Query Management for a Decentralised Enterprise Knowledge Graph

Bastien Vidé, Max Chevalier, Franck Ravat

Query Management for a Decentralised Enterprise Knowledge Graph

Bastien Vidé*†, Max Chevalier*, Franck Ravat*
* *Institut de Recherche en Informatique de Toulouse*
Toulouse, France
first.last@irit.fr
† *Umlaut*
Blagnac, France
first.last@umlaut.com

***Abstract*—Organisations manage large amount of data scattered in multiple data sources disseminated within the organisation. In this context, decision-making having a complete vision of the available data to make the right decisions is a challenge. Choosing the right strategy to bypass this issue may be impactful on the enterprise data management policy. A good alternative to limit data integration issues can be found in Knowledge Graphs. These Knowledge Graphs allow decision makers to understand which data of interest are existing in the enterprise. They are integrating the semantic of data rather than data itself. In a previous research, we proposed a decentralised enterprise knowledge graph (DEKG) architecture to facilitate decision making and consider the scattered data sources of interest. Due to the decentralised dimension of this architecture and the separation of the data that is scattered in remote sources, but also to the "centralised" aspect of knowledge graphs, one of the remaining challenges is the decentralised query management in order to answer a decision-maker needs. As a result we define in this paper a four step approach to manage queries and to build results based on a DEKG. We illustrate the proposed approach through examples and detail its implementation. We also show the results of preliminary experiments before giving some future work.**

***Index Terms*—Knowledge representation, Distributed database, Query processing**

1. Introduction

Knowledge Graph (KG) initially originates from the Semantic Web domain in order to organise Knowledge contained in different web pages, making Internet data machine-readable. The Enterprise Knowledge Graph is an application of KG to the organisations [4], [3] that particularly highlights the relationships existing between the data available within the company. These relationships mostly show semantic relationships between data. Thanks to the potential it gives, the Enterprise Knowledge Graph is viewed as a good alternative solution that overcomes the "data silos problem" [6]. The EKG intends to highlight relationships existing in data by ingesting as much data as possible of an organisation into a centralised storage [2]. The end-users then have an easy access to the whole organisation data thanks to this EKG. Without such an EKG, the end-user need to dig into different data sources and manually search for the correspondence between those different sources to understand how and what to query.

In previous work, we proposed a variation of EKG, called Decentralised Enterprise Knowledge Graph (DEKG) [11] which is a particular kind of Enterprise Knowledge Graph that offers a unified view of the organisation data without copying original data into a single centralised system, which simplifies the access to the organisation data. In order to have a good understanding of the available data, the DEKG generates a Business View (BV) which is a particular view that combines and simplifies all the underlying sources schemata.

Beyond the visualisation, the BV also allows the end-user to query data to answer decision-maker needs. Since data sources are scattered, the challenge here is to query the underlying sources in a decentralised way from the BV presented earlier. As a conclusion, such business view allows end-users to focus on the business perspective while ignoring the technical aspects that is managed by the proposed architecture.

This paper tackles the query management challenge in a DEKG context. To answer this issue, we first underline in Section 2 work related to EKG querying and distributed databases query management. In Section 3, we then propose an original 4-steps approach for query management in the context of a DEKG. We lastly demonstrate its feasibility in different conditions through an experimental implementation using three different data sets with multiple sources type in Section 4.

2. Related work

2.1. Query management

Pan et al. demonstrated the usefulness of an EKG [8] but also explained how to query them and introduced Query Answering in Knowledge Graphs [7] which helps non-technical

978-1-6654-6495-6/22/$31.00 ©2022 IEEE
DOI 10.1109/SITIS57111.2022.00012

users to query an EKG. They present Graph Exploration approach which translate the user natural language queries to navigate the graph, and Machine Learning approach which trains from the EKG itself and question/answers pairs. In the same way, Song et al. also showed how to both build and query an Enterprise Knowledge Graph [10] with question answering aimed at non-technical end-users. Using feature-based context-free grammar, they translate a user question into a SPARQL query. Furthermore, they included auto-completion to help the end-user to formulate their question, which can lack familiarity with the organisation data. These approaches show how important the EKG is for (non technical) end-users. However, they do not consider multiple data sources and assume the data is stored in a centralised EKG.

In a decentralized (or distributed) context we can underline work related to query management for Distributed Databases. Özsu et al. introduced the basics of Distributed Database, including all the principles of Query Processing in Federated systems [13]. They defined the query decomposition and data localization, and also how they both solve queries in a distributed environment. Czejdo et al. also worked on distributed queries of independent databases [1] to answer multi-database queries using a relational model. In Distributed Databases, the approaches are mainly working with relational databases. Even though the local-as-view (bottom-up) approach have been discussed, their Schema Mapping methods are focusing on whether two entities are identical or not. In a DEKG, the Schema Matching is more precise, with different link types between entities, relationships and properties (identical, similar, extends, ...). Therefore, those different link types need to be considered during the query decomposition.

2.2. Virtual Knowledge Graph

Previous work in EKG mainly relies on data integration. Recent work tend to overpass some limitation of such data integration strategy in designing a virtualized alternative of an EKG.

Virtual Knowledge Graph (VKG) is a paradigm which allows an end-users to access data without ingesting it by leveraging data virtualisation [12] over a Distributed Database system. For instance, Ontop[1] is a VKG solution focused on exposing a Knowledge Graph from multiple Relational Databases, using SPARQL endpoints and R2RML Schema Matching. The VKG, in its current state, seems to be focusing on adding the Knowledge Graph philosophy over existing relational databases only.

2.3. Discussion

Previous work mainly focused on either centralised system, or relational decentralised systems. As an alternative, we proposed in previous work [11] a Decentralised EKG architecture to offer the simplicity of a centralised EKG with heterogeneous and distributed data sources. Previously cited works are not applicable to this Decentralised EKG, where the data sources are matched using different semantic links. In this paper, we detail the remaining challenge related to DEKG and a new method to manage queries in a Decentralised EKG that is designed for distributed data such as "data silos" and which are shown within the business view as defined earlier. The following section of this paper discuss the DEKG architecture before detailing the 4-step query management approach we propose.

1. https://ontop-vkg.org/

3. Query management in a DEKG

3.1. DEKG principles

In a Decentralised Enterprise Knowledge Graph, the Data is distributed on multiple heterogeneous systems. The different data schemata are deduced, copied, matched in a Global Schema and shown to the end-user as the Business View [11]. The Figure 1 shows how the different schemata are interconnected from bottom (data sources) to top (the Business end-user).

TABLE 1. DATA SOURCE STRUCTURES AND CONTENT

DATABASE

Students Table

id	Last Name	First Name	Class (FK)
1	Doe	John	A
2	Doe	Jane	B

Class Table

id	Teacher (FK)
A	Alice Machin
B	Bob Lambda
C	John Doe

Teachers Table

id	Full name
1	Alice Machin
2	Bob Lambda
3	John Doe

People.csv

LastName	FirstName
Machin	Alice
Lambda	Bob
Doe	John
Doe	Jane

Grades.csv

FULL NAME	GRADE	COURSE
John Doe	15	M3xxx
Jane Doe	16	M1xxx

In order to be independent of all the underlying source types and systems, the different source schemata are represented as a graph of nodes and links. Each source s_n schema is represented by a graph g_n. This graph contains different nodes types, which can be Entity E_n, Relationship R_n, and Property P_n as shown on the Figure 1. All these nodes are linked together with L_n structural links, with different link types: $hasRel$ linking Entities to Relationships and $hasProp$ linking Entities and Relationships to Properties. For instance, the source schemata described in Table 1 are represented as graphs in the Figure 2.

The Global Schema GS is the union of all the source graph schemata $g_1, g_2, ..., g_n$, using multiple Schema Matching general algorithms and rules [11]. The GS contains a set of semantic links l_{SM}, Entity nodes E_{SM}, Relationship nodes R_{SM}. l_{SM} links can be typed as $identical$, $similar$, $extends$, $includes$ or $aggregates$, and may also contain additional values (i.e. inclusion percentage). All the Entities, Relationships and Properties in the Global Schema GS have pointers to their original *Source URIs*, using nodes attribute, which will allow us to query the original data sources to reply to the end-user query. To continue on the previous example, Figure 3 shows the final GS, where the previous

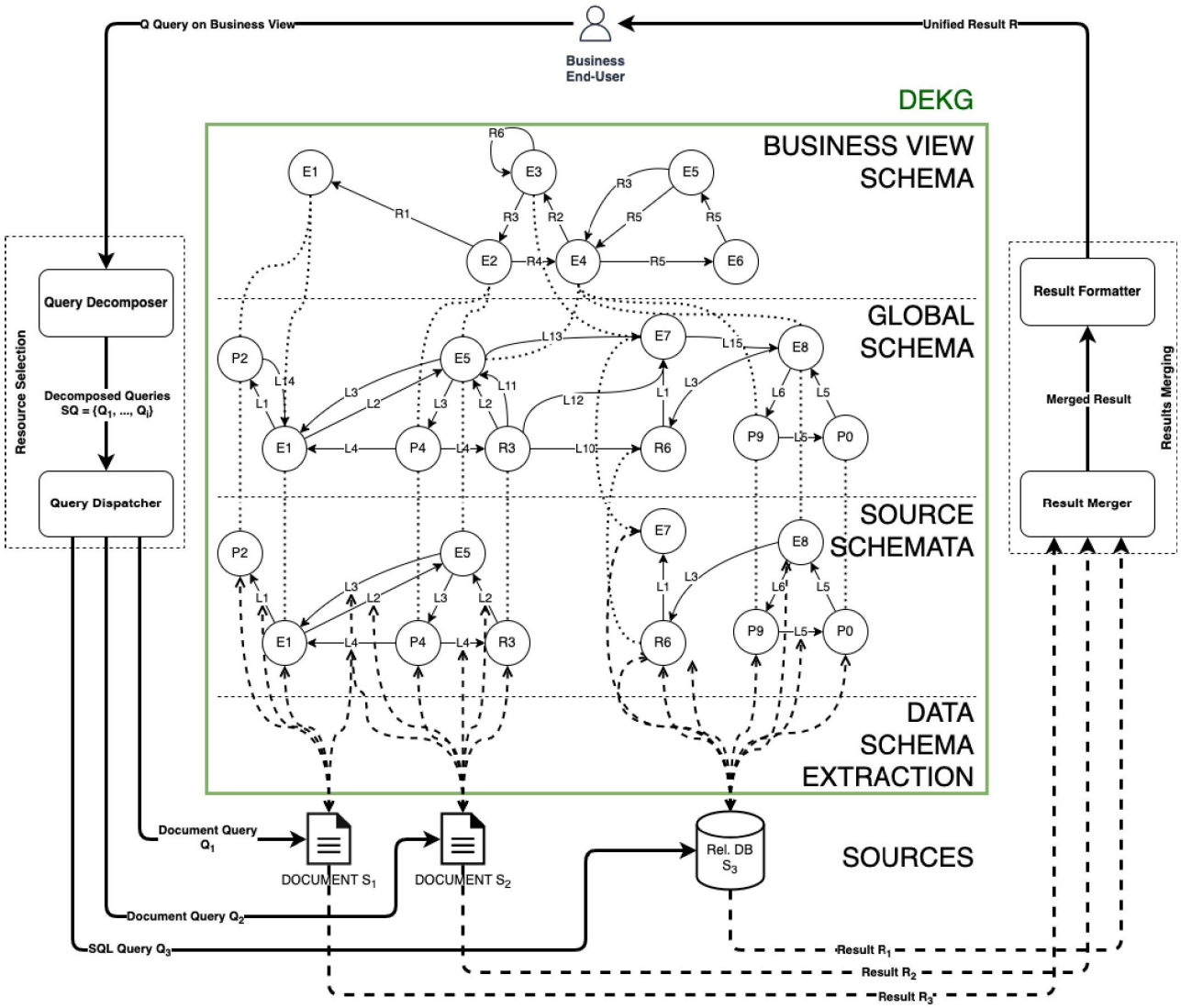

Figure 1. Distributed Query Management

schemata from deduced from the sources in Table 1 were matched together using general algorithms. The coloured arrows represent links that have been added during the DEKG schema matching (different colours mean different algorithms used).

Finally, those new links are used to create the Unified View, called Business View, shown to the top non-technical end-user. The Business View BV is a synthesis of the Global Schema GS using a grouping algorithm κ:
$BV = \kappa(GS)$
The objective of the algorithm ρ is to present a business vision to end-users, that focuses on conceptual entities connected through semantic relationships, while ignoring the physical implementation of the underlying databases. In the BV, the Entity nodes are kept as nodes, whereas Relationship nodes become links and Property nodes are included inside Entities and Relationships as attributes. Also, identical, similar and extended Entities and Relationships can be grouped together to simplify the final Business View.

Following the previous example, Figure 4 shows the deduced *business view* from the GS Global Schema. The end-user queries the DEKG directly using that proposed *business view*. We detail in the further sections how the end-user query is modelled, and the algorithms used to answer such queries.

3.2. Querying a DEKG: general principles

The end-user expresses their query using the BV. Our system will need the GS to query the different sources behind the queried entities, and reply a unified result to the

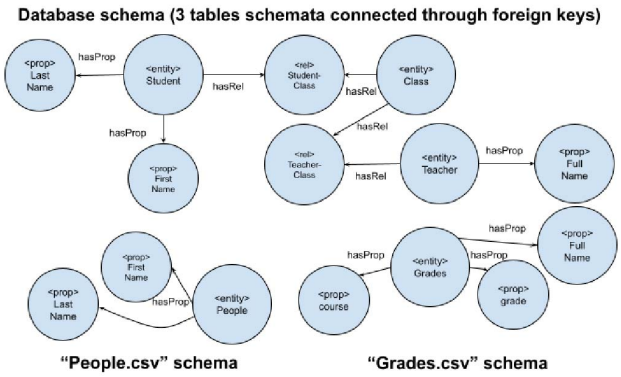

Figure 2. Data source deduced graph schema

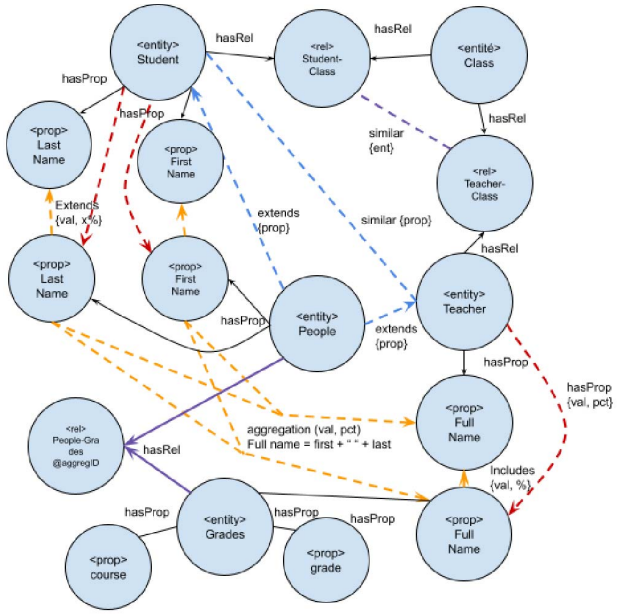

Figure 3. Full matched Global Schema example

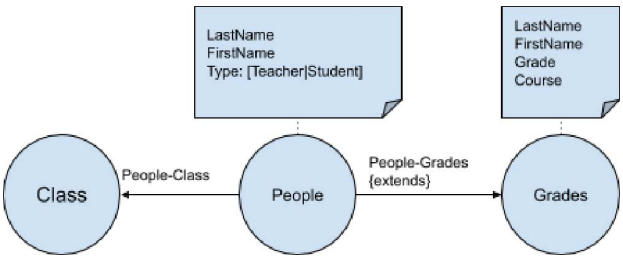

Figure 4. Generated Business View

end-user. To do so, multiple sub-queries will be built from the user query depending on the queried entities and their corresponding sources and all their sub-results merged into a single result, matching the original user query.

3.2.1. User's query expression. To be independent of any query languages, we model the user query Q expressed on the Business View BV using Graph Pattern Matching formalised as a Conjunctive Query. In a graph context, a Conjunctive Query is defined as the following expression: $CQ = \{(h_1, ..., h_i) | \exists b_1, ..., b_j \, (a_1 \wedge ... \wedge a_k \wedge r_1 \wedge ... \wedge r_l)\}$ where $h_1, ..., h_i$ is a set of head variables, $b_1, ..., b_j$ is a set of body variables, and $a_1, ..., a_k$ is a set of selection conditions and $r_1, ..., r_l$ is a set of relationship conditions [9]. Head h_n and body b_n variables can be Nodes, Relationships, or their attributes. Head variables are replied in the final reply, where body variables are used to apply selection or relationship conditions without them to be included in the final reply. The labelling functions are $\eta_p(n)$ giving the label of a node n in G, and $\xi_p(e)$ giving the label of an edge e in G. Finally, the relationships are expressed using either $\varepsilon_{DE}(n_1, e_1, n_2)$ function which represents a direct edge e_1 between the two nodes n_1 and n_2; or $\varepsilon_{\Phi}(n_1, e_1, ..., e_m, n_2)$ which represents a path of m connected edges between the two nodes n_1 and n_2.

Additionally, we allow the end-user to specify a $broad$ parameter B, which gives the ability to query more data sources following the "$similar$" link types created during the Schema Matching process.

Example. We will follow an example illustrating all the steps, based on the sources depicted in Table 1. The end-user wants to display all the People, all the Classes and all the People-Class relationships where People and Classes are linked by a People-Class relationship. Its query Q would be expressed as the following Conjunctive Query: $Q = \{p, pc, c | \exists (\emptyset) \, \varepsilon_{DE}(p, pc, c) \wedge \eta_p(p) = People \wedge \eta_p(c) = Class \wedge \xi_p(pc) = People\text{-}Class \wedge B\}$

3.2.2. Decentralised querying problems. In order to successfully answer the user query, we need to address two main problems [13], [5]: the **Resource Selection**, and the **Results Merging**. The **Resource Selection** objective is to determine subsets of queries to be executed on the underlying data stores (query decomposition). On the other side, the **Results Merging** objective is to combine all the received sub-replies into a unified reply to the user.

3.2.3. Preliminary Concepts. In order to query the graph, each entities, relationships and their properties contain the following information:

- Source S_e: Source URI of the element e
- Type T_e: Type of the e element (Entity, Relationship, Property)
- Name N_e: Name of the e element
- Level L_e: Context of the queried e element: $main$ if it is directly queried by the user and sub or $similar$ if it is required by another element.
- Parent Element P_e: If the Level L is sub or $similar$, represents the main element that requires the e element to be queried

The same way, we define multiple fuctions to get information from the DEKG Global Schema and query the underlying sources:

- $\gamma(BV, GS, e)$: From any element e in a Business View BV, returns the corresponding $element$ from the GS.
- $\beta(S, C, L, P)$: Builds a query on the source S with the required set C of selection conditions. If the L level specified is sub or similar, the parent element P must be specified.
- $\lambda(GS, e, T)$: Returns all elements linked to an element e within the Global Schema GS by a link of a specified type T.
- $\epsilon(S, Q)$: Executes the set of queries Q on the specified source S.
- $\rho(R, GS)$: Rearrange rows of a unified reply R from a sub or similar type using their shared properties in GS.
- $\pi(GS, e_1, e_2, ..., e_n)$: Returns a set of property types from GS that all the given elements $e_1, e_2, ..., e_n$ have in common (linked by identical, similar or includes links).

3.3. Resource Selection

We designed two different algorithms aiming at resolving the Resource Selection. First, the *Query Decomposer* aims at processing the initial Query Q made on the Business View BV from the User, and decomposing it as as a set of sub-queries SQ using the semantics links contained in the GS. The second algorithm, named Query Dispatcher, then sends each sub-query of SQ to the different concerned data sources and translates the concerned Conjunctive Query into the source native query language.

The Algorithm 1 represents this first step operation: it iterates over all the $elements$ from Q, meaning all entities and relationships stated in the query, and includes them to the sub-queries set SQ. The SubQueries SQ is a set of all the queries corresponding to Q. Each entry in SubQueries represent an entity or relationship to be queried, with their source, and eventual $level$ and $parentElement$. Then, it smartly selects all other useful elements: sub-elements (linked by $extends$ links) of the queried elements, as well as $identical$ elements and $similar$ elements (if user specified it using the B broad option). Our solution offers the advantage of using the semantic links previously defined during Schema Matching.

Algorithm 1: Query Decomposer

Data: GS (Global Schema), BV (Business View), Q (Query)
Result: SQ (SubQueries)

$SQ \leftarrow \emptyset$;
foreach $e \in Q$ **do**
  $GS_e \leftarrow \gamma(BV, GS, e)$;
  $S \leftarrow Q.filter(\text{"}element\text{"} = e)$
   $SQ.push(\beta(GS_e.source, S, \text{"}main\text{"}))$;
  $E_{sub} \leftarrow S.types$;
  if $E_{sub} = \emptyset$ **then**
    $E_{sub} \leftarrow$
     $\lambda(GS, GS_e.relationships, \text{"}EXTENDS\text{"})$;
  foreach $e_s \in E_{sub}$ **do**
    $SQ.push(\beta(e_s.source, S,$
      $\text{"}sub\text{"}, e.Type))$;
  $E_{sim} \leftarrow \lambda(GS, GS_e, \text{"}IDENTICAL\text{"})$;
  if $Q.broad$ **then**
    $E_{sim}.push(\lambda(GS, GS_e, \text{"}SIMILAR\text{"}))$;
  foreach $e_s \in E_{sim} \setminus E_{sub}$ **do**
    $SQ.push(\beta(e_s.source, S,$
      $\text{"}similar\text{"}, e))$;
return SQ

Algorithm 2: Query Dispatcher

Data: GS (Global Schema), SQ (SubQueries)
Result: SR (SubReplies)

$SR \leftarrow \emptyset$;
$Qs \leftarrow SQ.groupBy(\text{"}source\text{"})$;
foreach ***tuple*** $S, Q \in Qs$ **do**
  $SR.push(\epsilon(S, Q))$;
return SR

Example. Following the previously defined query Q, we select and query People and Class entity sources and also the People-Class relationship source. We will then follow all $similar$ (due to the broad B setting) and $extends$ link inside the Global Schema. That means, we will query the source corresponding to the $People$ sub-entities: the $Student$ and the $Teacher$ entities. The same goes for the $People - Class$ relationships, which is decomposed as $Student - Class$ and $Teacher - Class$ relationships

TABLE 2. SUB-QUERIES OF Q EXPRESSED IN CQ

Sub-queries CQ	Type	Level	Parent	Source
$\{p \mid \exists\, (\emptyset)\, \eta_p\, (p) = People\}$	Entity	main		People.csv
$\{s \mid \exists\, (\emptyset)\, \eta_p\, (s) = Student\}$	Entity	sub	People	db.Student
$\{t \mid \exists\, (\emptyset)\, \eta_p\, (t) = Teacher\}$	Entity	sub	People	db.Teacher
$\{sc \mid \exists\, (\emptyset)\, \xi_p\, (sc) = Student\text{-}Class\}$	Relationship	sub	People-Class	db.Student
$\{tc \mid \exists\, (\emptyset)\, \xi_p\, (tc) = Teacher\text{-}Class\}$	Relationship	sub	People-Class	db.Teacher
$\{c \mid \exists\, (\emptyset)\, \eta_p\, (c) = Classes\}$	Entity	main		db.Classes

queries. The resulting Sub-Queries for this example, to be executed on each source from this step's operation is depicted in the Table 2.

The *Query Dispatcher* then groups all sub-queries regarding the same source using key-value pairs, as depicted in algorithm 2, minimising the different system impacts. All the queries are then translated and sent to their original sources in parallel. All replies are stored as set of sub-replies SR, corresponding to their respective sub-queries in SQ. The results of sub-queries executed in this step is passed onto the **Results Merging** steps as a set of source sub-replies.

Example. Here, each Sub-query in SQ, depicted in Table 2, are executed onto each corresponding Source. For each source, we get a set of replies, as follows:

```
SR = {
    db.Student = { { Properties = {
        FirstName = 'Doe',
        LastName = 'Joe', ... },
        type = 'entity',
        level = 'subElement',
        parentElement = 'People' },
                    { Properties = {
            FirstName = 'Jane',
            LastName = 'Doe',
            ... },
        ... },
    ... },
    db.Teacher = { ... },
    ...
    db.Classes = { ... }
}
```

3.4. Results Merging

The *Result Merger* uses all sub-replies queried by the Query Dispatcher and merges them into a single, unified reply based on the Global Schema links. This step, described by the algorithm 3, first joins all the replies corresponding to queried relationships and their linked entities together. It continues with unifying all the other queried entities replies and finally joins all the replies corresponding to sub, identical and similar elements to their parent elements.

Example. Following our example, the replies from the previous step are merged into a single reply. Here, $Teacher$ and $Students$ data will be joined with $People$ reply data. To join correctly, it will be done using the properties couple $FirstName$, $LastName$ of $People$ and $Student$, and the $FullName$ property of $Teacher$. The $Class$ data is merged using the existing $Student - Class$ and $Teacher - Class$ relationships. The Unified reply can now be depicted as a table of properties (Table 3). The meta-data for each entity and relationships (level, parentElements...), despite not being explicitly shown in the Table, are still stored.

The *Result Formatter* is the last step. Its goal is to improve the readability and usability of the reply by matching the reply with the original BV. The algorithm 4 consists in rewriting the original $Types$ of the entities and relationships

Algorithm 3: Result Merger

Data: GS (Global Schema), SR (SubReplies), Q (Query)
Result: R (reply)

$R \leftarrow \emptyset$;
foreach $r \in Q.filter(\text{"}type\text{"} = \text{"}relationship\text{"})$ **do**
  $P, S \leftarrow \lambda(r, GS)$;
  $L \leftarrow SR.filter(\text{"}element\text{"} = r)$;
  $L \leftarrow innerJoin(L, SR.filter(\text{"}element\text{"} = P), \pi(GS, r, P))$;
  $L \leftarrow innerJoin(L, SR.filter(\text{"}element\text{"} = S), \pi(GS, r, S))$;
  $R \leftarrow R.union(L)$;
foreach $entity \in Q.elements.filter(\text{"}type\text{"} = \text{"}entity\text{"})$ **do**
  $R \leftarrow R.union(entity)$;
foreach $e \in SR.filter(\text{"}level\text{"} = \text{"}subElement\text{"})$ **do**
  if $Q.filter('element' = e) \neq \emptyset$ **then**
    $R[e.parentElement] \leftarrow innerJoin(R[e.parentElement], e)$;
  else
    $R[e.parentElement] \leftarrow outerJoin(R[e.parentElement], e)$;
foreach $e \in SR.filter(\text{"}level\text{"} = \text{"}similarElement\text{"})$ **do**
  $R[e.parentElement] \leftarrow outerJoin(R[Sub.parentElement], e)$;
return R

Algorithm 4: Result Formatter

Data: GS (Global Schema), R (Reply)
Result: UR (Unified Reply)

foreach $e \in R.filter(\text{"}@level\text{"} = \text{"}subElement\text{"})$ **do**
  $T_e \leftarrow N_{P_e}$;
foreach $element \in R.filter(\text{"}@level\text{"} = \text{"}similarElement\text{"})$ **do**
  $T_e \leftarrow N_{P_e}$;
return $\rho(R, GS)$

to their corresponding parent Types. Therefore, the new Type of each element corresponds to the original element type of the end-user Q query on the Business View BV. Additionally, the raw results are grouped by all their common properties, in order to offer the user more readability.

Example. The different results of $Teacher$ and $Student$, resulting from the previous step, will be unified as $People$ and a new column $Type$ will be added. This allows the user keep a track of the underlying entity that

TABLE 3. MERGED QUERY RESULT

People			**Class**
Teacher			
LastName	**FirstName**	**id**	**id**
Machin	Alice	1	A
Lambda	Bob	2	B
Doe	John	3	C
Student			
Doe	John	1	A
Doe	Jane	2	B

TABLE 4. FORMATTED QUERY RESULT

People				**Class**
LastName	**FirstName**	**Type**	**id**	**id**
Machin	Alice	Teacher	1	A
Lambda	Bob	Teacher	2	B
Doe	John	Teacher	3	C
		Student	1	A
Doe	Jane	Student	2	B

matches the meta-data provided by the Business View BV. The formatted reply in our example is presented in the Table 4.

4. Implementation & Experiments

In order to evaluate our proposition, we implemented the algorithms previously presented and demonstrate our proposition feasibility. We conducted an experimentation on different types of queries and datasets described below, and gathered key metrics. We used Javascript (NodeJS), getting a JSON formatted Conjunctive Query, the Global Schema, the Business View, and returns the reply as a JSON array with the execution time and intermediate metrics. We based this implementation on the DEKG architecture [11] and assumed here that all our sources were fully working and replying back tabular content of the source entities and relationships.

We executed our experiments with three datasets, which are presented in the first part of Table 5, which summarises the sources characteristics and those results. It first presents the three sources characteristics with the underlying source composition and database types. The second part of the table shows the resulting Global Schema and Business View composition.
Those three sources represent different use cases and different structural characteristics, whether in data volume, or number of total structural attributes they contain. Comparing those three sources will allow us to validate our proposal.
The first source is our toy dataset, presented and used in the Section 3. It has a low volume of data, but the schema is semantically rich. The second dataset is built against three France OpenData sets, which can be used to easily search vaccination centers from all Toulouse surrounding cities. This dataset is the highest volume we experimented, but is really well structured. Finally, we used enterprise employees skills data set used for Team of Teams, that was acquired with internal survey. This latest is lower volume than France OpenData but contains a higher number of properties per entities.

We experimented around different types of queries on each source: Single entity type selection, multiple entity types selection, relationship selection with two entities. We also tried out manually joining two entity types when they were not linked by a relationship, in the case where the schema matching was not able to match two similar entities. We gathered some key metrics about those queries. The results per query are available on our Github repository[2]. We selected the 4 more interesting queries in Table 6, listing their metrics, type and sources. The Decomposed and Merged columns show the number of decomposed entities and the number of results merged using the Global Schema semantic links and the Volume column depicts the number of elements returned in the query. The goal is to show

2. https://github.com/Mavyre/dekg-query-reply

TABLE 5. QUERY & REPLY EXPERIMENT METRICS

	Dataset	School example	France OpenData	Team of Teams
Sources	Source composition	DB + 2 CSV	3 REST APIs	2 CSV
	Total tables/documents	5	3	2
	Min/Max row per table/file	2 - 4	22 - 2307	34 - 40
	Total number of attributes	34	9582	10524
	Data volume	1 KB	12MB	340KB
DEKG	GS Nodes	22	52	281
	GS Links	31	56	7766
	BV Entities	5	10	214
	BV Relationships	6	13	7681

TABLE 6. PER-QUERY METRICS

Source	Query Type	Avg Time	Decomposed	Merged	Volume
School	Single Node subtype + Broad selection	41.044ms	0	0	4
School	Two entities with Relationship selection + single entity selection	40.213ms	4	2	12
OpenData	All nodes selection	89.536ms	0	0	2366
Team of Teams	Manually joined nodes	215.386ms	0	0	68

the feasibility of our proposition, and that the different algorithms are used to enhance the end-user final reply.

The results show that the query execution time mainly depends on the structure of the Global Schema and Business view, rather than the data volume queried and the number of properties per entity. For instance, the data in the Team of Team source have a lot more properties per entity than the other sources and a higher data volume, but still outperforms all the Team of Teams queries as it is structurally less complex. Also, the number of decomposed types doesn't entirely depend on the number of Semantic Links included in the Global Schema. Instead, it depends on the quality and their types (*extends*, *identical* and *similar* are the most important ones) and the compression rate between the BV and the GS. In our experiments, the company Team of Teams and OpenData France didn't trigger any decomposition as the source data schemata were not redundant, despite being related, showing the importance of the Global Schema construction and completeness to fully use the Query Decomposition and Results Merging algorithms.

However, even without the use of semantic links, these experiments show that our approach can still help query multiple sources at once without the end-user intervention. All the queries we executed in those experiments would need multiple manual queries on underlying sources (in the case of DB sources), manual file search and manual matching to obtain the same result as we did. In the case of big data sources, the manual approach is not a viable, nor doable option, and would have needed a specific script for each different queries we did. This shows that our approach enables an end-user to access to the data of multiple sources without any technical knowledge of the underlying sources.

5. Conclusion and future work

Non-technical end-users in organisation really need easy access to organisation information by focusing on business entities and their relationships without the need for unnecessary technical information or actions, such as locating source data in heterogeneous environments. To address this issue, we have defined the DEKG principle and in this paper and addressed the problem of querying its multiple underlying sources. To do so, we proposed a query process to answer Conjunctive Queries using semantic links contained in a DEKG, and shown how this query is then executed on the different sources. The proposed process is based on 4 steps we detailed throughout the paper: Query Decomposition, Query Dispatching, Result Merging and Result Formatting. Those steps use semantic contained in a DEKG Global Schema in order to provide the end-user all the data available and efficiently address the problems raised by heterogeneous data sources querying. Finally, our experiments highlighted that we allow a user to query the DEKG multiple data sources at once, directly from the simple Business View. We also shown that we provide the end-user a merged and formatted result using the Global Schema semantic links.

We're now working on extending our proposal by adding aggregation functions, such as COUNT, SUM, MIN, MAX, AVERAGE... Such functions might be really useful for business end-users, especially about decision-making queries. We're currently working on adding these functions even if one or more of the underlying data source do not support it natively. We also are continuing experimenting our approach against real-world very high-volume data which can contain inconsistencies between the different sources. We're also working on interactive and user-friendly user interface that will allow a non-technical end-user to easily browse and query a DEKG. The user queries would then be translated from the interface to a Conjunctive Query for the system to process them.

References

[1] B. Czejdo, M. Rusinkiewicz, and D. W. Embley. An approach to schema integration and query formulation in federated database systems. In *1987 IEEE Third International Conference on Data Engineering*, pages 477–484, Los Angeles, CA, USA, Feb. 1987. IEEE.

[2] R. Denaux, Y. Ren, B. Villazon-Terrazas, P. Alexopoulos, A. Faraotti, and H. Wu. Knowledge Architecture for Organisations. In J. Z. Pan, G. Vetere, J. M. Gomez-Perez, and H. Wu, editors, *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, pages 57–84. Springer International Publishing, Cham, 2017.

[3] L. Ehrlinger and W. Wöß. Towards a Definition of Knowledge Graphs. In *Joint Proceedings of the Posters and Demos Track of 12th International Conference on Semantic Systems*, page 4, Sept. 2016.

[4] J. M. Gomez-Perez, J. Z. Pan, G. Vetere, and H. Wu. Enterprise Knowledge Graph: An Introduction. In J. Z. Pan, G. Vetere, J. M. Gomez-Perez, and H. Wu, editors, *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, pages 1–14. Springer International Publishing, Cham, 2017.

[5] A. K. A. Hassan and M. J. Hadi. Distributed Information Retrieval Based On Metaheuristic Search and Query Expansion. page 9, 2017.

[6] Kendall Clark. What is a Knowledge Graph, June 2017.

[7] A. Moschitti, K. Tymoshenko, P. Alexopoulos, A. Walker, M. Nicosia, G. Vetere, A. Faraotti, M. Monti, J. Z. Pan, H. Wu, and Y. Zhao. Question Answering and Knowledge Graphs. In J. Z. Pan, G. Vetere, J. M. Gomez-Perez, and H. Wu, editors, *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, pages 181–212. Springer International Publishing, Cham, 2017.

[8] J. Z. Pan, G. Vetere, J. M. Gomez-Perez, and H. Wu, editors. *Exploiting Linked Data and Knowledge Graphs in Large Organisations*. Springer International Publishing, Cham, 2017.

[9] C. Sharma, R. Sinha, and K. Johnson. Practical and comprehensive formalisms for modelling contemporary graph query languages. *Information Systems*, 102:101816, Dec. 2021.

[10] D. Song, F. Schilder, S. Hertz, G. Saltini, C. Smiley, P. Nivarthi, O. Hazai, D. Landau, M. Zaharkin, T. Zielund, H. Molina-Salgado, C. Brew, and D. Bennett. Building and Querying an Enterprise Knowledge Graph. *IEEE Transactions on Services Computing*, 12(3):356–369, May 2019.

[11] B. Vidé, J. Marty, F. Ravat, and M. Chevalier. Designing a Business View of Enterprise Data: An approach based on a Decentralised Enterprise Knowledge Graph. In *25th International Database Engineering & Applications Symposium*, pages 184–193, Montreal QC Canada, July 2021. ACM.

[12] G. Xiao, L. Ding, B. Cogrel, and D. Calvanese. Virtual Knowledge Graphs: An Overview of Systems and Use Cases. *Data Intelligence*, 1(3):201–223, June 2019.

[13] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer New York, New York, NY, 2011.