 +10 **OR** ss.ss_coupon_amt>=5270 **AND** ss.ss_coupon_amt<=5270 +1000 **OR** ss.ss_wholesale_cost>=29 **AND** ss.ss_wholesale_cost<=29+20) WITH **Avg**(ss.ss_list_price) as B4_LP, **Count** (ss.ss_list_price) as B4_CNT, **Count** (DISTINCT ss.ss_list_price) as B4_CNTD, B3_LP, B3_CNT, B3_CNTD, B2_LP, B2_CNT, B2_CNTD, B1_LP, B1_CNT, B1_CNTD **MATCH**(ss:Store_Sales) **WHERE** ss.ss_quantity>=21 **AND** ss.ss_quantity<=25 **AND** (ss.ss_list_price>=45 **AND** ss.ss_list_price<=45 +10 **OR** ss.ss_coupon_amt>=826 **AND** ss.ss_coupon_amt<=826 +1000 **OR** ss.ss_wholesale_cost>=5 **AND** ss.ss_wholesale_cost<=5+20) WITH **Avg**(ss.ss_list_price) as B5_LP, **Count** (ss.ss_list_price) as B5_CNT, **Count**(**DISTINCT** ss.ss_list_price) as B5_CNTD, B4_LP, B4_CNT, B4_CNTD, B3_LP, B3_CNT, B3_CNTD, B2_LP, B2_CNT, B2_CNTD, B1_LP, B1_CNT, B1_CNTD **MATCH**(ss:Store_Sales) **WHERE** ss.ss_quantity>=26 **AND** ss.ss_quantity<=30 **AND** ((ss.ss_list_price>=174 **AND** ss.ss_list_price<=174 + 10) **OR** (ss.ss_coupon_amt>=5548 **AND** ss.ss_coupon_amt<=5548 + 1000) **OR** (ss.ss_wholesale_cost>=42 **AND** ss.ss_wholesale_cost<=42 + 20)) WITH **Avg**(ss.ss_list_price) AS B6_LP, **Count** (ss.ss_list_price) AS B6_CNT, **Count**(**DISTINCT** ss.ss_list_price) as B6_CNTD, B5_LP, B5_CNT, B5_CNTD, B4_LP, B4_CNT, B4_CNTD, B3_LP, B3_CNT, B3_CNTD, B2_LP, B2_CNT, B2_CNTD, B1_LP, B1_CNT, B1_CNTD **RETURN** * **LIMIT** 100;

TABLE 5. Q7 - Hierarchical query.

SQL	Cypher
SELECT i_item_id, **Avg**(ss_quantity) agg1, **Avg**(ss_list_price) agg2, **Avg**(ss_coupon_amt) agg3, **Avg**(ss_sales_price) agg4 **FROM** Store_Sales, customer, customer_demographics, date_dim, item, promotion **WHERE** ss_sold_date_sk = d_date_sk **AND** ss_item_sk = i_item_sk **AND** ss_customer_sk= c_customer_sk **AND** c_current_cdemo_sk= cd_demo_sk **AND** ss_promo_sk = p_promo_sk **AND** cd_gender = 'F' **AND** cd_marital_status = 'W' **AND** cd_education_status = '2 yr Degree' **AND** (p_channel_email = 'N' **OR** p_channel_event = 'N') **AND** d_year = 1998 **GROUP BY** i_item_id **ORDER BY** i_item_id **LIMIT** 100;	**MATCH**(cd:customer_demographics) **WHERE** cd_gender='F' **AND** cd_marital_status='W' **AND** cd_education_statu='2 yr Degree' WITH cd **MATCH**(cd)<-[:Current_CDemo]-(c:customer) WITH cd, c **MATCH**(c)<-[:BY_Customer]-(ss) WITH ss **MATCH**(p:promotion)<-[:ON_Promo]-(ss) **WHERE** p.p_channel_email='N' **OR** p.p_channel_event='N' WITH ss **MATCH**(d:date_dimd_year:1998)<-[:ON_Date]-(ss) WITH ss **MATCH** (ss)-[:OF_Item]->(i:item) WITH ss, i **RETURN** i.i_item_id, **Avg**(ss.ss_quantity) as agg1, **Avg**(ss.ss_list_price) as agg2, **Avg**(ss.ss_coupon_amt) as agg3, **Avg**(ss.ss_sales_price) as agg4 **ORDER BY** i.i_item_id **LIMIT** 100;

The *non-hierarchical* category of queries is used to compare performance of graph versus relational model since their star and snowflakes variants are identical.

The second category consists of 2 *hierarchical queries* which are executed to answer complex questions. These queries go through more than five dimensions, up to a depth of 2, and include hierarchies and aggregates. For example, Q7, shown in Table 5, computes the average quantity, list price, discount, and sales price for promotional items sold in stores where the promotion is not offered by mail or a special event. The results are restricted to a specific gender, marital and educational status.

The third category contains *hierarchical and cumulative queries* that are not only highly complex but also cumulative.

TABLE 6. Q13 - Hierarchical and cumulative query.

SQL	Cypher
SELECT Avg(ss_quantity), **Avg**(ss_ext_sales_price), **Avg**(ss_ext_wholesale_cost), **Sum**(ss_ext_wholesale_cost) **FROM** Store_Sales, store, customer_demographics, customer, household_demographics, customer_address, date_dim **WHERE** s_store_sk = ss_store_sk **AND** ss_sold_date_sk = d_date_sk **AND** d_year = 2001 **AND** ((ss_customer_sk = c_customer_sk **AND** c_current_cdemo_sk = cd_demo_sk **AND** c_current_hdemo_sk = hd_demo_sk **AND** cd_marital_status = 'U' **AND** cd_education_status = 'Advanced Degree' **AND** ss_sales_price **BETWEEN** 100.00 **AND** 150.00 **AND** hd_dep_count = 3) **OR** (c_current_hdemo_sk = hd_demo_sk **AND** ss_customer_sk = c_customer_sk **AND** c_current_cdemo_sk = cd_demo_sk **AND** cd_marital_status = 'M' **AND** cd_education_status = 'Primary' **AND** ss_sales_price BETWEEN 50.00 AND 100.00 **AND** hd_dep_count = 1) **OR** (ss_customer_sk = c_customer_sk **AND** c_current_hdemo_sk = hd_demo_sk **AND** c_current_cdemo_sk = cd_demo_sk **AND** cd_marital_status = 'D' **AND** cd_education_status = 'Secondary' **AND** ss_sales_price **BETWEEN** 150.00 **AND** 200.00 AND hd_dep_count = 1)) **AND** ((ca_address_sk = c_current_add_sk **AND** ss_customer_sk = c_customer_sk **AND** ca_country = 'United States' **AND** ca_state **IN** ('AZ','NE','IA') **AND** ss_net_profit BETWEEN 100 **AND** 200) **OR** (ss_customer_sk = c_customer_sk **AND** c_current_addr_sk = ca_address_sk **AND** ca_country = 'United States' **AND** ca_state **IN** ('MS','CA','NV') **AND** ss_net_profit **BETWEEN** 150 **AND** 300) **OR** (ca_address_sk = c_current_addr_sk **AND** ss_customer_sk = c_customer_sk **AND** ca_country = 'United States' **AND** ca_state **IN** ('GA','TX','NJ') **AND** ss_net_profit **BETWEEN** 50 **AND** 250));	**MATCH**(d:date_dim{d_year:2001}) WITH d **MATCH**(d)<-[:ON_Date]-(ss:Store_Sales) WITH ss **MATCH**(s:store)<-[:AT_Store]-(ss:Store_Sales) WITH ss **MATCH**(ss:Store_Sales)-[:BY_Customer]->(c) WITH ss, c **MATCH**(c)-[:Current_Addr]->(ca:customer_address) WITH ss, c, ca **MATCH** (c)-[:Current_HDemo]->(hd:household_demographics) WITH ss, ca, c, hd **MATCH** (c)-[:Current_CDemo]->(cd:customer_demographics) WITH ss, ca, c, hd, cd **WHERE** ((cd.cd_marita_status = 'U' **AND** cd.cd_education_status = 'Advanced Degree' **AND** ss.ss_sales_price>=100.00 **AND** ss.ss_sales_price<=150.00 **AND** hd.hd_dep_count = 3) **OR** (cd.cd_marita_status = 'M' **AND** cd.cd_education_status = 'Primary' **AND** ss.ss_sales_price>=50.00 **AND** ss.ss_sales_price<=100.00 **AND** hd.hd_dep_count = 1) OR (cd.cd_marita_status = 'D' **AND** cd.cd_education_status = 'Secondary' **AND** ss.ss_sales_price>= 150.00 **AND** ss.ss_sales_price<=200.00 **AND** hd.hd_dep_count = 1)) **AND** ((ca.ca_country = 'United States' **AND** ca.ca_state **IN** ['AZ','NE','IA'] **AND** ss.ss_net_profit>=100 **AND** ss.ss_net_profit<=200) **OR** (ca.ca_country ='United States' **AND** ca.ca_state **IN** ['MS','CA','NV'] **AND** ss.ss_net_profit>=150 **AND** ss.ss_net_profit<=300) **OR** (ca.ca_country = 'United States' **AND** ca.ca_state **IN** ['GA','TX','NJ'] **AND** ss.ss_net_profit>=50 **AND** ss.ss_net_profit<=250)) **RETURN Avg**(ss.ss_quantity), **Avg**(ss.ss_ext_sales_price), **Avg**(ss.ss_ext_wholesale_cost), **Sum**(ss.ss_ext_wholesale_cost);

Those queries return a single row aggregating all selected rows or nodes. For example, Q13, shown in Table 6, calculates the average sales quantity, the average sales price, the average wholesale cost ant the total wholesale cost for store sales of different customer types including their household demographics, sales price and different combinations of states and sales profit for a given year.

The purpose of the following experiments is to demonstrate that we can implement a graph data warehouse using our approach and apply a variety of queries on it. We evaluate the performance of each approach based on the execution time of the set of TPC-DS queries that we defined previously. We report the execution time for queries adapted for star schema and snowflake schema. Note that since Neo4j has its own query language, the queries are translated into the query language Cypher.

V. RESULTS

In this section, we report the performance evaluation results of the relational and graph data warehouses based on two metrics: write latency and read latency.

A. WRITE LATENCY

In graph databases, relationships between nodes are considered the first-class citizen [36]. While relational databases rely on joins to answer complex queries, graph databases store physically links between nodes. Consequently, writing data on graph data warehouses is significantly longer than relational data warehouses due to the time required to create

relationships between nodes. This has been checked on the experiments that we conducted. As shown in Figure 4, the loading time of the relational data warehouse is up to fourteen times faster than the graph data warehouse. In addition, unlike relational databases where the time of creation of the star and snowflake data warehouses is the same, the creation of a graph data warehouse with a snowflake-like schema takes more time than a graph data warehouse with a star-like schema. Indeed, the snowflake-like graph data warehouse requires the creation of more relationships, which requires more time.

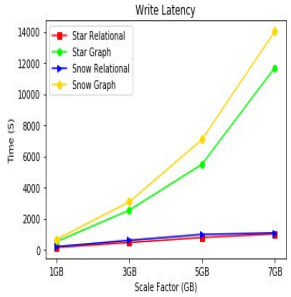


FIGURE 4. **Write latency.**

B. READ LATENCY

We distinguish the three types of queries detailed in Table 2.

1) NON-HIERARCHICAL QUERIES

The run times of non-hierarchical queries Q3, Q28, Q42, Q52 and Q55 are shown respectively in Figures 5a, 5b, 5c, 5d and 5e. Queries Q3, Q42, Q52 and Q55 are very similar. Query Q3 does not make a selection on the year, contrary to the other queries, hence its execution time is higher. However, the growth of the execution time as a function of the amount of data to process remains similar. Query Q28 has a different structure. It requires reading the table of facts several times, keeping a large amount of information in memory, hence the longer execution time. As previously mentioned in Section IV-C, these queries do not involve hierarchies. More precisely, the dimension customer and all its related hierarchies are not present in these queries. Further, the query is written in the same way for star and snowflake schemas. Consequently, the response time of the normalized and denormalized data warehouses is the same. Thus, the curves of normalized and denormalized data warehouses are superposed. These experiments show that for these queries which are not complex, not requiring to link a large number of different data which would require several joins in the relational model, relational databases are more efficient than graph databases (up to ten times faster).

2) HIERARCHICAL QUERIES

The run times of queries Q7 and Q27 are shown respectively in Figures 6a and 6b. Queries Q7 and Q27 are similar. They are hierarchical queries, with a maximum depth

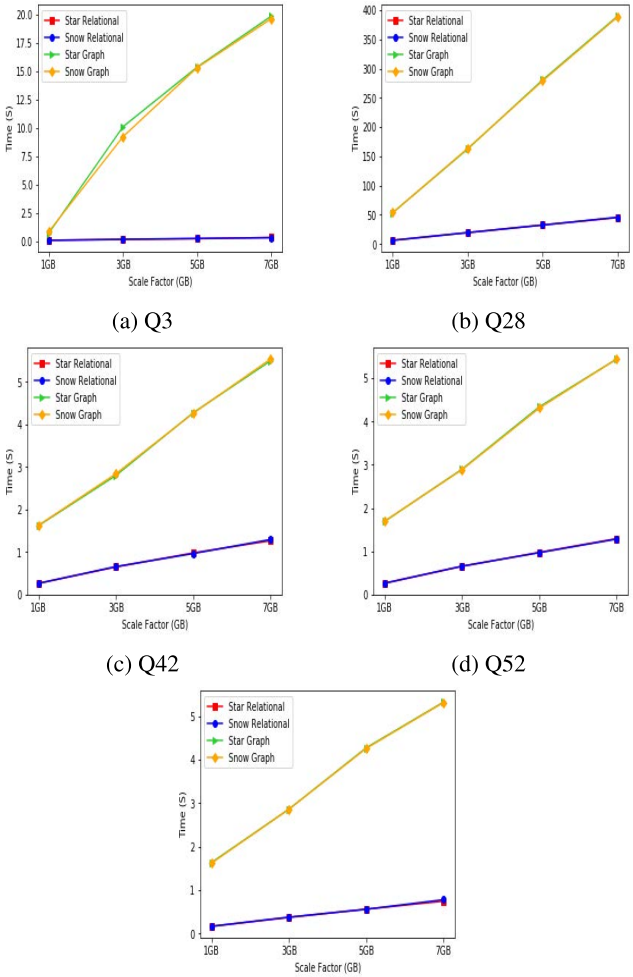
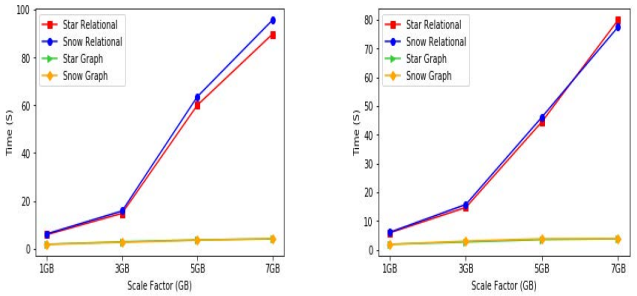


FIGURE 5. **Read latency - non-hierarchical queries.**

of 2, and include the computation of aggregates and a sorting of the results. The execution times are therefore close. For the second type of queries, our results show that the snowflake schema is more time consuming than the star schema in the case of relational data warehouses. However, for graph databases, the curves of star and snowflake graph data warehouses are superposed. Surprisingly, they have the same performance. Additionally, the graph data warehouse is significantly more efficient than the relational data warehouse (up to more than twenty times faster). Also, when the data size increases, the graph data warehouse becomes more and more efficient than the relational one.

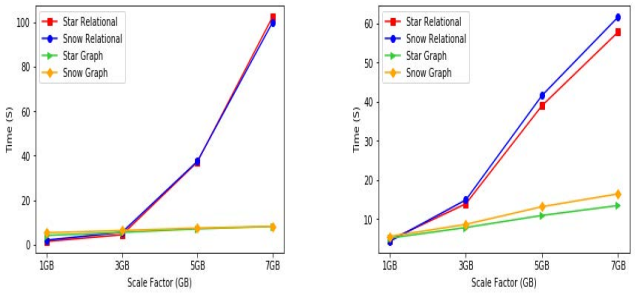
3) HIERARCHICAL AND CUMULATIVE QUERIES

The run times of queries Q13 and Q48 are shown in 7a and 7b respectively. Both queries compute a few aggregates (average and sum) on sales for one year and for conditions on various sub-dimensions of customers: customer_demographics, household_demographics, customer_address.



(a) Q7 (b) Q27

FIGURE 6. Read latency - hierarchical queries.



(a) Q13 (b) Q48

FIGURE 7. Read latency - hierarchical and cumulative queries.

For this category, the performance of relational and graph data warehouses is similar when the data size is small. However, for larger data sizes, graph data warehouses answer queries faster than the relational ones (up to ten times faster).

C. DISCUSSION

The main drawback of the snowflake schema in a relational implementation (R-OLAP) is that the additional levels of depth for the dimensions require longer traversals to access the information in the deeper dimensions. These traversals result in joins in the relational model, and increase the execution time compared to the star schema. Reciprocally a snowflake schema allows a more efficient storage than a star schema, and the consideration of various sources, which will add up to deeper dimensions. Our experiments on non-hierarchical queries show that a relational implementation of a star schema is more efficient than a graph implementation. However, in the case of a snowflake schema, the increase in the length of the paths in the queries has relatively little impact on the performance of the graph-based data warehouse. Graph-oriented data warehouses have been shown to be effective for the second and third types of queries, those involving many dimensions or hierarchies. Indeed, in graph-based data warehouses, the relationships between the fact and the dimension, and between the attributes of the same dimension (hierarchies) are physically implemented. Thus, in the case of an increase in the volume to be processed, when queries go deeper in the dimensions, graph-based data warehouses are more efficient, and more robust to the increase in complexity. This is because while relational databases crawl all tables until data matching the search criteria is found, graph databases crawl only those nodes that meet the criteria. Therefore, both star and snowflake graphical data warehouses are effective depending on the use case (normalized or non-normalized data). The results show that it is possible to consider the snowflake scheme for graph-based data warehouses to easily add additional data connected to dimensions without significant impact on query response time.

VI. CONCLUSION

This paper investigates the design, implementation, and evaluation of graph data warehouses. The goal of this study is to determine whether a traditional relational data warehouse or a graph data warehouse would be more effective. We have proposed a set of transformation rules called MDM2G to convert a multidimensional model of a data warehouse into a graph database. These rules transform a multidimensional model into a graph database using two schemas: star-like and snow-like schemas. Experiments are conducted using data generated from the TPC-DS benchmark. We generated respectively data sets of size 1GB, 3GB, 5GB and 7GB. The experimental setup shows the way OLAP systems can be implemented with graph databases using Neo4j. This process includes data transformation, data loading and performing complex analytical queries. The entire process allows us to compare the different approaches with each other. We also compare the performance of graph data warehouses to similar relational data warehouses. Results show that both of our proposed graph data warehouses perform well, with denormalized schema being hardly more efficient for some queries. The results of the experiments exhibit the advantage of the use of graph NoSQL technologies for implementing OLAP systems and answer complex questions. In this work, our evaluation has been based on objective measures. However, other subjective measures could be used such as the maturity, ease of programming, security and flexibility. In our further research works we will focus on the use of graph databases to store and analyze biological networks in order to provide fast answers for complex queries and predict hidden relationships between proteins.

DISCLOSURE STATEMENT

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

DATA DEPOSITION

The data that support the findings of this study are openly available in github at https://github.com/hakidsdc/SNOW-STAR-TPCDS.

REFERENCES

[1] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *Int. J. Inf. Manage.*, vol. 35, no. 2, pp. 137–144, Apr. 2015.

[2] W. H. Inmon, *Building the Data Warehouse*. Hoboken, NJ, USA: Wiley, 1996.
[3] R. Kimball, *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. Hoboken, NJ, USA: Wiley, 1996.
[4] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *ACM SIGMOD Rec.*, vol. 26, no. 1, pp. 65–74, Mar. 1997.
[5] O. Mangisengi and A. M. Tjoa, "A multidimensional modeling approach for OLAP within the framework of the relational model based on quotient relations," in *Proc. ACM Int. 1st Work. Data. War. OLAP. (DOLAP)*, Nov. 1998, pp. 40–46.
[6] B. Dinter, C. Sapia, G. Höfling, and M. Blaschka, "The OLAP market: State of the art and research issues," in *Proc. 1st ACM Int. Workshop Data Warehousing OLAP (DOLAP)*, Nov. 1998, pp. 22–27.
[7] P. Lehmann and J. Jaszewski, "Business terms as a critical success factor for data warehousing," in *Proc. Int. 1st Workshop Design. Manag. Data Warehouse (DMDW)*, Jun. 1999, pp. 22–27.
[8] D. L. Moody and M. A. Kortink, "From enterprise models to dimensional models: A methodology for data warehouse and data mart design," in *Proc. Int. 2nd Workshop Design Man. Data Warehouse (DMDW)*, Jun. 2000, p. 5.
[9] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: A data provenance perspective," in *Proc. 48th Annu. Southeast Regional Conf. (ACM SE)*, Apr. 2010, pp. 1–6.
[10] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Record.*, vol. 39, no. 4, pp. 12–27, 2010.
[11] A. K. Zaki, "NoSQL databases: New millennium database for big data, big users, cloud computing and its security challenges," *Int. J. Res. Eng. Technol.*, vol. 3, no. 15, pp. 403–409, May 2014.
[12] A. Schram and K. M. Anderson, "MySQL to NoSQL: Data modeling challenges in supporting scalability," in *Proc. 3rd Annu. Conf. Syst., Program., Appl., Softw. Hum. (SPLASH)*, Oct. 2012, pp. 191–202.
[13] A. Haseeb and G. Pattun, "A review on NoSQL: Applications and challenges," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 1, pp. 203–207, Jan. 2017.
[14] R. Hecht and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in *Proc. IEEE Int. 1st Conf. Cloud. Serv. Comput. (CSC)*, Dec. 2011, pp. 336–341.
[15] B. G. Tudorica and C. Bucur, "A comparison between several NoSQL databases with comments and notes," in *Proc. IEEE Int. 10th Conf. Netw. Educ. Res. (RoEduNet)*, Jun. 2011, pp. 1–5.
[16] P. Atzeni, F. Bugiotti, L. Cabibbo, and R. Torlone, "Data modeling in the NoSQL world," *Comput. Standards Interfaces*, vol. 67, Jan. 2020, Art. no. 103149.
[17] Z. Bicevska and I. Oditis, "Towards NoSQL-based data warehouse solutions," *Proc. Comput. Sci.*, vol. 104, pp. 104–111, Dec. 2017.
[18] M. Golfarelli and S. Rizzi, "From star schemas to big data: 20 + years of data warehouse research," in *A Comprehensive Guide Through Italian Database Res. Over Last 25 Years*. Springer, 2018, pp. 93–107.
[19] K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Using the column oriented NoSQL model for implementing big data warehouses," in *Proc. Int. Conf. Parallel. Distrib. Process. Tech. App. (PDPTA)*, Jul. 2015, pp. 469–475.
[20] M. Chevalier, M. El Malki, O. Teste, and R. Tournier, "Implementing multidimensional data warehouses into NoSQL," in *Proc. Int. 17th Conf. Entr. Inf. Sys. (ICEIS)*, Apr. 2015, pp. 172–183.
[21] L. C. Scabora, J. J. Brito, R. R. Ciferri, and C. D. D. A. Ciferri, "Physical data warehouse design on NoSQL databases," in *Proc. Int. 18th Conf. Entr. Inf. Sys. (ICEIS)*, Apr. 2016, pp. 111–118.
[22] M. Y. Santos and C. Costa, "Data models in NoSQL databases for big data contexts," in *Proc. Int. 1st Conf. Data Mining Big Data (DMBD)*, Jun. 2016, pp. 475–485.
[23] M. Y. Santos, B. Martinho, and C. Costa, "Towards NoSQL-based data warehouse solutions," *J. Manag. Anal.*, vol. 4, no. 2, pp. 111–129, Mar. 2017.
[24] M. Boussahoua, O. Boussaid, and F. Bentayeb, "Logical schema for data warehouse on column-oriented NoSQL databases," in *Proc. Int. 28th Conf. Database Expert Sys. Appl. (DEXA)*, Aug. 2017, pp. 247–256.
[25] K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Towards an OLAP environment for column-oriented data warehouses," in *Proc. Int. 16th Conf. Data. Warehouse knowl. Disc. (DaWaK)*, Sep. 2014, pp. 221–232.
[26] I. B. Messaoud, A. A. Alshdadi, and J. Feki, "Building a document-oriented warehouse using NoSQL," *Int. J. Oper. Res. Inf. Sys.*, vol. 12, no. 2, pp. 33–54, Apr. 2021.
[27] M. Chevalier, M. El Malki, O. Teste, and R. Tournier, "Document-oriented data warehouses: Models and extended cuboids, extended cuboids in oriented document," in *Proc. IEEE Int. 10th Conf. Res. Challenge Inf. Sci. (RCIS)*, Jun. 2016, pp. 1–11.
[28] R. A. S. N. Soransso and M. C. Cavalcanti, "Data modeling for analytical queries on document-oriented DBMS," in *Proc. ACM 33th Annu. Symp. Appl. Comput (SAC)*, Apr. 2018, pp. 541–548.
[29] E. Gallinucci, M. Golfarelli, and S. Rizzi, "Approximate OLAP of document-oriented databases: A variety-aware approach," *Inf. Sys.*, vol. 85, pp. 114–130, Nov. 2019.
[30] Y. Cheng, P. Ding, T. Wang, W. Lu, and X. Du, "Which category is better: Benchmarking relational and graph database management systems," *Data. Sci. Eng.*, vol. 4, no. 4, pp. 309–322, Nov. 2019.
[31] M. Macak, M. Stovcik, and B. Buhnova, "The suitability of graph databases for big data analysis: A benchmark," in *Proc. Int. 5th Conf. Internet Things Big Data Secur. (IoTBDS)*, May 2020, pp. 213–220.
[32] A. Castelltort and A. Laurent, "NoSQL graph-based OLAP analysis," in *Proc. Int. 12th Conf. Knowl. Disc. Inf. Retr. (KDIR)*, Oct. 2014, pp. 217–224.
[33] A. Sellami, A. Nabli, and F. Gargouri, "Transformation of data warehouse schema to NoSQL graph data base," in *Proc. Int. 18th Conf. Intell. Sys. Design Appl. (ISDA)*, Dec. 2018, pp. 410–420.
[34] A. Sellami, A. Nabli, and F. Gargouri, "Graph NoSQL data warehouse creation," in *Proc. 22nd Int. Conf. Inf. Integr. Web-based Appl. Services*, Nov. 2020, pp. 34–38.
[35] L. Gómez, B. Kuijpers, and A. Vaisman, "Online analytical processsing on graph data," *Intell. Data Anal.*, vol. 24, no. 3, pp. 515–541, May 2020.
[36] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, 2015.
[37] B. Chuck, D. Herreman, D. Schau, and R. Bell, *Data Modeling Techniques for Data Warehousing*. IBM Corp., 1998.
[38] C. J. M. Tauro, S. Aravindh, and A. B. Shreeharsha, "Comparative study of the new generation, agile, scalable, high performance NOSQL databases," *Int. J. Comput. Appl.*, vol. 48, no. 20, pp. 1–4, Jun. 2012.
[39] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process.*, Aug. 2013, pp. 15–19.
[40] D. Serrano, D. Han, and E. Stroulia, "From relations to multi-dimensional maps: Towards an SQL-to-HBase transformation methodology," in *Proc. IEEE Int. 8th Conf. Cloud Comput. (CLOUD)*, Jun. 2015, pp. 81–89.
[41] G. Karnitis and G. Arnicans, "Migration of relational database to document-oriented database: Structure denormalization and data transformation," in *Proc. IEEE Int. 7th Conf. Comput. Intell. Commun. Syst. Netw. (CICSyN)*, Jun. 2015, pp. 113–118.
[42] T. Jia, X. Zhao, Z. Wang, D. Gong, and G. Ding, "Model transformation and data migration from relational database to MongoDB," in *Proc. IEEE Int. Congr. Big Data (BigData Congress)*, Jun. 2016, pp. 60–67.
[43] M. L. Chouder, S. Rizzi, and R. Chalal, "EXODuS: Exploratory OLAP over document stores," *Inf. Syst.*, vol. 79, pp. 44–57, Jan. 2019.
[44] R. De Virgilio, A. Maccioni, and R. Torlone, "R2G: A tool for migrating relations to graphs," in *Proc. Int. 17th Conf. Extend Database Technol. (EDBT)*, Mar. 2014, pp. 640–643.
[45] S. Lee, B. H. Park, S.-H. Lim, and M. Shankar, "Table2Graph: A scalable graph construction from relational tables using map-reduce," in *Proc. IEEE 1st Int. Conf. Big Data Comput. Service Appl.*, Mar. 2015, pp. 294–301.
[46] K. Dehdouh, O. Boussaid, and F. Bentayeb, "Big data warehouse: Building columnar NoSQL OLAP cubes," *Int. J. Decis. Support Syst. Technol.*, vol. 12, no. 1, pp. 1–24, Jan. 2020.
[47] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, "Implementation of multidimensional databases with document-oriented NoSQL," in *Proc. Int. 17th Conf. Big Data Anal. Knowl. Discovery (DaWak)*, Sep. 2015, pp. 379–390.
[48] P. Gomez, R. Casallas, and C. Roncancio, "Data schema does matter, even in NoSQL systems!" in *Proc. IEEE 10th Int. Conf. Res. Challenges Inf. Sci. (RCIS)*, Jun. 2016, pp. 1–6.
[49] F. Atigui, F. Ravat, O. Teste, and G. Zurfluh, "Using OCL for automatically producing multidimensional models and ETL processes," in *Proc. Int. 14th Conf. Data Warehousing Knowl. Discovery (DaWak)*, Sep. 2012, pp. 42–53.
[50] F. Ravat, O. Teste, R. Tournier, and G. Zurfluh, "Algebraic and graphic languages for OLAP manipulations," *Int. J. Data Warehousing Mining*, vol. 4, no. 1, pp. 17–46, Jan. 2008.

[51] R. Angles and C. Gutírrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1–39, Feb. 2008.
[52] H. Akid and M. B. Ayed, "Towards NoSQL graph data warehouse for big social data analysis," in *Proc. Int. 16th Conf. Intell. Syst. Design. Appl. (ISDA)*, Dec. 2016, pp. 965–973.
[53] S. A. Moosavi, M. Jalali, N. Misaghian, S. Shamshirband, and M. H. Anisi, "Community detection in social networks using user frequent pattern mining," *Knowl. Inf. Syst.*, vol. 51, pp. 159–186, Apr. 2017.
[54] D. S. Himmelstein, A. Lizee, C. Hessler, L. Brueggeman, S. L. Chen, D. Hadley, A. Green, P. Khankhanian, and S. E. Baranzini, "Systematic integration of biomedical knowledge prioritizes drugs for repurposing," *eLife*, vol. 6, p. e26726, Sep. 2017.
[55] A. Fabregat, F. Korninger, G. Viteri, K. Sidiropoulos, P. Marin-Garcia, P. Ping, G. Wu, L. Stein, P. D'Eustachio, and H. Hermjakob, "Reactome graph database: Efficient access to complex pathway data," *PLOS Comput. Biol.*, vol. 14, no. 1, Jan. 2018, Art. no. e1005968.
[56] S. Timón-Reina, M. Rincón, and R. Martínez-Tomás, "An overview of graph databases and their applications in the biomedical domain," *Database*, vol. 2021, pp. 1–22, May 2021.
[57] M. Mezzanzanica, F. Mercorio, M. Cesarini, V. Moscato, and A. Picariello, "GraphDBLP: A system for analysing networks of computer scientists through graph databases," *Multimedia Tools Appl.*, vol. 77, no. 14, pp. 18657–18688, Jan. 2018.
[58] F. Mercorio, M. Mezzanzanica, V. Moscato, A. Picariello, and G. Sperlì, "A tool for researchers: Querying big scholarly data through graph databases," in *Proc. Eur. Conf. (PKDD/ECML)*, Sep. 2019, pp. 760–763.
[59] R. H. Güting, "GraphDB: Modeling and querying graphs in databases," in *Proc. Int. 20th Conf. Very Large. Databases (VLDB)*, Sep. 1994, pp. 12–15.
[60] S. Batra and C. Tyagi, "Comparative analysis of relational and graph databases," *Int. J. Soft. Comupt. Eng.*, vol. 2, no. 2, pp. 509–512, May 2012.
[61] R. Angles, "The property graph database model," in *Proc. Int. 12th Workshop Foundations Data Manag. (AMW)*, May 2018, pp. 1–10.
[62] M. Poess, R. O. Nambiar, and D. Walrath, "Why you should run TPC-DS: A workload analysis," in *Proc. Int. 33th Conf. Very Large Databases (VLDB)*, Sep. 2007, pp. 1138–1149.
[63] R. O. Nambiar and M. Poess, "The making of TPC-DS," in *Proc. Int. 32th Conf. Very Large Databases (VLDB)*, Sep. 2006, pp. 1049–1058.
[64] W. Qu and S. Dessloch, "Distributed snapshot maintenance in wide-column NoSQL databases using partitioned incremental ETL pipelines," *Inf. Syst.*, vol. 70, pp. 48–58, Oct. 2017.

HAJER AKID (Member, IEEE) is currently pursuing the joint Ph.D. degree in computer systems engineering with the University of Strasbourg, France, and the University of Sfax, Tunisia. She has been joining the REGIM-Laboratory, since 2016, and the ICube Laboratory, since 2018. While for technical affiliation, she has been a member of Largest Technical Organization in the world, since 2016. More precisely, she is a member of Big Data, Social Networking and Smart Cities IEEE Communities. Her research interests include decision support systems, NoSQL databases, machine learning, and big data.

GABRIEL FREY has been an Associate Professor with the University of Strasbourg, since 2006. He is a member of the ICube Laboratory, Data Science and Knowledge Research Team. His research interests include data mining and machine learning, optimization methods, and inverse problems, mainly applied to bioinformatics and medical imaging.

MOUNIR BEN AYED (Senior Member, IEEE) received the Ph.D. degree in biomedical engineering from the Paris 12 Val de Marne University, in 1989, and the Habilitation degree in computer system engineering from the National Engineering School (ENIS), University of Sfax, Tunisia, in 2013. He is currently a Professor with the Computer Sciences and Communication Department, Faculty of Sciences of Sfax. He currently teaches software engineering, data base management systems, decision support systems, and data warehouse. He is a member of the REGIM-Laboratory. Most of his research are designed and evaluated in the medical field. He was the chair, an organization committee member, and a technical committee member of many national and international conferences. Among the numerous conferences, he organized the Engineering Sciences for Biology and Medicine, ESBM'2013 and ESBM'2015. He created the IEEE-EMBS Tunisia Chapter, in 2009. He was the Chair of this Chapter, from 2009 to 2012, the Vice Chair, from 2013 to 2014, and the Chair of the Second Time, from 2015 to 2016.

NICOLAS LACHICHE has been an Associate Professor with the University of Strasbourg, since 1999, and the Head of the ICube Laboratory, Data Science and Knowledge Research Group, since 2012. Beforehand, he did his Ph.D. Research in Nancy and was a Research Associate for two years with the University of Bristol, U.K. His research interests include data mining and machine learning with a focus on handling complex data and problems in various domains, such as chemistry, environment, health, and industry 4.0.