Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea magistrale in Ingegneria e Scienze Informatiche

Implicit Data Aggregation on a Graph-based Data Integration System

Relatore:
Matteo Golfarelli

Presentata da:
Filippo Pistocchi

Collaboratori:
Oscar Romero, Sergi Nadal (Universitat Politècnica de Catalunya, BarcelonaTech)

Sessione IV
Anno Accademico 2021/2022

Vorrei dedicare questa sezione a tutti coloro che hanno contribuito con il loro supporto allo sviluppo di questo progetto e coloro che mi hanno sostenuto durante il mio percorso di studi che mi ha portato al conseguimento della Laurea Magistrale in Ingegneria e Scienze Informatiche. Vorrei ringraziare in primis il Prof. Matteo Golfarelli, che mi ha dato la possibilità di andare in Erasmus a Barcellona per poi successivamente continuare l'esperienza all'estero svolgendo anche l'attività di tesi e tirocinio. Ci tengo inoltre a fare un grande ringraziamento al Prof. Oscar Romero e al Prof. Sergi Nadal per l'accoglienza ricevuta presso il gruppo di ricerca DTIM e il supporto ricevuto durante lo sviluppo di questo progetto. Vorrei fare anche un ringraziamento speciale ai miei genitori e a tutta la mia famiglia per avermi supportato durante questo lungo percorso di studi. Sono convinto di avervi resi orgogliosi di me. Ringrazio anche i miei cari amici Cristiano, Emanuele, Federico, Michele e Tommaso per avere condiviso tantissimi bei momenti durante tutti gli anni di studio passati assieme. Volevo ringraziare inoltre tutte quelle persone che ho conosciuto a Barcellona durante l'Erasmus, come Antoine, Eric, Frederik, Gabriele, Jonas, Jeoren, Patrick, Sofia, Tomas, e molti altri, per avere contribuito a rendere speciale questa esperienza. Ci tengo a ringraziare infine anche Aleksandra che mi è stata vicina per questi ultimi due anni di studio e che mi ha fatto ritrovare quella motivazione che stava con il passare degli anni diminuendo e mi ha dato la forza di andare avanti. Senza di voi non ce l'avrei mai fatta.

Abstract

Nowadays, IoT and social networks are the main sources of big data, they generate a massive amount of assets and companies have to develop data-driven strategies to exploit the value of information that's behind data. The shape of data sources is typically heterogeneous, since data can be generated from different sources distributed all around the world. The sparsity and the heterogeneous shape of data make much more difficult the process of data wrangling and knowledge discovering, and these are the reasons why data-driven companies must use data integration techniques to address this complexity.

The DTIM research group at Universitat Politècnica de Catalunya (UPC) upon I have been working with is interested in such thematic and in 2015 they developed Graph-driven Federated Data Management (GFDM), that proposes in a very intuitive way a graph-based data integration architecture. What we would like to do in this project is to extend GFDM, to support automatic data aggregation following the OLAP data processing grounded on multidimensional modeling, as data warehouses do, but on top of graph data. This idea will be carried out by developing a framework able to perform OLAP-like queries over GFDM, mainly focusing on the well-known Roll-Up operation. In this thesis we have developed a method that given a query is able to align data coming from different data sources and sitting at different granularities level that participate in the same conceptual aggregation hierarchy. Our method is able to identify implicit aggregations that would allow to align data from different data sources and integrate them seeminglessly at the correct granularity level. After an accurate design and implementation phase we can finally consider our goal accomplished, developing with success the Implicit Roll-Up algorithm satisfying our requirements.

Contents

List of Figures

List of Algorithms

Chapter 1

Motivation

The idea of this project came from the very well known necessity of facilitating as much as possible the data scientist job of data wrangling, since to develop robust mathematical, machine learning and in particular deep models it is very important to have access to huge amount of (correct) data. Nowadays, IoT and social networks are the main sources of big data, generating a massive amount of assets. Accordingly, stakeholders in the organizations have to deal with heterogeneous datasets generated independently and by different processes. Clearly handling such resources it is not a trivial task and to do so without specific tools it may become very complex. Data integration systems have been developed to overcome these problems, with the purpose of re-conciliating huge amount of heterogeneous and distributed data sources, facilitating the process of knowledge discovering, making it possible to navigate data as a single integrated schema.

First, we will discuss the two main approaches of data integration: Physical and Virtual data integration. Afterwards it will be present Graph-driven Federated Data Management (GFDM), a virtual data integration system based on graphs developed by the DTIM research group at Universitat Politècnica de Catalunya (UPC). Finally, as a goal of this project we would like to develop a framework able to implement the OLAP (On-line Analytical Processing) operations, typically associated to physical data integration models, over GFDM which does not support this framework by default. Consider as an example the following scenario; A company may buy IoT sensor data from different organizations. The first source of sensor data provides the number of cars circulating in a certain relevant road of a city per hour. While another source may provide data of other streets per day. A simple query such as the total number of cars of any street of that city per day would, typically, only take into account the data source providing daily data. However, this is a clear example that hour and day participate in the same aggregation hierarchy, and an implicit aggregation from hour to day might let us use both the first and the second data source to answer this query.

1.1 Data integration techniques

The aim of this section is to open a small parenthesis to describe the main data integration techniques, detailing a bit more precisely their main characteristic, in order to make it easier to understand the motivations behind this work.

1.1.1 Physical data integration

Physical data integration integrates data by means of ETL (extract-transform-load) processes. These processes extract data from disparate potentially heterogeneous sources as Data lakes, information systems or external data sources (E), transform them (T), cleaning, homogenizing and preparing data to finally load them in a multidimensional schema (L). Thus, the target multidimensional schema contains instances generated from the transformation and integration of individual instances extracted from the sources. The multidimensional view of data is distinguished by the fact, dimension and level dichotomy, and it is characterized by representing data as if placed in an n-dimensional space, allowing us to easily understand and analyze data in terms of facts (the subjects of analysis), dimensions showing the different points of view from where a subject can be analyzed and levels describing the granularities of each dimension. A multidimensional schema is a formalism (Entity Relationship like) that allows to model data behaving to multidimensional domains that will be then implemented into a relational database through tables.

Querying over multidimensional structures is then mainly performed by means of OLAP (On-line Analytical Processing) which in essence performs aggregations and other operations equivalent to descriptive statistics analysis. The most prominent OLAP operators are Roll-Up and Drill-Down, which are the opposite operation. The Roll-Up operator aggregates data to a coarser granularity level on an aggregation hierarchy, while the Drill-Down operator decreases the aggregation level of a dimension. The main advantage of such integration techniques is the possibility of working directly on clean, integrated and structured data already integrated in the data warehouse in the form of a multidimensional schema, while some disadvantages are the computational costs of ETL cycles and the impossibility of working with "fresh" data. More insights related to physical data integration can be found in [3].

1.1.2 Virtual data integration

Virtual data integration do not extract, transform and load instances in the target schema. Even if the concept of target schema also exists in these systems, this target schema does not contain data but mappings to allow the system to compute, on-the-fly, via query rewriting, the target concept instances from the data sources at hand. Thus, instead of ETL, it relies on mappings between the local and target schemata. These mappings are

used by query rewriting algorithms that compute the result of queries over the target schema on several queries on the data sources and glue together their results to produce the illusion of a unified view over them. The main advantages of virtual data integration techniques are the data freshness since information are retrieved at query time directly from the real sources and also the exploitation of data locality. The disadvantage of these approaches are performance problems given that queries are rewritten in runtime and executed over different sources and finally their results merged together. For this reason, traditionally, these systems allow very limited query expressiveness (e.g. selections, projections and joins).

Let us define now the ways to define mappings, having three main approaches. The simpler vision to define mappings is the monolithic approach **GAV** (global as view), characterizing the target schema in terms of queries over the sources (the mapping is a query over the source). Then there are **LAV** (local as view) approaches adopting the well known mediator architecture, characterizing sources in terms of queries over the target schema. Finally, **GLAV** (global as local as view) approaches defines a hybrid mapping view considering both the advantages of GAV and LAV, characterizing queries over the sources in terms of queries over the target schema.

Consider finally the main solutions to implement virtual data integration, having classic (relational) database (**DB**) and knowledge representation (**KR**) approaches. DB methods define the integrated schema (TBOX) as an integrated database. The definition of such structure requires domain experts with accurate understanding of the schema for the definition of conjunctive queries through data sources relying on shared join predicates. KR based systems, differently, defines the integrated schema by means of fragment of (DL-Lite) description logics [6] offering typically a graph-based data model giving the expressiveness of a graph model and the simplicity of querying it with the drawback of having complex reasoning mechanisms underneath to enable query answering.

1.1.3 Graph-based data integration

Let us introduce now graph-based solution to perform virtual data integration adopting KR approaches. These techniques rely typically on KG (Knowledge Graphs) properties to generate (DL-Lite) ontology describing the integrated schema (TBOX) and linking shared and distributed data through semantic pointers. The use of RDFS (Resource Description Framework Schema), or even more complex ontology description languages, as OWL (Web Ontology Language), will allow defining robust constraints and taxonomies over the TBOX in the form of description logic fragments. The usage of a graph representation to describe the integrated schema has many advantages: Firstly, a graph notation will allow by definition to define flexible and extensible models. The simplicity of a graph representation will also facilitate the comprehension of the schema and as consequence the querying will be easier since it will be based on pattern-matching expressions or even simpler visual queries.

Graph-based virtual data integration techniques can be distinguished in two main families; ontology-based data access systems and ontology-mediated queries. In both the models the queries are executed over an integrated schema (TBOX modeled with knowledge graphs), formally defined as Global level, which will have the responsibility to retrieve all the queried data from each source aligning all of them delivering a complete result to the user through reasoning. The main difference between these approaches is the way of how mapping are defined. Ontology-based models relies on GAV mappings, while Ontology-mediated models are based on LAV or GLAV mapping. Ontology-mediated approaches relies on the concept of wrappers as a named query, each one exposing the variables that will be extracted to the respective connected source. Typically, the integration graphs defining the whole integration system (both the TBOX and the wrappers) are defined through graph semantic having LAV or GLAV mappings as sub-graphs of the TBOX.

1.2 Graph-driven Federated Data Management

In this section we introduce Graph-driven Federated Data Management system because it is the starting point of this work, having as goal to develop a framework able to extend such system. The DTIM research group in Universitat Politècnica de Catalunya (UPC) is interested in data integration thematic and in 2015 they developed Graph-driven Federated Data Management (GFDM), that proposes a graph-based architecture to perform virtual data integration adopting GLAV mappings [1]. GFDM system however differs to the classic idea of virtual integration systems based on ontology since it actually performs query answering without relying on reasoning, adopting ontology for the sake of defining proper constraints and taxonomies over the TBOX allowing to have a well formatted and easily accessible schema. GFDM is then a virtual data integration system and as such supports selection, projection and join.

1.3 Contribution

In this work, we aim at also including implicit aggregations and enrich the amount of automatically supported operations. To do so, we make a multidimensional interpretation of the graph (i.e., the capability to support dimensions, levels, dimension hierarchies and measures to generate multidimensional cubes). On top of that multidimensional interpretation, we will be able to perform OLAP-like aggregation equivalent to that of a traditional Roll-up. Thus, our approach is the first one to extend a graph-based virtual data integration system to incorporate OLAP-like aggregations, which do not support OLAP by default. Let's identify this structure as a multidimensional graph, which we denote as $\mathcal{MG}$.

Let us now consider the chance to have datasets providing data behaving to the same domain but at different granularities having all the possible granularities in $\{0, ..., k, ..., n \mid 0 < k < n\}$. Consider now the user querying the kth granularity level. Then what we would like to do is allow GFDM to generate a result considering not just information at k granularity but also the union of all the information provided at each lower granularity level than k, as $\{0, ..., k-1\}$, aggregated to k granularity. Let us call this operation **implicit Roll-Up**. Let use Figure 1.1 to make an example. The studied fact is *Sale* which is described by the measures *total revenue* and *number of unit*, and it is also represented in a multidimensional space with a *spatial*, *temporal* and *product* dimension. According to the query (the red dashed line) the user is asking for the second granularity level of each dimension, respectively *Region*, *Month* and *Sub Category*. Then, according to what we said before, the expected result would be given by the union of all the data provided at the queried granularity k and the aggregation of all the lower granularity level into k. We will finally have a union of all the data already provided at *Region*, *Month* and *Sub Category* granularity with the data provided at *City*, *Day* and *Product* granularity aggregated to their respective higher granularity level.

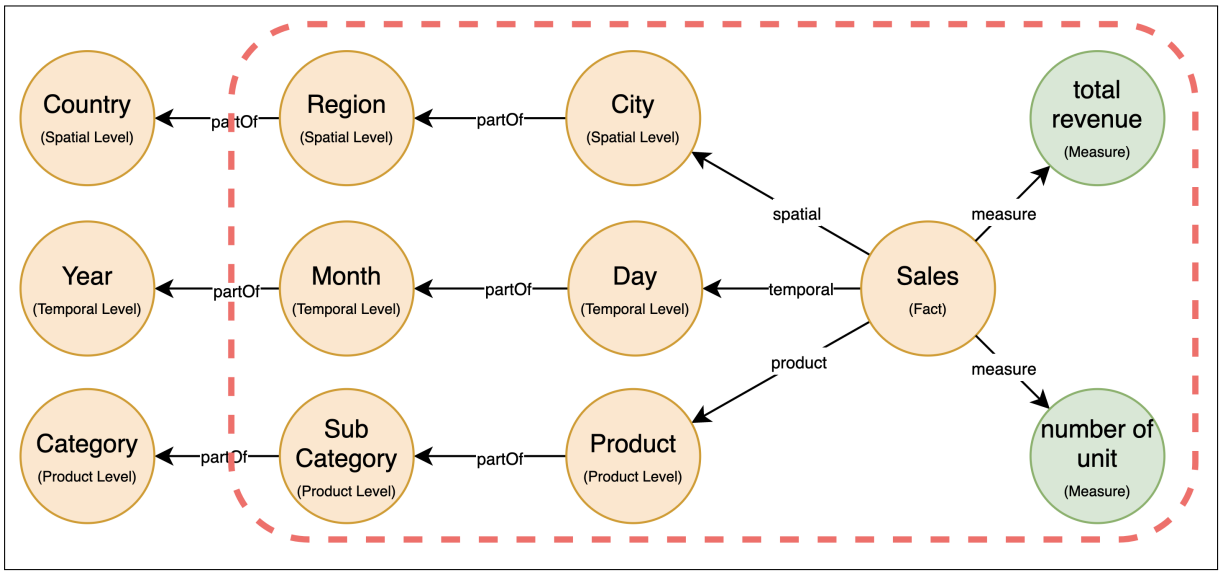


Figure 1.1: Idea of a simplified multidimensional graph $\mathcal{MG}$ having three different granularity level for three dimension and two numerical data. Also, a query have been depicted as a dashed red line.

1.3.1 Benefits of the approach

The advantages of an implicit Roll-Up operation in this case would be twofold; Firstly this operation would save a lot of time to data scientist interested in this kind of operation,

providing a huge automation process underneath GFDM querying mechanism, delivering correctly aggregated data. Secondly, implicit Roll-Up would also guarantee more correction and accuracy for the aggregated data since in the aggregation computation it will be considered both the queried granularity data measures and all the smaller granularity level measures, and very likely, the lower granularity level will have higher cardinality and this will lead to a much richer output. The definition of a generic multidimensional graph structure will also enable the possibility of developing other OLAP-like operators over GFDM system.

1.4 Outline

The rest of the paper is structured as follows. In Section 2 we will define the requirements that will be provided as a formal statements of what the framework does. Section 3 will ground this project, describing all the related works. Then we have Section 4 and 5 where the former formalizes the model adopted by GFDM and gives some intuitions of how to implement implicit Roll-Up over GFDM itself and the latter describes how implicit Roll-Up is implemented, validating theoretically the approach. Then we have Section 6 describing the prominent implementation details and Section 7 showing how the method implemented performs validating experimentally such approach. Finally, Section 8 will sum-up the work done, stating the conclusions.

Chapter 2

Objectives

This section presents the requirements of the to-be vision. All the following functionalities that the framework will have will be implemented over GFDM as a natural extension of the system. It will follow also a clear distinction of all the different kind of functionality.

2.1 Functional Requirements

This section will describe the functional requirements as hierarchies. These lists represent exactly what the system is and what the system does.

1. Perform implicit Roll-Up given a query over a multidimensional graph, automatically delivering the right data granularity.
    - 1.1 Understanding if it is possible to execute the implicit Roll-Up operation given the query.
        - 1.1.1 If it is not possible to execute the implicit Roll-Up, the output of the query would be the baseline rewriting GFDM.
        - 1.1.2 If it is possible to execute the implicit Roll-Up, the output of the query would be the query generated by performing an implicit Roll-up on top of the rewriting done by the GFDM.
    - 1.2 Extraction of group by clauses from the query.
        - 1.2.1 Parsing of the group by clauses generating the SQL select statements in string format.
    - 1.3 Extraction of aggregating clauses from the query.
        - 1.3.1 Parsing of the aggregating clauses generating the SQL statements to perform aggregation in string format.
    - 1.4 Generation of dimension queries.

1.5 Generation Roll-Up queries.

1.6 Generation of the SQL query.

1.6.1 Rewriting all the Roll-Up queries into Conjunctive queries respecting the minimality of the source query generating.

1.6.2 Mapping each Conjunctive query into SQL query.

1.6.3 Wrapping up all the SQL queries with aggregation clauses (e.g. GROUP BY statement).

1.6.4 Folding all the SQL queries into a single SQL as a union of all of them.

1.6.5 Wrapping up the final SQL query with aggregation clauses (e.g. GROUP BY statement).

1.7 Possibility to flag if performing implicit Roll-Up or not.

2. Possibility to execute the SQL query generated.

3. Definition of a programmatic model able to define $\mathcal{MG}$ in a domain scenario.

3.1 Definition of the scenario name.

3.2 Definition of the target schema.

3.3 Definition of the query.

3.4 Definition of the wrappers and attributes.

3.5 Definition of the aggregation functions.

4. Automation pipeline able to generate automatically all the configuration files for GFDM , parsing the programmatic model and generating a directory with the name of the scenario containing all the configuration files.

4.1 Generation of the target schema triples (`global_graph.txt`).

4.2 Generation of the LAV mappings for each wrapper (`mappings.txt`).

4.2.1 For each wrapper, it will be written the line number of the triples in `global_graph.txt` that behaves to the respective LAV mapping.

4.3 Generation of the SPARQL query (`query.txt`).

4.4 Generation of the source graph containing wrappers and attributes triples (`source_graph.txt`).

4.5 Generation of a file pointing each wrapper name to the CSV file path containing the respective data (`wrappers_files.txt`).

4.6 Generation of a template CSV file wrapper (`*wrapperName*.csv`).

4.7 Automatic generation of other configuration files independent to the domain scenario (`metamodel.txt`,`prefixes.txt`).

5. Definition of a model able to run experiments.
    5.1 Definition of the parameters to tests.
        5.1.1 Number of dimensions.
        5.1.2 Number of levels.
        5.1.3 Number of wrappers.
    5.2 Output generation.
        5.2.1 Milliseconds for rewriting of all the Roll-Up queries.
        5.2.2 Milliseconds for running the complete SQL query.
        5.2.3 Milliseconds for running the algorithm without considering the previous costs overhead.
    5.3 Writing the output in `experiments.csv` file.

2.2 Non-Functional Requirements

Here will follow all the requirements that are not functionalities.

1. Define the $\mathcal{MG}$ by re-using GFDM syntax.
2. The extension should be most transparent as possible to the user.
3. Multi-platform compatibility (using relative paths with ad-hoc system separators).
4. Robustness of the automation pipeline.

Chapter 3

Related work

This section aims to ground this project into its main study area, providing some comparison to other works which are similar. As we introduced in Section 1 the interest area of this project is query answering using views, taking [4] as a theoretical base defining virtual data integration approaches and also physical data integration techniques formally defined in [3].

DB-based Data-integration

DB based approach has been the pioneer of data integration, proposing GAV based systems as TSIMMIS [7], Garlic [8] and MOMIS [9]. Suddenly, with the rising of web sources also approaches LAV based have been considered having the most prominent rewriting techniques as the bucket algorithm [12], the inverse rules algorithm [13] and the MiniCon algorithm[14]. Finally, in the past years also GLAV mapping methodologies have been considered to manage problems related to data exchanges as in [10]. The description of DB-based approaches and their related work have been inspired to the respective chapter in [1]. Table 3.1 describes the works for each kind of mapping model for DB-based systems.

KR-Based Data-integration

The ontology-based data access (OBDA) approaches for data integration [5] are the main representative of graph integration techniques and are based on DL-Lite description logics [6]. The most of OBDA adopts GAV mapping as Ontop [15], Morph [16] and Mastro [17]. Also, LAV and GLAV have been developed having LAV mapping based system [18], [19] and [20] and GLAV mapping based [21], [22] and also Graph-driven Federated Data Management [1]. This paragraph is inspired to Section 2 of [1]. Table 3.1 describes the works for each kind of mapping model for KR-based systems.

	GAV	LAV	GLAV
DB	[7], [8], [9]	[12], [13], [14]	[10], [11]
KR	[15], [16], [17]	[18], [19], [20]	[21], [22], [1]

Table 3.1: Related work for the kind data and model (DB and KR) and mapping model (GAV, LAV and GLAV).

GFDM Data-integration

This work, as we introduced previously, is a framework that extends Graph-driven Federated Data Management, a graph based ontology-mediated data integration system [1]. The model adopted in GFDM to describe integration graphs will be the building block for the definition of a multidimensional graph structure. The flexibility of the graph data model offered by KGs will be used in this work to define annotations in order to define the metadata of a multidimensional model, such as facts, measures, dimension and levels. The main component of GFDM is the Rewriting Algorithm, able to rewrite a query over the integrated schema as a union of conjunctive queries over the sources. The framework developed will massively use this feature to extract information behaving to each wrapper to aggregate them to the queried granularity level.

Multidimensional graph model

As introduced before, to define a multidimensional graph it is necessary to generate annotations through the graph syntax, describing facts, measures, dimension and levels over a graph structure. We see several attempts in literature trying to do so; In [2] we see an attempt of describing multidimensional graph structures as well over a property graph model, in [23] the multidimensional graph have been described through a collection of homogeneous labeled graph snapshots, [24] and [28] do so using a homogeneous node attributed graph and [26] and [29] do so with a heterogeneous attributed graph model. There are several differences between this work and the works cited above; Firstly, the graph model adopted here is different, since none of the works presented adopts a Knowledge graphs representation. Regarding the works cited above, none of that make annotation over the TBOX as we do in this work, while annotations are directly placed on data, as the ABOX. This is mainly due to the fact that such works are not grounded in data integration and for this reason they don't really need to have a graph schema to define constraints over data. The multidimensional graph defined over the ABOX will be parsed with the aim of generating a structure similar to a data cube. Once generated the data cube, then it will be possible to perform OLAP operations on top of that.

In this works as opposite, the annotations are placed over the target schema (the TBOX) and the implementation of the OLAP operators will be done by querying properly the target graph. Then the OLAP operators will be implemented directly over GFDM

system without relying on separate cube structures.

The last point to observe is the kind of OLAP operation offered by each of the frameworks cited. The possibility of generating a multidimensional cube starting from a multidimensional graph, will allow having the typical environment in which OLAP operators are implemented, and therefore it will be easier to implement more kind of operators. This framework as opposite offers then just the possibility of performing the Roll-Up operation.

Chapter 4

Technology preliminaries

The aim of this section is to describe GFDM system, in order to give to the reader basic knowledge that will allow understanding all the reasoning behind implicit Roll-Up algorithm.

4.1 GFDM model

In this section, it will be defined the graph model adopted by GFDM. This section will not map 1-1 with the paper [1] since the purpose here it is just to give a brief introduction to the domain in order to make chapter 5 easily accessible even reading just this report. The GFDM 's schema is a connected integration graph $\mathcal{I}$, described as a knowledge graph (KG) via metadata, and this model will be the integrated view of the whole system which will give the possibility to perform data integration. As we know, KG systems are represented by means of triples in the form of $KG_{\langle S,P,O\rangle} = \{(S,P,O)|(S,O) \sqsubseteq V_{\mathcal{I}} \wedge P \sqsubseteq E_{\mathcal{I}}\}$, where S, P and O are respectively the subject, the predicate and the object, while $V_{\mathcal{I}}$ and $E_{\mathcal{I}}$ are the vertexes and the edges of $\mathcal{I}$. Consider, from now on, each element with subscript $_{\langle S,P,O\rangle}$ as composed by a set of triples.

Let us now describe the vertexes and the edges of $\mathcal{I}$ and their relationship, using Figure 4.1 as an example of the following formalization. The integration graph is divided in two main components connected to each other; The global graph $\mathcal{G}$ is a connected graph that represents metadata related to the reconciliated view of the sources, and it is also the only part of $\mathcal{I}$ that the user can see and query. The source graph $\mathcal{S}$ is also a connected graph that as opposite represents the metadata related to the wrappers and the attributes exposed by each one of them.

In the global graph $\mathcal{G}$ there are three main kinds of vertexes $V_{\mathcal{G}}$, the concepts C, the features F and also the identifier features F^{id} and two kinds of edges $E_{\mathcal{G}}$, one labeled $hasFeature$ and it is used to describe features and the other is used to link concepts

labeled with custom labels.

$$C \wedge F \sqsubseteq V_{\mathcal{G}} \tag{4.1}$$

$$F^{id} \sqsubseteq F \sqsubseteq V_{\mathcal{G}} \tag{4.2}$$

$$hasFeature \sqsubseteq E_{\mathcal{G}} \tag{4.3}$$

$$customLabel \sqsubseteq E_{\mathcal{G}} \tag{4.4}$$

It will follow now the formalization of the triple notation of $\mathcal{G}$. Consider $concepts(\mathcal{G})$ as all the concepts C_i in the global graph $\mathcal{G}$ defined by equation 4.5, $linkedFeatures(C_i)$ as all the features $F_{(i,j)}$ where C_i has a directed edge, labelled with $hasFeature$, in $F_{(i,j)}$ defined by equation 4.6 and $linkedConcepts(C_i)$ as all the concept $C_{(i,j)}$ where C_i has a directed edge, labelled with a custom label, in $C_{(i,j)}$ defined by equation 4.7.

$$\langle C \rangle \equiv concepts(\mathcal{G}) \equiv \{C_0, ..., C_n\} \tag{4.5}$$

$$\langle F \rangle \equiv linkedFeatures(C_i) \equiv \{F_{(i,0)}, ..., F_{(i,m)}\} \tag{4.6}$$

$$\langle C \rangle \equiv linkedConcepts(C_i) \equiv \{C_{(i,0)}, ..., C_{(i,o)}\} \tag{4.7}$$

The triples representing concept to feature relationship are described in Equation 4.8 as the union of all the triples in the form $\langle C_i, hasFeature, F_{(i,j)} \rangle$ where C_i is a concept in $concepts(\mathcal{G})$ and $F_{(i,j)}$ is the jth feature in $linkedFeatures(C_i)$. Then the triples representing concept to concept relationship are described in Equation 4.9 as the union of all the triples in the form $\langle C_i, customLabel, C_{(i,j)} \rangle$ where C_i is a concept in $concepts(\mathcal{G})$ and $C_{(i,j)}$ is the jth concept in $linkedConcepts(C_i)$. Consider also $predicate(C_i, C_{(i,j)})$ as the predicate linking C_i to $C_{(i,j)}$. Example 1 shows the relationship between concepts and features and Example 2 shows the relationship between concepts and concepts depicted in Figure 4.1.

$$C \rightarrow F_{\langle S,P,O \rangle} \equiv \cup_{i=0}^{n} \cup_{j=0}^{m} \langle C_i, hasFeature, F_{(i,j)} \rangle \tag{4.8}$$

$$C \rightarrow C_{\langle S,P,O \rangle} \equiv \cup_{i=0}^{n} \cup_{j=0}^{o} \langle C_i, predicate(C_i, C_{(i,j)}), C_{(i,j)} \rangle \tag{4.9}$$

Formally we have that $\mathcal{G}$ is given by the union of the triples describing concept to feature and concept to concepts relationships as illustrated in equation 4.10.

$$\mathcal{G}_{\langle S,P,O \rangle} \equiv C \rightarrow C_{\langle S,P,O \rangle} \cup C \rightarrow F_{\langle S,P,O \rangle} \tag{4.10}$$

The source graph $\mathcal{S}$, that is responsible for defining the semantic of the sources, is divided in two main kind vertexes $V_{\mathcal{S}}$ as well. There are the wrappers W and the attributes A linked to each other by the label $hasAttribute$.

$$W \wedge A \sqsubseteq V_{\mathcal{S}} \tag{4.11}$$

$$hasAttribute \sqsubseteq E_{\mathcal{S}} \tag{4.12}$$

Let us see now the triple notation for $\mathcal{S}$. Let $wrappers(\mathcal{S})$ be all the wrappers W_i in $\mathcal{S}$, defined by equation 4.13. Let also $attributes(W_i)$ be all the attributes $A_{(i,j)}$ associated to a given wrapper W_i with a *hasAttribute* relationship, defined in equation 4.14. The triple notation for $\mathcal{S}$ is finally described in equation 4.15, and it is given by the union of all the triple in the form $\langle W_i, hasAttribute, A_{(i,j)}\rangle$ for each W_i in $wrappers(\mathcal{S})$ and for each $A_{(i,j)}$ that is the jth attribute in $attributes(W_i)$, as shown in Example 3.

$$\langle W\rangle \equiv wrappers(\mathcal{S}) \equiv \{W_0, ..., W_n\} \tag{4.13}$$

$$\langle A\rangle \equiv attributes(W_i) \equiv \{A_{(i,0)}, ..., A_{(i,m)}\} \tag{4.14}$$

$$\mathcal{S}_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{n} \cup_{j=0}^{m} \langle W_i, hasAttribute, A_{(i,j)}\rangle \tag{4.15}$$

Consider now the relationship *sameAs* that will link each A in $\mathcal{S}$ to the respective F in $\mathcal{G}$. The function $sameAsTriples(\mathcal{G}, \mathcal{S})$ will generate all the triples that link each A to its respective F, as shown in Example 4. Let us finally see the triple notation for $\mathcal{I}$ as the union of all the triple of $\mathcal{G}$, all the triple of $\mathcal{S}$ and the triple connecting the two graph with *sameAs* relationship in equation 4.15.

$$\mathcal{I}_{\langle S,P,O\rangle} \equiv \mathcal{G}_{\langle S,P,O\rangle} \cup \mathcal{S}_{\langle S,P,O\rangle} \cup sameAsTriples(\mathcal{G}, \mathcal{S}) \tag{4.16}$$

The final step in the model description is the concept of LAV (Local as View) mappings. Lav mappings are strictly related to each wrapper and represent the covered area of W in $\mathcal{G}$, as a connected sub-graph of $\mathcal{G}$ (this is a simplification of LAV mappings since in [1] they are actually composed by triples of the sub-graph of $\mathcal{G}$ plus the related triples in $\mathcal{S}$). Let us use $mapping(W)$ denoting the LAV mapping related to W.

$$mapping(W)_{\langle S,P,O\rangle} \sqsubseteq \mathcal{G}_{\langle S,P,O\rangle} \tag{4.17}$$

Example 1. In Figure 4.1 the triples involving C and F in $\mathcal{G}$ are $\langle Item, hasFeature, name\rangle$, $\langle Item, hasFeature, price\rangle$, $\langle Sales, hasFeature, id\rangle$, $\langle Sales, hasFeature, number\_of\_unit\rangle$.

Example 2. In Figure 4.1 the only triple involving C and C in $\mathcal{G}$ is $\langle Sales, product, Item\rangle$.

Example 3. According to Figure 4.1 the triples for $\mathcal{S}$ would be $\langle w1, hasAttribute, name\_a\rangle$, $\langle w1, hasAttribute, price\_a\rangle$, $\langle w1, hasAttribute, id\_w1\_a\rangle$, $\langle w2, hasAttribute, id\_w2\_a\rangle$ and $\langle w2, hasAttribute, number\_of\_unit\_a\rangle$.

Example 4. Taking Figure 4.1 as an example, the triples holding the relationships *sameAs* are $\langle name\_a, sameAs, name\rangle$, $\langle price\_a, sameAs, price\rangle$, $\langle id\_w1\_a, sameAs, id\rangle$, $\langle id\_w2\_a, sameAs, id\rangle$ and $\langle number\_of\_unit\_a, sameAs, number\_of\_unit\rangle$.

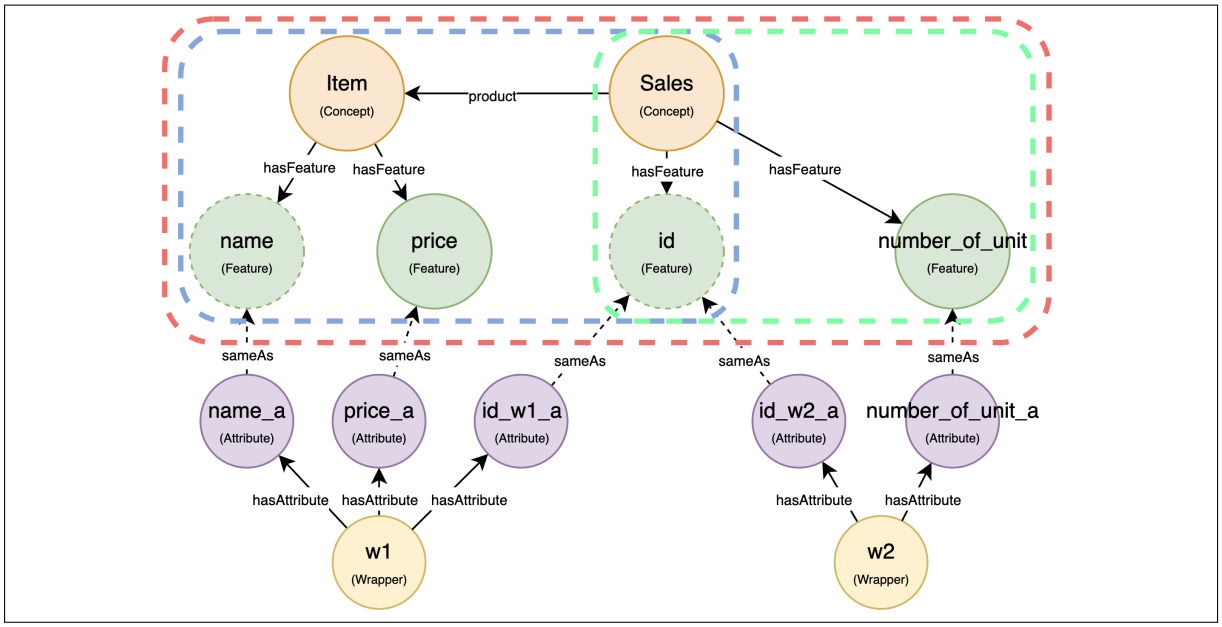


Figure 4.1: A graphical representation of the graph model adopted by GFDM. The example depicts an integration graph $\mathcal{I}$ related to sales, and a query φ over $\mathcal{G}$ (the dashed red line), the LAV mapping of wrapper *w1* (blue dashed line) and the LAV mapping of *w2* (green dashed line).

4.2 Query answering

This section will describe how querying works in GFDM and what output does a query generate. A query is defined by φ and it is a connected sub-graph of $\mathcal{G}$.

$$\varphi_{\langle S,P,O\rangle} \sqsubseteq \mathcal{G}_{\langle S,P,O\rangle} \tag{4.18}$$

The algorithm that is responsible for query answering is the *Rewriting Algorithm* (RA) and its semantic is clearly explained in paper [1]. Its purpose is to find all the single LAV mapping holding equation 4.19 and 4.21, or any possible graph connected combination of LAV mappings in $\mathcal{G}$ (the combination is done with shared F^{id} in mappings), as $\{mapping(W_0) \cup ... \cup mapping(W_n)\}$, holding equation 4.19 and 4.21.

$$\varphi_{\langle S,P,O\rangle} \sqsubseteq mapping(W_i)_{\langle S,P,O\rangle} \tag{4.19}$$

$$\varphi_{\langle S,P,O\rangle} \sqsubseteq \{mapping(W_0)_{\langle S,P,O\rangle} \cup ... \cup mapping(W_n)_{\langle S,P,O\rangle}\} \tag{4.20}$$

$$mapping(W_i)_{\langle S,P,O\rangle} \cap \varphi_{\langle S,P,O\rangle} \neq \emptyset \tag{4.21}$$

Let us call each mapping, or mapping combination, holding these properties conjunctive query $\mathcal{CQ}$ (this definition is a simplification because $\mathcal{CQ}$ is a source query in the form of

a relational algebra query, composed by wrappers, join conditions between shared F^{id} and projections).

$$\mathcal{CQ} \equiv \{mapping(W_0), ..., mapping(W_n)\} \tag{4.22}$$

Consider now $rewrite(\varphi, \varphi_{min})$ as the execution of RA over φ being minimal for φ_{min}. The minimal property is satisfied if removing any mapping in any $\mathcal{CQ}_i$, equation 4.20 or 4.19 do not hold anymore for φ_{min} (e.g. the mapping/s are not covering anymore φ_{min}). Then $\langle \mathcal{CQ} \rangle$ will be minimal pruning all the $\mathcal{CQ}_i$ non minimal.

$$\langle \mathcal{CQ} \rangle \equiv rewrite(\varphi, \varphi_{min}) \equiv \{\mathcal{CQ}_0, ..., \mathcal{CQ}_n\} \tag{4.23}$$

Finally, each $\mathcal{CQ}_i$ can be converted in a SQL query φ_{sql}.

$$\varphi_{sql} \equiv toSql(\mathcal{CQ}_i) \tag{4.24}$$

4.3 Multidimensional graph

This section will describe how annotations will be placed into the model described previously in order to transform an integration graph into a multidimensional graph. It will be shown an intuition of how facts, measures, dimension, levels and aggregations can be represented into GFDM by adopting its own model and syntax and how it will be possible to use annotations to perform the join of dimensional data to implement the Roll-Up operation.

4.3.1 Star schema representation

The star schema is a well-known design pattern able to represent a multidimensional data as relational model. The representation adopted to design a multidimensional graph is inspired to the idea of star schema, as shown in Figure 4.2. A concept vertex $V_{\mathcal{G}}$ have been used to define the study fact, which is related by concept to concept relationships to each lower granularity level of each related dimension, defining the dimension name as a label over the edge $E_{\mathcal{G}}$. Let us now describe how a dimension will be represented in GFDM with the help of Figure 4.2. As in the multidimensional cube model, where each dimension is divided in levels, each one associated to an identifier, the same idea is replicated into the multidimensional graph $\mathcal{MG}$. A dimension will be represented as a sequence of connected levels in a multidimensional graph. Each level is represented as a concept vertex $V_{\mathcal{G}}$, and levels are linked via *partOf* relationship as edges $E_{\mathcal{G}}$ describing the different granularity. Then each level as in the star schema is represented with an identifier that will be represented by an identifier feature vertex F^{id}. Figure 4.2 depicts a dimension with three level, as *City*, *Region* and *Country*, each one linked to at least an identifier feature via *hasFeature* relationship, having respectively *city*, *region* and *country*.

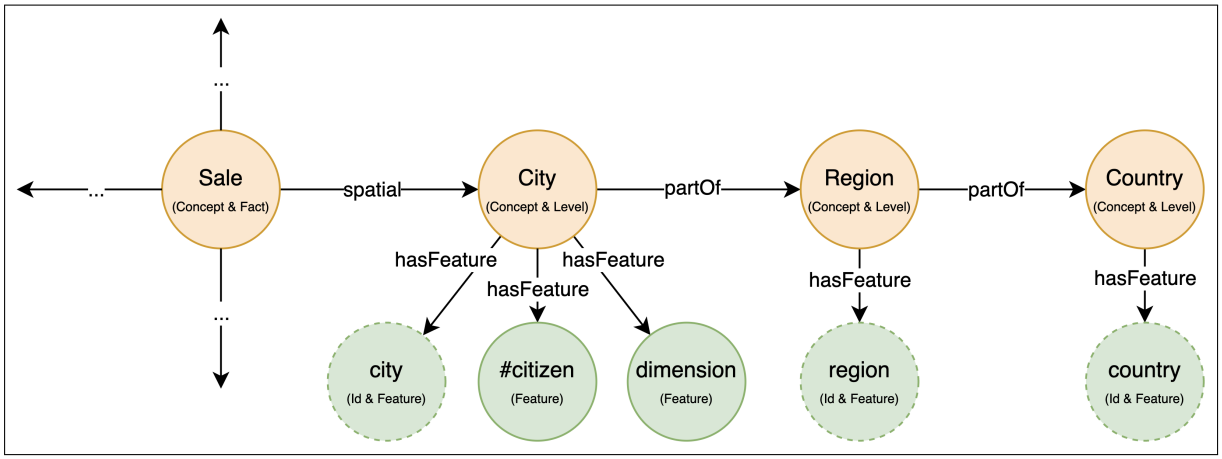


Figure 4.2: Star schema in GFDM.

4.3.2 Aggregation function

Let see now how to place annotation into the integration graph describing aggregation operations. Aggregation functions are very important to annotate an aggregation semantic for a specific feature into a multidimensional graph $\mathcal{MG}$. That feature vertex will then take the role of a measure, since it will have an aggregation semantic. In GFDM aggregations will be represented using the graph syntax, making pointing to a concept vertex $V_{\mathcal{G}}$, with the semantic of an aggregation to a feature vertex using the label aggregates for the edge $E_{\mathcal{G}}$. In Figure 4.3 have been depicted a clear example, having the feature *number_ of_ unit* linked to an aggregating function *sum* and the feature *total_ revenue* linked to both *avg* and *sum* aggregating function. It is very important having the aggregation functions represented into $\mathcal{MG}$ since being the Roll-Up performed implicitly, it is important for the algorithm knowing the semantic of the aggregation.

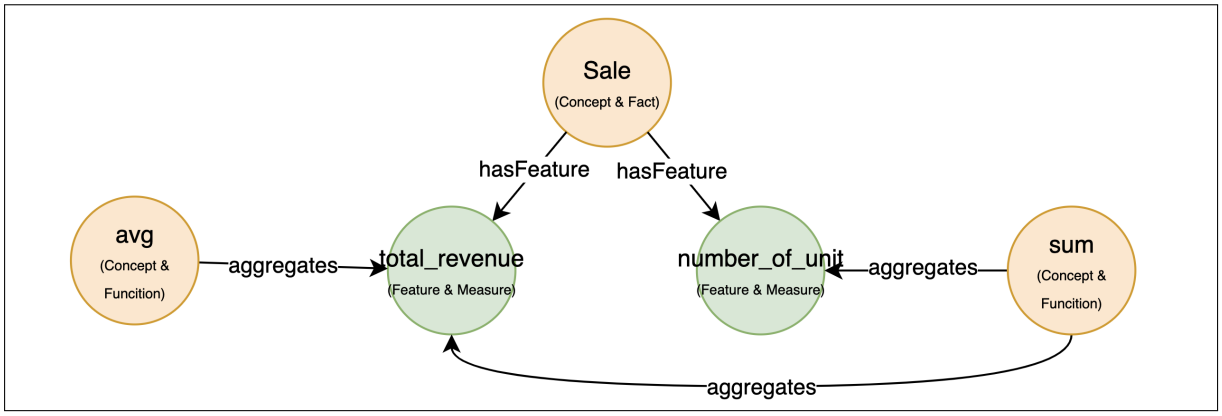


Figure 4.3: Example of aggregation functions *avg* and *sum*.

4.3.3 Join dimensional data within a dimension

In this section, we introduce the intuition of how the join of dimensional data will be executed. As we know this operation is very important for the Roll-Up since it will allow to aggregate data at a lower granularity to the desired higher granularity and this feature can already be achieved in GFDM. To execute the join of dimensional data, it will be exploited the Rewriting Algorithm described in section 4.2, since it will allow executing join operations through different wrapper attributes pointing to the same identifier feature. The idea we got to solve this problem is using look-up tables to perform the join of dimensional data; Look-up tables, that will be represented as wrappers, will lead the join operation within dimensions storing into tables how a level maps to the following higher granularity one. This kind of approach will need a preliminary phase in which all the look-up table wrappers W_{lut} are going to be loaded into the system, or linked to any multidimensional data repository. Figure 4.4 depicts an example of a multidimensional integration graph $\mathcal{I}$, having on the right-hand side $\mathcal{MG}$ describing a dimension with two levels, and on the left-hand side $\mathcal{S}$ that contains wrappers and their attributes. Also, a visual query φ over $\mathcal{MG}$ is represented as a red dashed line. We can see that three wrappers have been depicted in the figure, where *w1* and *w2* are regular data wrapper having a single attribute, respectively *city* and *country*, while the third wrapper *w3* is a look-up table wrapper W_{lut} that will allow joining city to country. In particular, W_{lut} stores the functional dependency of each instance of city, describing how each *city* can be converted into its respective a *country*. Given a query φ that contains two or more consecutive levels, with their respective identifier feature, for a dimension, it will be possible to join all the queried identifier feature if there is a W_{lut} bridging each level identifier to the consecutive one. Let finally perform a *rewrite* operation for φ respecting the minimal property for φ itself as described in equation 4.25.

$$rewrite(\varphi, \varphi) \equiv \{\mathcal{CQ}_1, \mathcal{CQ}_2\} \tag{4.25}$$

$$\mathcal{CQ}_1 \equiv \{mapping(W1), mapping(W3)\} \tag{4.26}$$

$$\mathcal{CQ}_2 \equiv \{mapping(W2), mapping(W3)\} \tag{4.27}$$

Finally, we see each $\mathcal{CQ}$ converted as relational algebra expressions as follows.

$$relationalAlgebra(\mathcal{CQ}_1) \equiv \Pi_{city\_3,region\_3,revenue}(w1 \bowtie_{city\_1=city\_3} w3) \tag{4.28}$$

$$relationalAlgebra(\mathcal{CQ}_2) \equiv \Pi_{city\_3,region\_3,revenue}(w2 \bowtie_{region\_2=region\_3} w3) \tag{4.29}$$

4.3.4 Join with minimal property

The approach illustrated so far has actually a problem in joining dimensional data. The issue is that we are using the join operation expecting to aggregate lower granularity

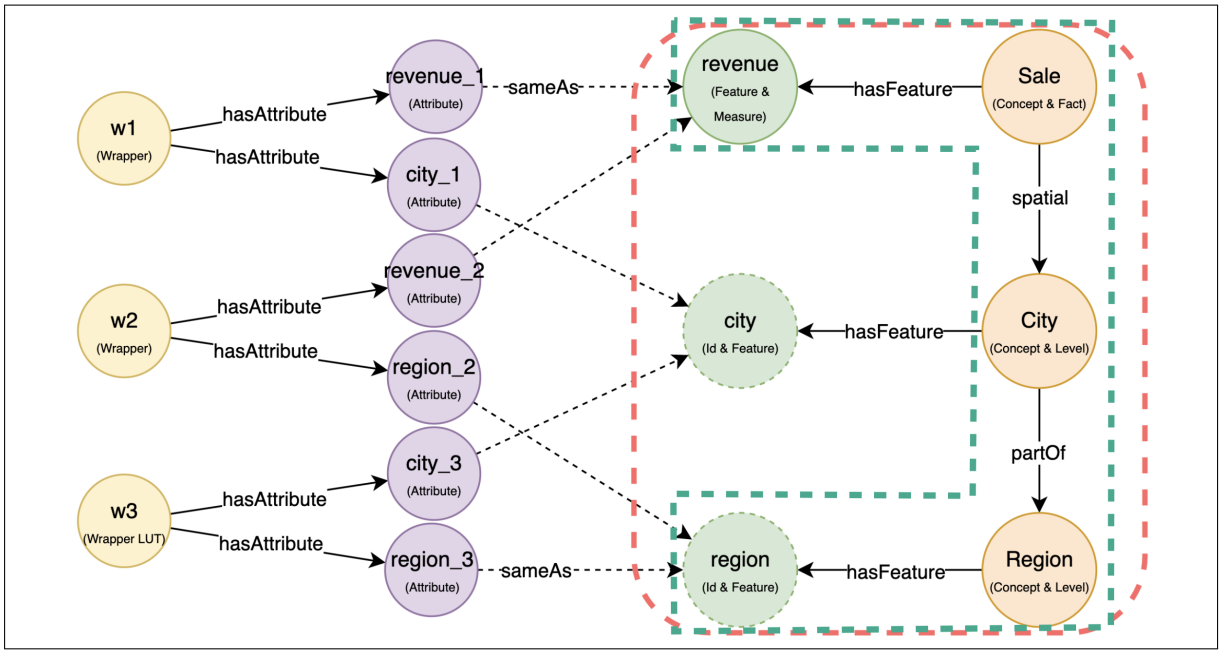


Figure 4.4: How dimensional data are joined within a dimension over GFDM, where the dashed red line is a visual query φ and the dashed green line is a minimal query φ_{min}.

data to higher granularity data but what actually happens is also the opposite operation, having higher granularity levels joined with lower granularity one performing implicit Drill-Down for higher granularity levels. As we can see, equation 4.26 performs a Roll-Up of data aggregating through look-up wrappers lower granularity to higher granularity data, while 4.27 performs the opposite operation doing a Drill-Down aggregating through look-up wrappers higher granularity to lower granularity data. The operation described in equation 4.27 is clearly wrong, being unknown how a levels Drill-Downs. The result of such operation would then generate many record for each granularity level as the possible Drill-Downs producing in consequence incorrect data aggregations. This problem has been solved by adopting the minimal property. Consider φ and φ_{min} in Figure 4.4 with the goal of aggregating the *City* granularity to *Region* granularity, performing joins just in the upward direction (lower to higher granularity). Let now rewrite φ with the minimality for φ_{min} as shown in equation 4.30.

$$rewrite(\varphi, \varphi_{min}) \equiv \{\{mapping(W1), mapping(W3)\}\} \equiv \{\mathcal{CQ}_1\} \tag{4.30}$$

Applying the minimal property the covering mappings $\{mapping(W2), mapping(W3)\}$ have been removed because the combination of wrapper is not minimal for φ_{min} (e.g. if removing $mapping(W3)$, $mapping(W2)$ covers as well φ_{min}). The minimal property will be used in this way to allow the join of dimensional data, preserving correctness of information.

Chapter 5

Description of the method

In this chapter we first describe the model developed, aimed to support the operation of implicit Roll-Up, and after it will follow a detailed explanation of the algorithm that will implement the implicit Roll-Up operation itself. The model will extend the system GFDM defined in chapter 4, and in particular, section 4.3.3 is the starting point of the reasoning behind the implicit Roll-Up algorithm, since it is related to the join of dimensional data.

5.1 Preparation

This section will formalize the Multidimensional Cube model $\mathcal{MC}$ and the Multidimensional Graph model $\mathcal{MG}$, showing and comparing the characteristics of each one. Let us start defining $\mathcal{MC}$ as

$$\mathcal{MC} \equiv \langle \mathcal{F}, \mathcal{D}, \mathcal{L}, \mathcal{M}, levels : \mathcal{D} \to \langle \mathcal{L} \rangle, partOf : \mathcal{L} \to \mathcal{L}' \rangle \tag{5.1}$$

where $\mathcal{F}$ is the described fact, $\mathcal{D}$ are all the multidimensional spaces known as dimensions, $\mathcal{L}$ are all the different granularity levels behaving to $\mathcal{D}$ and $\mathcal{M}$ are all the study measures. Finally, we have $levels$ and $partOf$ that are both functions. The first, for a given $\mathcal{D}$, gets all the levels $\mathcal{L}$ in $\mathcal{D}$ and the second describes the functional dependency of the given $\mathcal{L}$. Let us now introduce the formalization of multidimensional graph model $\mathcal{MG}$ as a concrete implementation of the well known $\mathcal{MC}$ over the graph model proposed in [1].

$$\mathcal{MG} \equiv \langle \mathcal{F}, \mathcal{D}, \mathcal{L}, \mathcal{M}, \mathcal{AF}, levels : \mathcal{D} \to \langle \mathcal{L} \rangle, aggregates : \mathcal{AF} \to \langle \mathcal{M} \rangle, partOf : \mathcal{L} \to \mathcal{L}' \rangle \tag{5.2}$$

The multidimensional graph model considers in addition the aggregating functions $\mathcal{AF}$, each one associated to a set of measures $\mathcal{M}$ by the function $aggregates$. In $\mathcal{MG}$ model aggregating functions needs to be "hard-coded" into the schema in order to perform implicit Roll-Up, since being the operation performed implicitly, it will not be possible to choose it while executing the query. The connections between $\mathcal{G}$ and $\mathcal{MG}$ is done

generating taxonomies between $V_{\mathcal{G}}$ and $E_{\mathcal{G}}$, and $V_{\mathcal{MG}}$ and $E_{\mathcal{MG}}$, and the relationships are the following;

$$\mathcal{L} \wedge \mathcal{AF} \sqsubseteq C \sqsubseteq V_{\mathcal{MG}} \tag{5.3}$$

$$\mathcal{M} \sqsubseteq F \sqsubseteq V_{\mathcal{MG}} \tag{5.4}$$

$$partOf \wedge aggregates \sqsubseteq E_{\mathcal{MG}} \tag{5.5}$$

From now on it will be introduced the triple notation behind each element of $\mathcal{MG}$ relying on the triple syntax introduced in section 4.1 and Figure 5.1 will be used as an example.

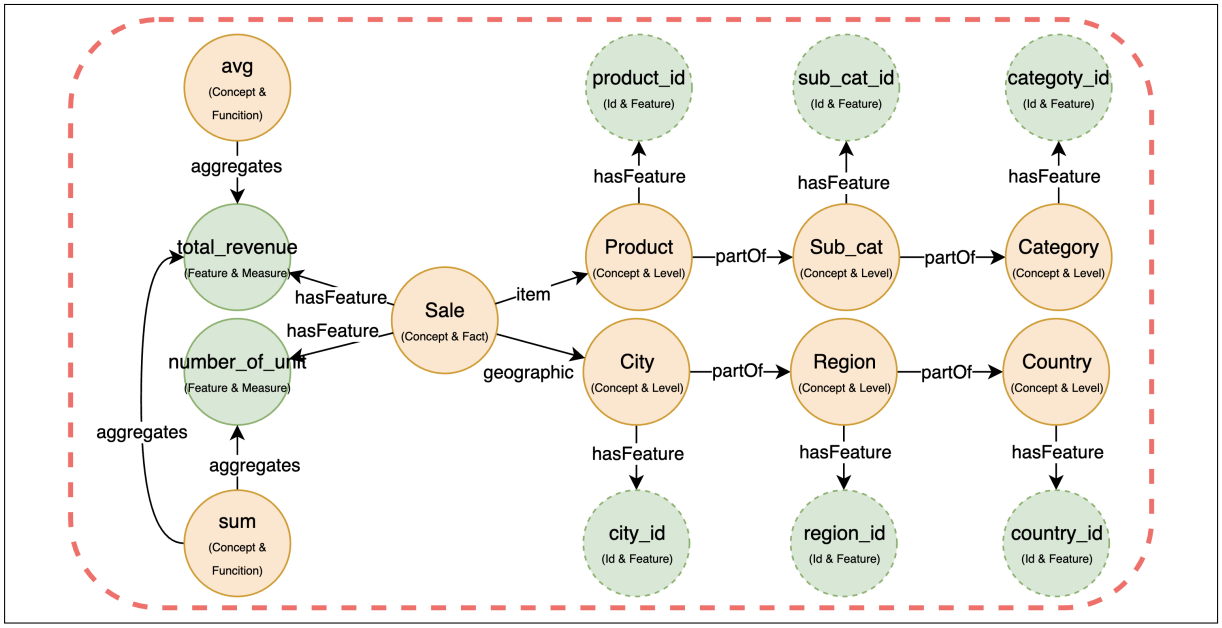


Figure 5.1: Running example, where the dashed red line is the query φ triggered.

We now introduce the triple notation for $\mathcal{MG}$ starting defining the triples involved in a dimension $\mathcal{D}$. Example 1 also shows an example of dimension triples in the running example for the *geographic* dimension. Consider $levels(\mathcal{D})$ as a totally-ordered set describing all the levels $\mathcal{L}_i$ in the given $\mathcal{D}$, ordered by lower level granularity lo higher level granularity, formally defined by equation 5.6.

$$\langle \mathcal{L} \rangle \equiv levels(\mathcal{D}) \equiv \{\mathcal{L}_0, ..., \mathcal{L}_n\} \tag{5.6}$$

Equation 5.7 describes the triples that associate each level $\mathcal{L}_i$ to its identifier feature F^{id} as the union of all the triples in the form $\langle \mathcal{L}_i, hasFeature, id(\mathcal{L}_i) \rangle$, where $\mathcal{L}_i$ is a level in $levels(\mathcal{D})$ and $id(\mathcal{L}_i)$ is its identifier feature. Equation 5.8 represents all the triples that describes the functional dependencies as the union of all the triples in the form $\langle \mathcal{L}_i, partOf, \mathcal{L}_{i+1} \rangle$, where $\mathcal{L}_i$ is a level in $levels(\mathcal{D})$ and $\mathcal{L}_{i+1}$ is its right higher granularity

level. Formally, the triples in $\mathcal{D}$ are then given by the union of all the triples describing the functional dependencies in $\mathcal{D}$ and all the tripled describing the identifier features for each $\mathcal{L}$ in $\mathcal{D}$, as showed in equation 5.9.

$$\mathcal{L} \rightarrow F^{id}_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{n} \langle \mathcal{L}_i, hasFeature, id(\mathcal{L}_i)\rangle \tag{5.7}$$

$$\mathcal{L} \rightarrow \mathcal{L}^{'}_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{n-1} \langle \mathcal{L}_i, partOf, \mathcal{L}_{i+1}\rangle \tag{5.8}$$

$$\mathcal{D}_{\langle S,P,O\rangle} \equiv \mathcal{L} \rightarrow \mathcal{L}^{'}_{\langle S,P,O\rangle} \cup \mathcal{L} \rightarrow F^{id}_{\langle S,P,O\rangle} \tag{5.9}$$

Let us introduce now the triple notation for a fact $\mathcal{F}$ and an example of its triple notation can be found in Example 2. Consider $dimensions(\mathcal{F})$ as all the dimension $\mathcal{D}_i$ in $\mathcal{F}$ and $measures(\mathcal{F})$ as all the measures $\mathcal{M}_i$ in $\mathcal{F}$.

$$\langle \mathcal{D}\rangle \equiv dimensions(\mathcal{F}) \equiv \{\mathcal{D}_0, ..., \mathcal{D}_n\} \tag{5.10}$$

$$\langle \mathcal{M}\rangle \equiv measures(\mathcal{F}) \equiv \{\mathcal{M}_0, ..., \mathcal{M}_m\} \tag{5.11}$$

Let now consider all those triples linking the fact to all the dimensions $\mathcal{D}$ and measures $\mathcal{M}$ in $\mathcal{F}$, having respectively $\langle \mathcal{F}, dimensionName, lowerGranularityLevel(\mathcal{D}_i)\rangle$ where the fact is linked to the lower granularity level of the ith dimension and $\langle \mathcal{F}, hasFeature, \mathcal{M}_i\rangle$ where the fact is linked to the ith measure. Equation 5.12 describes the union of all the triples linking $\mathcal{F}$ to each $lowerGranularityLevel(\mathcal{D}_i)$ with $\mathcal{D}_i$ in $dimensions(\mathcal{F})$ with a $dimensionName$ relationship, and equation 5.13 describes then the union of all the triples linking $\mathcal{F}$ to each $\mathcal{M}_i$ in $measures(\mathcal{F})$ with a $hasFeature$ relationship. Then the triple notation of $\mathcal{F}$, as defined in equation 5.14, is given by the union of all the dimensions in $\mathcal{F}$, all the labels linking the fact to the dimensions and to all the measures.

$$\mathcal{F} \rightarrow \mathcal{L}_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{n} \langle \mathcal{F}, dimensionName, lowerGranularityLevel(\mathcal{D}_i)\rangle \tag{5.12}$$

$$\mathcal{F} \rightarrow \mathcal{M}_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{m} \langle \mathcal{F}, hasFeature, \mathcal{M}_i\rangle \tag{5.13}$$

$$\mathcal{F}_{\langle S,P,O\rangle} \equiv \langle \mathcal{D}\rangle_{\langle S,P,O\rangle} \cup \mathcal{F} \rightarrow \mathcal{L}_{\langle S,P,O\rangle} \cup \mathcal{F} \rightarrow \mathcal{M}_{\langle S,P,O\rangle} \tag{5.14}$$

Let us finally see the triple notation for an aggregating function $\mathcal{AF}$ that showed in Example 3. Consider $functions(\mathcal{MG})$ as all the aggregating function $\mathcal{AF}_i$ in $\mathcal{MG}$, given by Equation 5.15 and $aggregates(\mathcal{AF}_i)$ as all the measures $\mathcal{M}_{(i,j)}$ associated to the aggregating function $\mathcal{AF}_i$, given by Equation 5.16. Consider an aggregating function $\mathcal{AF}$ can be linked to many measures $\mathcal{M}$.

$$\langle \mathcal{AF}\rangle \equiv functions(\mathcal{MG}) \equiv \{\mathcal{AF}_0, ..., \mathcal{AF}_n\} \tag{5.15}$$

$$\langle \mathcal{M}\rangle \equiv aggregates(\mathcal{AF}_i) \equiv \{\mathcal{M}_{(i,0)}, ..., \mathcal{M}_{(i,m)}\} \tag{5.16}$$

The triples describing an aggregating function are in the form $\langle \mathcal{AF}_i, aggregates, \mathcal{M}_{(i,j)}\rangle$ where $\mathcal{AF}_i$ is the given aggregation function in $functions(\mathcal{MG})$ and $\mathcal{M}_{(i,j)}$ is the jth

measure in $aggregates(\mathcal{AF}_i)$. Equation 5.17 describes the union of all the triples for $\mathcal{AF}_i$ and each measure $\mathcal{M}_{(i,j)}$ and equation 5.18 describes the union of all the triples of each $\mathcal{AF}_i$ in $aggregates(\mathcal{AF}_i)$ associated to each $\mathcal{M}_{(i,j)}$.

$$\mathcal{AF}_{i\langle S,P,O\rangle} \equiv \cup_{j=0}^{m} \langle \mathcal{AF}_i, aggregates, \mathcal{M}_{(i,j)} \rangle \tag{5.17}$$

$$\mathcal{AF}_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{n} \mathcal{AF}_{i\langle S,P,O\rangle} \tag{5.18}$$

Let us finally formalize the triple notation of $\mathcal{MG}$ as the union of the triples of each fact $\langle \mathcal{F} \rangle$, the triples of each aggregation functions $\langle \mathcal{AF} \rangle$ and the triples of $\mathcal{G}$, as shown in Equation 5.19. Follows the description of taxonomies for the elements described above in Equation 5.20 and 5.21.

$$\mathcal{MG}_{\langle S,P,O\rangle} \equiv \mathcal{F}_{\langle S,P,O\rangle} \cup \mathcal{G}_{\langle S,P,O\rangle} \cup \mathcal{AF}_{\langle S,P,O\rangle} \tag{5.19}$$

$$\mathcal{G}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{MG}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{I}_{\langle S,P,O\rangle} \tag{5.20}$$

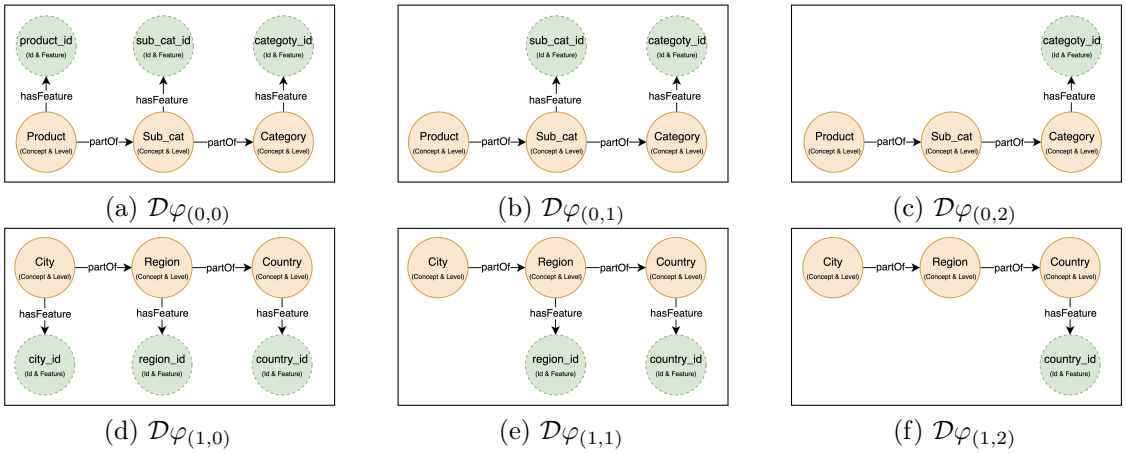
$$\mathcal{D}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{F}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{MG}_{\langle S,P,O\rangle} \tag{5.21}$$

Example 1. The triples representing the *geographic* dimension in Figure 5.1 are $\langle City, partOf, Region \rangle$ and $\langle Region, partOf, Country \rangle$ that describe the functional dependencies and then $\langle City, hasFeature, city\_id \rangle$, $\langle Region, hasFeature, region\_id \rangle$ and $\langle Country, hasFeature, country\_id \rangle$ describing the identifier features for each level.

Example 2. The triples describing the Sale fact in Figure 5.1 are given by a union of all the triples of the *item* and *geographic* dimension with also the triples $\langle Sale, item, Product \rangle$ and $\langle Sale, geographic, City \rangle$ that link the fact to each dimension and also the triples $\langle Sale, hasFeature, total\_revenue \rangle$ and $\langle Sale, hasFeature, number\_of\_unit \rangle$ that link the fact to each measure.

Example 3. $\langle sum, aggregates, number\_of\_unit \rangle$, $\langle sum, aggregates, total\_revenue \rangle$ and $\langle avg, aggregates, total\_revenue \rangle$ describe the aggregation functions in Figure 5.1.

Let us consider now the querying for $\mathcal{MG}$, where φ is a pattern matching or visual query over $\mathcal{MG}$. Introduce also the concept of dimension query $\mathcal{D}\varphi$ and $\mathcal{R}\varphi$. Example 4 shows a complete example of the rationale that will be explained from now on, describing what dimension queries and Roll-Up queries are and how they are generated. A dimension query is a sub-graph of a given $\mathcal{D}$ in φ and its purpose is to allow joining dimensional data at low level granularity aggregating them to the required higher granularity asked by φ through W_{lut} (method described in section 4.3.3). Let us take as example the dimension queries for the *geographic* dimension in Figure 5.2 (image (d), (e) and (f)). The queried granularity in φ in the running example is *Country* since it is the higher granularity level that contains the identifier. Then what we would like to achieve is to include in the query result all the data at *Country* granularity provided by $\mathcal{D}\varphi_{(1,2)}$, the data at *Region*



(a) $\mathcal{D}\varphi_{(0,0)}$ (b) $\mathcal{D}\varphi_{(0,1)}$ (c) $\mathcal{D}\varphi_{(0,2)}$

(d) $\mathcal{D}\varphi_{(1,0)}$ (e) $\mathcal{D}\varphi_{(1,1)}$ (f) $\mathcal{D}\varphi_{(1,2)}$

Figure 5.2: Dimension queries $\mathcal{D}\varphi$ for *item* and *geographic* dimension.

granularity joined with the relative *Country* provided by $\mathcal{D}\varphi_{(1,1)}$ and also the data at *City* granularity joined firstly with *Region* and finally with *Country* provided by $\mathcal{D}\varphi_{(1,0)}$. Let now formalize the definition of dimension query $\mathcal{D}\varphi$. Let $dimensions(\varphi)$ be a set of dimensions $\mathcal{D}_i$ in φ as shown in Equation 5.22 and consider also Equation 5.23 as all the levels $\mathcal{L}_{(i,j)}$ in $\mathcal{D}_i$.

$$\langle \mathcal{D} \rangle \equiv dimensions(\varphi) \equiv \{\mathcal{D}_0, ..., \mathcal{D}_n\} \tag{5.22}$$

$$\langle \mathcal{L} \rangle \equiv levels(\mathcal{D}_i) \equiv \{\mathcal{L}_{(i,0)}, ..., \mathcal{L}_{(i,m)}\} \tag{5.23}$$

Let then formalize with Equation 5.24 the triples for a dimension query $\mathcal{D}\varphi_{(i,j)}$ associated to the ith dimension $\mathcal{D}_i$ and the jth level $\mathcal{L}_{(i,j)}$ in $\mathcal{D}_i$. A dimension query $\mathcal{D}\varphi_{(i,j)}$ associated to $\mathcal{D}_i$ and $\mathcal{L}_{(i,j)}$ is calculated as the triple in $\mathcal{D}_i$ minus the union of all the triples in the form $\langle \mathcal{L}_{(i,j)}, hasFeature, id(\mathcal{L}_{(i,j)}) \rangle$ representing all the identifier features of all the level with smaller granularity then $\mathcal{L}_{(i,j)}$.

$$\mathcal{D}\varphi_{(i,j)\langle S,P,O\rangle} \equiv \mathcal{D}_{i\langle S,P,O\rangle} - \{\cup_{z=0}^{j} \langle \mathcal{L}_{(i,z-1)}, hasFeature, id(\mathcal{L}_{(i,z-1)}) \rangle\} \tag{5.24}$$

Once given the triple notation of a dimension query for a fixed $\mathcal{D}_i$ and $\mathcal{L}_{(i,j)}$, now it is possible to calculate the union set of all the dimension queries for a dimension $\mathcal{D}_i$ as the union of all the dimension queries for each $\mathcal{L}_{(i,j)}$ in $\mathcal{D}_i$ as described in Equation 5.25. It is also possible to define the union set of all the dimension queries in φ as the union of all the dimension queries calculated above with Equation 5.25 for each dimension $\mathcal{D}_i$ in $dimensions(\varphi)$ as shown in Equation 5.26.

$$\mathcal{D}\varphi_i \equiv \cup_{j=0}^{m} \mathcal{D}\varphi_{(i,j)} \equiv \{\mathcal{D}\varphi_{(i,0)}, ..., \mathcal{D}\varphi_{(i,m)}\} \tag{5.25}$$

$$\mathcal{D}\varphi \equiv \cup_{i=0}^{n} \mathcal{D}\varphi_i \equiv \{\{\mathcal{D}\varphi_{(0,0)}, ..., \mathcal{D}\varphi_{(0,m)}\}, ..., \{\mathcal{D}\varphi_{(n,0)}, ..., \mathcal{D}\varphi_{(n,m)}\}\} \tag{5.26}$$

Then we will have that the number of dimension queries for the ith dimension $\mathcal{D}_i$ in $dimensions(\varphi)$ is given by the number of levels in $\mathcal{D}_i$, as described in Equation 5.27.

Finally, Equation 5.28 describes the total number of dimension query for φ as the sum of the number of dimension query for each $\mathcal{D}_i$ in $dimensions(\varphi)$.

$$cardinality(\mathcal{D}\varphi_i) \equiv size(levels(\mathcal{D}_i)) \tag{5.27}$$

$$cardinality(\mathcal{D}\varphi) \equiv \sum_{i=0}^{n} cardinality(\mathcal{D}\varphi_i) \tag{5.28}$$

A Roll-Up query $\mathcal{R}\varphi$ is a sub-graph of φ and it is a query that has as dimensions one possible combination of the dimension queries (e.g. one entry of the Cartesian Product of dimension queries $\mathcal{D}\varphi_i$). There will be generated one Roll-Up query for each possible dimensional wrapper, and each query will then be used to get the data associated to just one wrapper configuration. Figure 5.3 depicts all the Roll-Up queries generated for the running example and Figure 5.4 depicts which wrapper is covered by each Roll-Up query, assuming the existence of the look-up tables W_{lut} (Figure 5.3(j)). Equation 5.29 computes the Cartesian Product set between each set of dimension query $\mathcal{D}\varphi_i$, each one associated to a dimension $\mathcal{D}_i$.

$$\times\mathcal{D}\varphi \equiv \{\mathcal{D}\varphi_0 \times ... \times \mathcal{D}\varphi_n\} \tag{5.29}$$

Then it will be possible to calculate the triples of a Roll-Up query of a single entry of the Cartesian Product set associated to $\times\mathcal{D}\varphi_i$ as described in Equation 5.30. The formula consists of making the union of all the dimension queries triples in $\times\mathcal{D}\varphi_i$ and the factual data that is given removing all the dimensions to φ, that will actually be replaced by the triples of a combination of dimension query. Afterwards it will be also possible to calculate all the triples of each Roll-Up query as the union set of each $\mathcal{R}\varphi_i$ as described in Equation 5.31.

$$\mathcal{R}\varphi_{i\langle S,P,O\rangle} \equiv \{\cup_{j=0}^{size(\mathcal{R}\varphi_i)-1} \times \mathcal{D}\varphi_{(i,j)\langle S,P,O\rangle}\} \cup \{\varphi - dimensions(\varphi)\} \tag{5.30}$$

$$\langle\mathcal{R}\varphi\rangle_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{n} \mathcal{R}\varphi_{i\langle S,P,O\rangle} \tag{5.31}$$

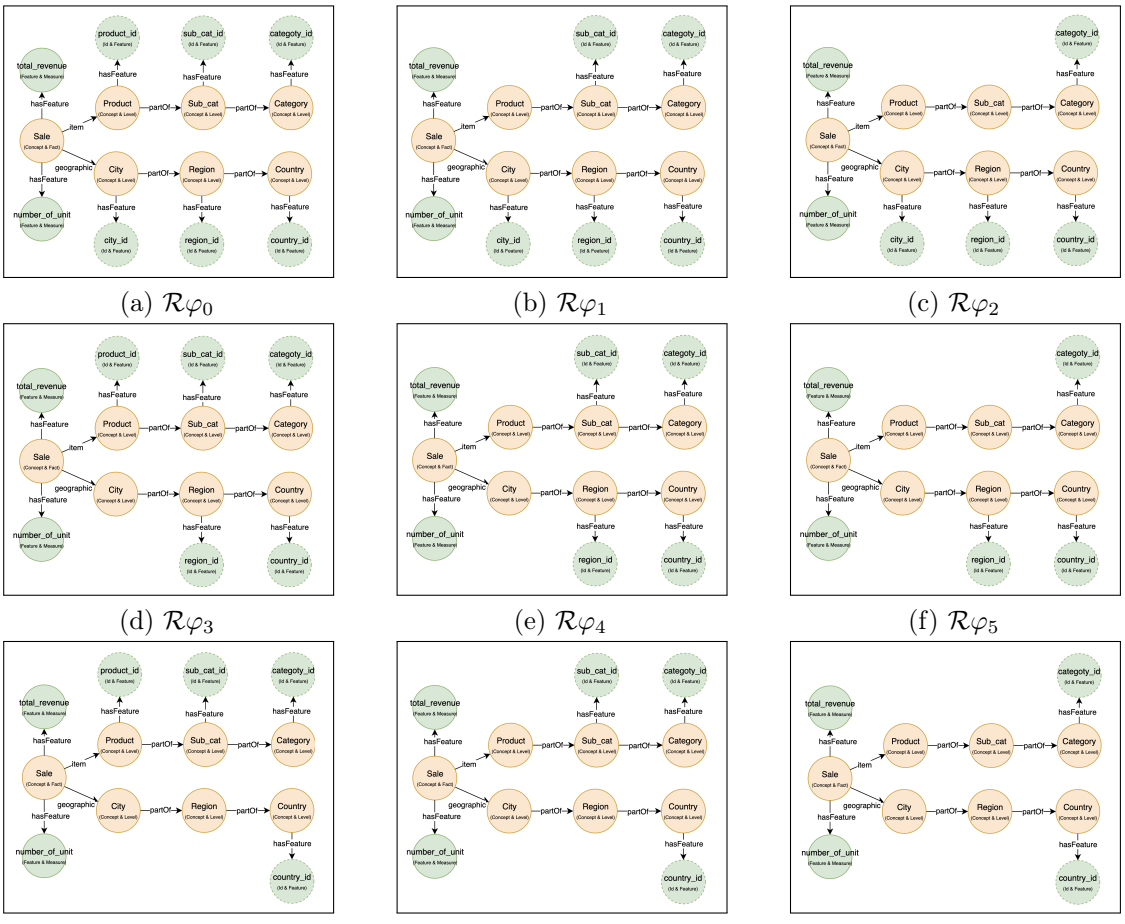
Then, referring to Equation 5.22 describing all the $\mathcal{D}_i$ in φ we have that the number of Roll-Up queries is given by the multiplication of all the number of dimension queries in each $\mathcal{D}_i$ (given by Equation 5.27). The number of the Roll-Up queries for φ is given by Equation 5.32.

$$cardinality(\mathcal{R}\varphi) \equiv \prod_{i=0}^{n} cardinality(\mathcal{D}\varphi_i) \tag{5.32}$$

Taxonomies in Equation 5.33 and 5.34 describe the relationships between queries.

$$\mathcal{D}\varphi_{\langle S,P,O\rangle} \sqsubseteq \mathcal{D}_{\langle S,P,O\rangle} \tag{5.33}$$

$$\mathcal{D}\varphi_{\langle S,P,O\rangle} \sqsubseteq \mathcal{R}\varphi_{\langle S,P,O\rangle} \sqsubseteq \varphi_{\langle S,P,O\rangle} \sqsubseteq \mathcal{G}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{MG}_{\langle S,P,O\rangle} \tag{5.34}$$

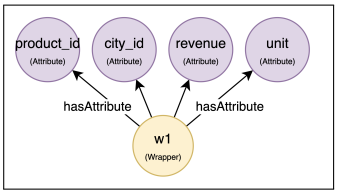


(a) $\mathcal{R}\varphi_0$ (b) $\mathcal{R}\varphi_1$ (c) $\mathcal{R}\varphi_2$

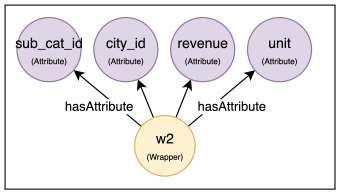
(d) $\mathcal{R}\varphi_3$ (e) $\mathcal{R}\varphi_4$ (f) $\mathcal{R}\varphi_5$

(g) $\mathcal{R}\varphi_6$ (h) $\mathcal{R}\varphi_7$ (i) $\mathcal{R}\varphi_8$

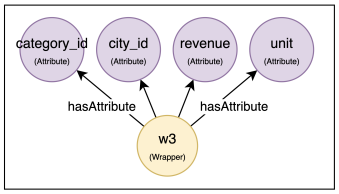
Figure 5.3: Roll-Up queries $\mathcal{R}\varphi$ for *item* and *geographic* dimension.



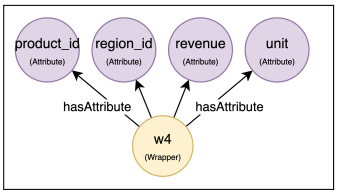
(a) Wrapper covered by $\mathcal{R}\varphi_0$



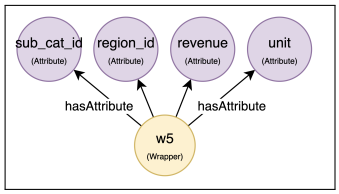
(b) Wrapper covered by $\mathcal{R}\varphi_1$



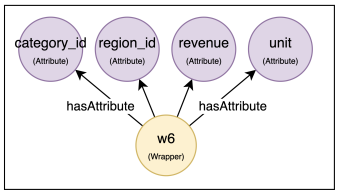
(c) Wrapper covered by $\mathcal{R}\varphi_2$



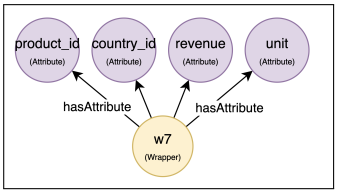
(d) Wrapper covered by $\mathcal{R}\varphi_3$



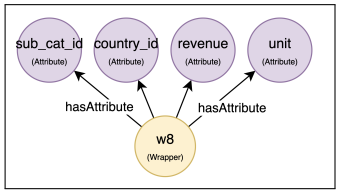
(e) Wrapper covered by $\mathcal{R}\varphi_4$



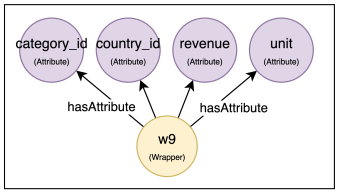
(f) Wrapper covered by $\mathcal{R}\varphi_5$



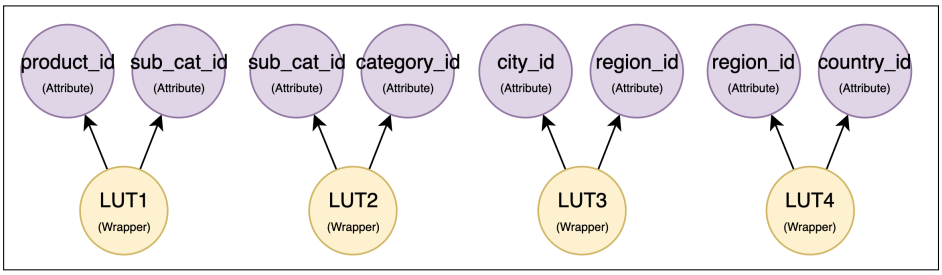
(g) Wrapper covered by $\mathcal{R}\varphi_6$



(h) Wrapper covered by $\mathcal{R}\varphi_7$



(i) Wrapper covered by $\mathcal{R}\varphi_8$



(j) Look-up table wrappers W_{lut}

Figure 5.4: Which $\mathcal{R}\varphi_i$ covers which wrapper.

Let us finally introduce some functions that are traverses over $\mathcal{MG}$ that will be useful in the next section for the execution of the algorithm; $measures(\varphi)$, $features(\varphi)$, $levels(\varphi)$, $dimensions(\varphi)$ are respectively all the measures $\mathcal{M}$, features F, levels $\mathcal{L}$ and dimensions $\mathcal{D}$ in φ. Then $id(C^+)$ is the identifier feature F^{id} for a given C or $\mathcal{L}$. The functions $lowerGranularityLevel(\mathcal{D})$ and $higherGranularityLevel(\mathcal{D})$ as respectively the lower granularity $\mathcal{L}$ and the higher granularity $\mathcal{L}$ of a given dimension $\mathcal{D}$, holding the relationship $\langle \mathcal{L}, hasFeature, id(\mathcal{L}) \rangle$ (e.g. the higher or lower granularity level should be also linked to an F^{id}). Regarding the generation of dimension queries and Roll-Up queries we have that $dimensionQueries(\mathcal{D}_i)$ gets all the dimension queries $\langle \mathcal{D}\varphi \rangle$ in $\mathcal{D}_i$ (as equation 5.25), $cartesianProduct(\langle\langle \mathcal{D}\varphi \rangle\rangle)$ that calculates the Cartesian product of all the dimension queries as described in equation 5.29 and $rollUpQuery(\varphi, \langle \mathcal{D}\varphi \rangle_i)$ that generates the Roll-Up query for the dimension query $\langle \mathcal{D}\varphi \rangle_i$ as described in equation 5.30.

Example 4. Taking as example Figure 5.2, let show how the Roll-Up queries are generated. Let us start considering all the triples in $\mathcal{D}_1$, as $\langle City, partOf, Region \rangle$, $\langle Region, partOf, Country \rangle$, $\langle City, hasFeature, city\_id \rangle$, $\langle Region, hasFeature, region\_cat\_id \rangle$ and $\langle Country, hasFeature, country\_id \rangle$. Generate now the dimension queries $\mathcal{D}\varphi$ for $\mathcal{D}_1$. It will be generated a dimension query $\mathcal{D}\varphi_{(1,i)}$ for each level $\mathcal{L}_{(1,i)}$ in $\mathcal{D}_1$ as shown in equation 5.24. For $\mathcal{L}_0$ (*City* granularity) any triple will be removed since it is already the finest granularity level having $\mathcal{D}\varphi_{(1,0)\langle S,P,O \rangle} \equiv \mathcal{D}_{1\langle S,P,O \rangle}$. For $\mathcal{L}_1$ (*Region* granularity) the triple $\langle City, hasFeature, city\_id \rangle$ will have to be removed to $\mathcal{D}_1$ having $\mathcal{D}\varphi_{(1,1)\langle S,P,O \rangle} \equiv \mathcal{D}_{1\langle S,P,O \rangle} - \langle City, hasFeature, city\_id \rangle$. Finally, for $\mathcal{L}_2$ (*Country* granularity) the triples $\langle City, hasFeature, city\_id \rangle$ and $\langle Region, hasFeature, region\_id \rangle$ will have to be removed to $\mathcal{D}_1$ then we have $\mathcal{D}\varphi_{(1,2)\langle S,P,O \rangle} \equiv \mathcal{D}_{1\langle S,P,O \rangle} - \langle City, hasFeature, city\_id \rangle - \langle Region, hasFeature, region\_cat\_id \rangle$. Then for equation 5.25 $\mathcal{D}_1$ will have the dimension queries $\mathcal{D}\varphi_1 \equiv \{\mathcal{D}\varphi_{(1,0)}, \mathcal{D}\varphi_{(1,1)}, \mathcal{D}\varphi_{(1,1)}\}$, and finally for equation 5.26 the set of dimension queries, each dimension in φ is $\mathcal{D}\varphi \equiv \{\{\mathcal{D}\varphi_{(0,0)}, \mathcal{D}\varphi_{(0,1)}, \mathcal{D}\varphi_{(0,2)}\}, \{\mathcal{D}\varphi_{(1,0)}, \mathcal{D}\varphi_{(1,1)}, \mathcal{D}\varphi_{(1,2)}\}\}$. Compute now the Cartesian Product within the dimension queries set according to equation 5.29 as $\times\mathcal{D}\varphi \equiv \{\mathcal{D}\varphi_{(0,0)}, \mathcal{D}\varphi_{(0,1)}, \mathcal{D}\varphi_{(0,2)}\} \times \{\mathcal{D}\varphi_{(1,0)}, \mathcal{D}\varphi_{(1,1)}, \mathcal{D}\varphi_{(1,2)}\}$ giving as result $\times\mathcal{D}\varphi \equiv \{\{\mathcal{D}\varphi_{(0,0)}, \mathcal{D}\varphi_{(1,0)}\}, \{\mathcal{D}\varphi_{(0,0)}, \mathcal{D}\varphi_{(1,1)}\}, \{\mathcal{D}\varphi_{(0,0)}, \mathcal{D}\varphi_{(1,2)}\}, \{\mathcal{D}\varphi_{(0,1)}, \mathcal{D}\varphi_{(1,0)}\}, \{\mathcal{D}\varphi_{(0,1)}, \mathcal{D}\varphi_{(1,1)}\}, \{\mathcal{D}\varphi_{(0,1)}, \mathcal{D}\varphi_{(1,2)}\}, \{\mathcal{D}\varphi_{(0,2)}, \mathcal{D}\varphi_{(1,0)}\}, \{\mathcal{D}\varphi_{(0,2)}, \mathcal{D}\varphi_{(1,1)}\}, \{\mathcal{D}\varphi_{(0,2)}, \mathcal{D}\varphi_{(1,2)}\}\}$ Let now generate the Roll-Up queries, that are also represented as example in Figure 5.3, starting from $\mathcal{R}\varphi_0$ associated to the first entry $\{\mathcal{D}\varphi_{(0,0)}, \mathcal{D}\varphi_{(1,0)}\}$ in $\times\mathcal{D}\varphi$ as described in equation 5.30, having $\mathcal{R}\varphi_{0\langle S,P,O \rangle} \equiv \mathcal{D}\varphi_{(0,0)\langle S,P,O \rangle} \cup \mathcal{D}\varphi_{(1,0)\langle S,P,O \rangle} \cup \{\varphi - dimensions(\varphi)\}$ where $\varphi - dimensions(\varphi)$ are $\langle Sale, hasFeature, total\_revenue \rangle$, $\langle Sale, geographic, City \rangle$, $\langle Sale, item, Product \rangle$ and $\langle Sale, hasFeature, number\_of\_unit \rangle$. To compute the entire set of Roll-Up queries, the previous operation needs to be executed for each entry in the Cartesian Product set, as defined by equation 5.31.

5.2 The Implicit Roll-Up algorithm

In this section, it will follow up a detailed explanation of the algorithm developed to perform implicit Roll-Up. What we would like to achieve with implicit Roll-Up algorithm is: Given a query φ, understanding what kind of data granularity is asking for and present as result an aggregation of the information provided from wrappers at the queried granularity and also an aggregation of data provided by wrappers at lower granularity. The strategy adopted to achieve such goals is based the idea of dimension queries and Roll-Up queries, generating and executing each possible combination of dimensional query covering any possible wrapper and allowing them to join the required dimensional data, as shown in Figure 4.4. Then we will have that each Roll-Up query will retrieve the aggregated data of just one possible wrapper configuration. The main flow *implicitRollUp* has been shown in Algorithm 1, where its main steps are depicted. The algorithm will take as input the multidimensional graph $\mathcal{MG}$, the global query φ and the overall semantic of the explicit final aggregation $\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle'$. As output, it will generate a SQL query φ_{sql}.

Algorithm 1 implicitRollUp

Input: $\mathcal{MG}, \varphi, \langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle'$
Output: φ_{sql}

1: **if** $canAggregate(\varphi)$ **then**
2: $\quad \langle \mathcal{L} \rangle \leftarrow extractGroupByClauses(\varphi)$
3: $\quad \langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle \leftarrow extractAggregationClauses(\mathcal{MG}, \varphi)$
4: $\quad GB \leftarrow parseGroupByClauses(\langle \mathcal{L} \rangle)$
5: $\quad AGG \leftarrow parseAggregationClauses(\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle)$
6: $\quad AGG' \leftarrow parseAggregationClauses(\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle_{'})$
7: $\quad \langle \mathcal{R}\varphi \rangle \leftarrow generateRollUpQueries(\varphi)$
8: $\quad \langle \varphi_{sql} \rangle \leftarrow rewriteAll(\varphi, \langle \mathcal{R}\varphi \rangle)$
9: $\quad \varphi_{sql} \leftarrow makeSqlQuery(\langle \varphi_{sql} \rangle, GB, AGG, AGG')$
10: $\quad$ **return** φ_{sql}
11: **else**
12: $\quad \varphi_{sql} \leftarrow toSql(rewrite(\varphi, \varphi))$
13: $\quad$ **return** φ_{sql}
14: **end if**

canAggregate

The procedure *canAggregate*, illustrated in the Algorithm 2, will be used to detect all the scenarios in which it will be possible or not to execute the Implicit Roll-Up algorithm; If it will be possible to extrapolate an aggregation semantic from φ then it will be possible to execute Implicit Roll-Up as well, otherwise not. The input of this procedure is the source

query φ, while the output will be a *Boolean* expression that will be *true* if it is possible to aggregate φ otherwise $false$. The criteria to understand if φ has an aggregation semantic are the following:

- The query should cover at least a dimension $\mathcal{D}$ (e.g. there should be at least a group by clause). The procedure is described in Algorithm 3.
- The query should cover at least a measure $\mathcal{M}$.
- The query should not cover any other features rather than measures and levels identifiers.

Algorithm 2 canAggregate

Input: φ
Output: *Boolean*

1: **return** $extractGroupByClauses(\varphi) \neq \emptyset \wedge measures(\varphi) \neq \emptyset \wedge (features(\varphi) - id(levels(dimensions(\varphi))) - measures(\varphi)) \equiv \emptyset$

clauses extraction

In order to extract the SQL group by semantic, after understanding if it is possible to have implicit Roll-Up or not, it will be executed two algorithms over the global query φ; Algorithm 3 will be executed to get the group-by clauses of the final query and Algorithm 4 will get the measures $\mathcal{M}$ and the function $\mathcal{AF}$ that will be used to aggregate $\mathcal{M}$ itself. Algorithm 3, *extractGroupByClauses*, will be used to understand which are the group-by clauses in φ given as input, finding for each $\mathcal{D}_i$ in $dimensions(\varphi)$ which level is the higher granularity one that has also F^{id} (e.g. holds the triple $\langle \mathcal{L}, hasFeature, id(\mathcal{L}) \rangle$) using the traverse $higherGranularityLevel(\mathcal{D}_i)$. As output, it will be given a set of levels $\langle \mathcal{L} \rangle$ and each level name will be a group-by clause. This algorithm is strictly associated to the procedure *parseGroupByClauses* that will allow making in a correct string format (comma separated) all the group-by clauses in a way that can perfectly fit both into the "SELECT" and "GROUP BY" statements of a SQL query. Algorithm 4, *extractAggregationClauses* will be used as opposite to extract the aggregation semantic for the queried measures $\mathcal{M}$. As input, it will be given the global query φ and the multidimensional graph $\mathcal{MG}$ and as output will be given a set of tuples, having an $\mathcal{AF}$ each one associated to a set of $\mathcal{M}$ behaving to φ that the respective aggregating function will aggregate. The output will be calculated firstly finding all the aggregation function $\mathcal{AF}_i$ in $functions(\mathcal{MG})$ and then intersecting all the measure behaving to the relationship $aggregates(\mathcal{AF}_i)$ to all the queried measures and keeping the solution having a non-empty set of measures. For algorithm 4 there is a parsing function as well,

Algorithm 3 extractGroupByClauses

Input: φ
Output: $\langle \mathcal{L} \rangle$

1: $\langle \mathcal{L} \rangle \leftarrow \emptyset$
2: **for each** $\mathcal{D}_i \in dimensions(\varphi)$ **do**
3: $\quad \langle \mathcal{L} \rangle_i \leftarrow higherGranularityLevel(\mathcal{D}_i)$
4: **end for**
5: **return** $\langle \mathcal{L} \rangle$

parseAggregationClauses that will generate a formatted string (comma separated) for the aggregation part of the SQL query into the "SELECT" statement.

Algorithm 4 extractAggregationClauses

Input: $\varphi, \mathcal{MG}$
Output: $\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle$

1: $\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle \leftarrow \emptyset$
2: **for each** $\mathcal{AF}_i \in functions(\mathcal{MG})$ **do**
3: $\quad \langle \mathcal{M} \rangle \leftarrow aggregates(\mathcal{AF}_i) \cap measures(\varphi)$
4: $\quad$ **if** $\langle \mathcal{M} \rangle \neq \emptyset$ **then**
5: $\quad\quad \langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle_i \leftarrow (\mathcal{AF}_i, \langle \mathcal{M} \rangle)$
6: $\quad$ **end if**
7: **end for**
8: **return** $\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle$

generateRollUpQueries

The following step of the *implicitRollUp* algorithm will be performed by Algorithm 5, *generateRollUpQueries*. The input of the algorithm is φ and the output is a set of Roll-Up queries $\langle \mathcal{R}\varphi \rangle$. Algorithm 5 will firstly generate the dimension queries $\langle \langle \mathcal{D}\varphi \rangle \rangle$ for each dimension $\mathcal{D}_i$ in $dimensions(\varphi)$ as $dimensionQueries(\mathcal{D}_i)$. Then it will be calculated the Cartesian product for $\langle \langle \mathcal{D}\varphi \rangle \rangle$ as $cartesianProduct(\langle \langle \mathcal{D}\varphi \rangle \rangle)$. Finally, for each set of the Cartesian product $\langle \mathcal{D}\varphi \rangle_i$ there will be generated a Roll-Up query $\mathcal{R}\varphi$ as $rollUpQuery(\varphi, \langle \mathcal{D}\varphi \rangle_i)$.

rewriteAll

Finally, once generated all the Roll-Up queries it will be possible to use the rewriting algorithm to rewrite each Roll-Up query into a query in SQL format. Algorithm 6 takes as input the query over $\mathcal{MG}$ and the set of Roll-Up queries $\langle \mathcal{R}\varphi \rangle$ giving as output

Algorithm 5 generateRollUpQueries

Input: φ
Output: $\langle \mathcal{R}\varphi \rangle$

1: $\langle\langle \mathcal{D}\varphi \rangle\rangle \leftarrow \emptyset$
2: **for each** $\mathcal{D}_i \in dimensions(\varphi)$ **do**
3: $\quad \langle\langle \mathcal{D}\varphi \rangle\rangle_i \leftarrow dimensionQueries(\mathcal{D}_i)$
4: **end for**
5: $\times\mathcal{D}\varphi \leftarrow cartesianProduct(\langle\langle \mathcal{D}\varphi \rangle\rangle)$
6: $\langle \mathcal{R}\varphi \rangle \leftarrow \emptyset$
7: **for each** $\langle \mathcal{D}\varphi \rangle_i \in \times\mathcal{D}\varphi$ **do**
8: $\quad \langle \mathcal{R}\varphi \rangle_i \leftarrow rollUpQuery(\varphi, \langle \mathcal{D}\varphi \rangle_i)$
9: **end for**
10: **return** $\langle \mathcal{R}\varphi \rangle$

a set of SQL queries $\langle \varphi_{sql} \rangle$ each one of that aggregates one possible combination of wrapper to the queried granularity. Algorithm 6 *rewriteAll* will perform multiple call of the function *rewrite* for each $\langle \mathcal{R}\varphi \rangle_i$, checking the *minimal* property for φ. The minimal constraints will be given by φ_{min} that will be generated removing all the triple $\langle \mathcal{L}, hasFeature, id(\mathcal{L}) \rangle$ to φ rather than in the higher granularity level of each dimension. Let us explain now why it is important to guarantee the minimal property for φ_{min}. The minimal property will allow not to perform impossible Drill-Down operations through look-up tables, and executing joins just in the upward direction and not in the opposite, guaranteeing to aggregate dimensional data correctly. Therefore, all $\mathcal{CQ}_i$ non minimal will be automatically discarded (e.g. the join operation aggregating data at higher granularity level to low granularity level not be considered).

Algorithm 6 rewriteAll

Input: $\varphi, \langle \mathcal{R}\varphi \rangle$
Output: $\langle \varphi_{sql} \rangle$

1: $\varphi_{min} \leftarrow \varphi$
2: **for each** $\mathcal{D}_i \in dimensions(\varphi)$ **do**
3: $\quad \varphi_{min} \leftarrow \varphi_{min\langle S,P,O \rangle} - \{\cup_{j=0}^{size(levels(\mathcal{D}_i))-1} \langle \mathcal{L}_{levels(\mathcal{D}_i)_j}, hasFeature, id(\mathcal{L}_{levels(\mathcal{D}_i)_j}) \rangle\}$
4: **end for**
5: $\langle \varphi_{sql} \rangle \leftarrow \emptyset$
6: **for each** $\mathcal{R}\varphi_i \in \langle \mathcal{R}\varphi \rangle$ **do**
7: $\quad \mathcal{CQ} \leftarrow rewrite(\mathcal{R}\varphi, \varphi)$
8: $\quad \langle \varphi_{sql} \rangle_i \leftarrow toSql(\mathcal{CQ})$
9: **end for**
10: **return** $\langle \varphi_{sql} \rangle$

makeSqlQuery

Algorithm 7 takes as input the set of SQL queries $\langle \varphi_{sql} \rangle$ generated by the procedure above, the group by clauses GB, the implicit aggregation clauses AGG and the explicit aggregation clauses $AGG^{'}$, producing as output a single SQL queries. It performs three important operations:

- It transforms each SQL query $\langle \varphi_{sql} \rangle$ wrapping each one of them with a group by clause, aggregating all the information for each granularity level. This operation allows making aggregation within each data wrapper, delivering the correct granularity for each one of them according to the implicit aggregation semantic of AGG.
- The SQL queries will be flattened folding all the SQL queries into a single query applying the UNION SQL operation, generating a single table having all the data wrappers aligned to the queried granularity.
- Finally, to preserve the correctness of the final result, it will be performed the final explicit aggregation with the semantic of $AGG^{'}$.

Algorithm 7 makeSqlQuery

Input: $\langle \varphi_{sql} \rangle, GB, AGG, AGG^{'}$
Output: φ_{sql}

1: **for each** $q_i \in \langle \varphi_{sql} \rangle$ **do**
2: $\quad \langle \varphi_{sql} \rangle_i \leftarrow wrapGroupBy(q_i, GB, AGG)$
3: **end for**
4: $\varphi_{sql} \leftarrow \langle \varphi_{sql} \rangle_1$
5: $\langle \varphi_{sql} \rangle \leftarrow \{\langle \varphi_{sql} \rangle - \langle \varphi_{sql} \rangle_1\}$
6: **for each** $q_i \in \langle \varphi_{sql} \rangle$ **do**
7: $\quad \varphi_{sql} \leftarrow \{\varphi_{sql} + "UNION" + q_i\}$
8: **end for**
9: **if** $AGG^{'} \neq null$ **then**
10: $\quad \varphi_{sql} \leftarrow wrapGroupBy(\varphi_{sql}, GB, AGG^{'})$
11: **end if**
12: **return** φ_{sql}

Algorithm 8 wrapGroupBy

Input: φ_{sql}, GB, AGG

Output: φ_{sql}

1: **return** $"SELECT" + GB + AGG + \varphi_{sql} + "GROUP\ BY" + GB + AGG$

Chapter 6

Implementation

In this section, it will be explained how all the algorithm and functions illustrated in section 5 have been actually implemented. Since this work is about graph, let's illustrate the main ways to deal with this kind of data structure. The most typical way to query a graph structure is by using *traverse* operations. Nowadays, there exists a lot of graph databases in-memory or not that allow to store graphs structure, offering *traverse* operations, typically in the form of pattern-matching queries.

In this work, by the way, we defined a new programmatic graph data model from scratch to overcome few limitations given by the original system. Since GFDM needs configuration files to describe the entire integration graph $\mathcal{I}$ and this procedure is very error-prone to do it as "handwriting", the new graph data model have been developed right to automate such procedure. It has been developed a domain specific language, that will be detailed in section 6.3 in order to define each testing scenario more easily, automating the file generation procedure. Once defined the domain specific language, a new model and *traverses* over the model itself to generate $\mathcal{I}$ configuration files, it was pretty straightforward to keep defining all the remaining *traverses* to implement the implicit Roll-Up algorithm directly over this data model as well. The entire implementation process has been done using as language Scala[1] because it is very good for mathematical models and domain specific language definition. The decision of using Scala as programming language have been also very good since a functional programming style is less error-prone rather than the typical imperative one. For this reason the development of the system have been very quick staring from the first releases.

6.1 Modules

This section is aimed to describe the code organization and clearly show how the different modules of the application interfaces to each other. Figure 6.1 depicts a UML package

[1] https://www.scala-lang.org

diagram that describes so. We want to first make clear that the organization described in Figure 6.1 will not exactly map 1-1 with the real source code, since its purpose is just to make clear the software structure and its behavior. Since in this work we are doing an extension of GFDM, in the diagram we can clearly distinguish the package **GFDM** that contains all the module behaving to the extended system and **ImplicitRollUp** that contains all the sources behaving to the framework developed. The package *model* contains the object-oriented models and all the traverses, and the *algorithm* package contains the algorithm illustrated in section 5.2, implemented into the source file *ImplicitRollUp*. This module uses the interface *QueryRewriting* to generate the SQL view and wrap it up with the aggregation syntax. The traverses into the module *Graph* will be able to generates the configuration file into the package *configurationScenarions/scenario*, that will be used by the module *ModelGeneration* to crate an instance of $\mathcal{I}$, that will finally be used by *QueryRewriting*. The DSL module adorns the *Graph* and the *Scenario* model structures. Finally, the package *Scenarios* will contain the running examples that will be generally written using the DSL syntax.

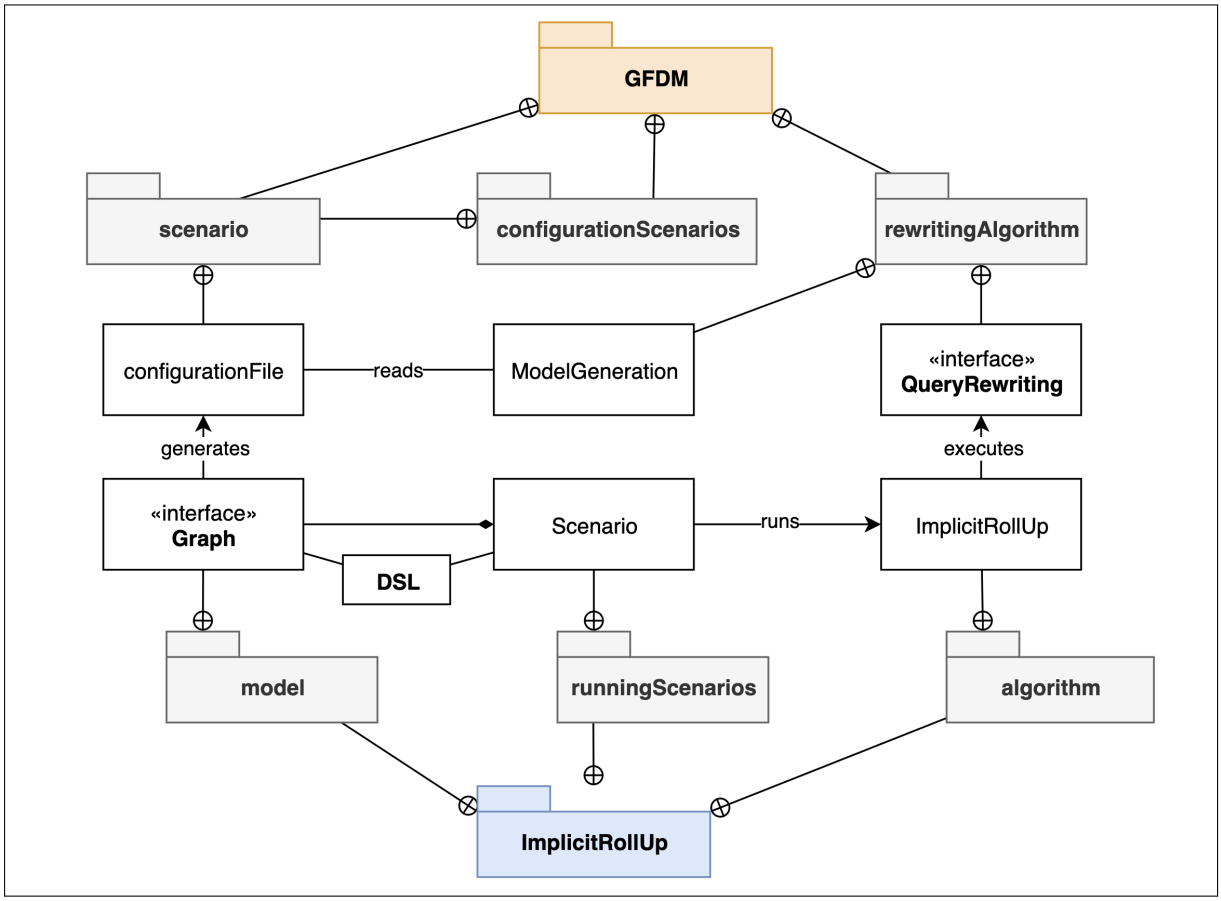


Figure 6.1: UML package diagram.

6.2 Execution flow

The sequence diagram in Figure 6.2 describes clearly how the communication between the different modules is performed. The execution flow starts in a *Scenario* where using the DSL or not both the semantic of $\mathcal{I}$ and the scenario will be described. The *Utils* module contains the automation flaw that generates all the configuration files (described in section 2.1) that GFDM needs. Right after, the *ImplicitRollUp* module will execute the implicit Roll-Up algorithm that is also described in section 5.2. The implicit Roll-Up algorithm will also need the rewriting algorithm module to execute the query rewriting and producing a relational algebra expression that will be converted to a SQL expression by the module *Sql*.

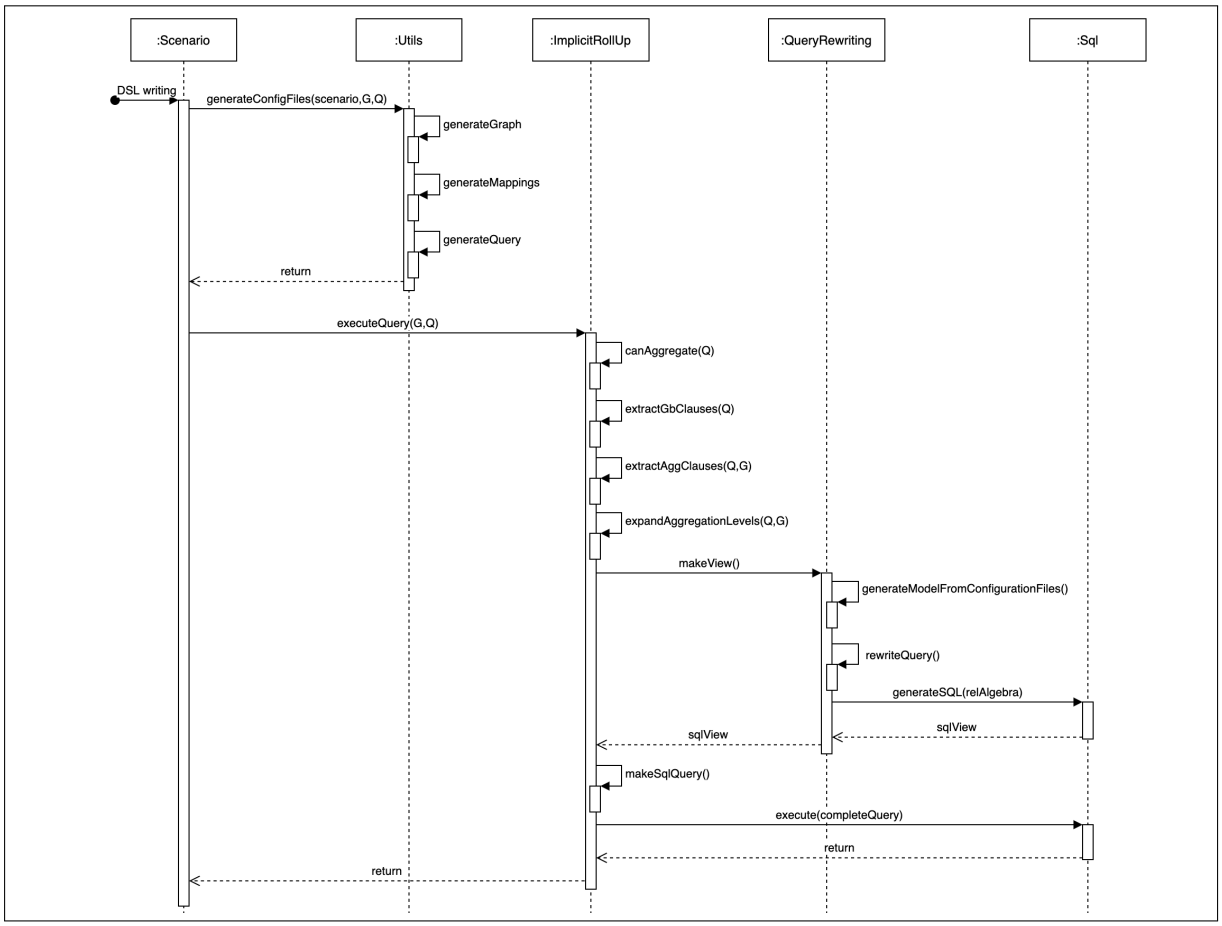


Figure 6.2: UML sequence diagram.

6.3 Domain Specific Language

In this section will follow some code snippets describing the grammar of the DSL (domain specific language). In particular, two kinds of DSL have been developed; The first one is related to the generation of the integration graph $\mathcal{I}$ and the second is specific to the generation of a configuration scenario. Let us start with a code snippet that will show how to build $\mathcal{I}$. On the top side of the code it will be defined all the features we would like to use. After, it will be defined $\mathcal{I}$ using the intuitive semantic of the DSL. As we know $\mathcal{I}$ is composed both by $\mathcal{MG}$ and $\mathcal{S}$. The first listing starts describing $\mathcal{MG}$. Since φ is subsumed by $\mathcal{MG}$ we have the same syntax for the definition of a query.

Listing 6.1: Multidimensional graph definition

```
val REGION = IdFeature("Region")
val COUNTRY = IdFeature("Country")

val NAME = IdFeature("Name")
val CATEGORY = IdFeature("Category")

val REVENUE = Measure("Revenue")

val multidimensionalGraph =
   Concept("Sales")
     .hasFeature {REVENUE}
     .->("location") { // Concept to concept/level relationship
      Level("Region")
        .hasFeature {REGION}
        .partOf { // Level to level relationship
         Level("Country")
           .hasFeature {COUNTRY}
        }
     }
     .->("product") {
      Level("Name")
        .hasFeature {NAME}
        .partOf {
         Level("Category")
           .hasFeature {CATEGORY}
        }
     }
```

The second listing describes the DSL for the generation of $\mathcal{S}$. It will be possible to define both wrappers and attributes, and to describe the *sameAs* relationship with F.

Listing 6.2: Source graph definition

```
val w1 =
   Wrapper("W1")
     .hasAttribute {
      Attribute("country") sameAs COUNTRY
     }
     .hasAttribute {
      Attribute("region") sameAs REGION
     }
     .hasAttribute {
      Attribute("revenue") sameAs REVENUE
     }
```

Finally, the last DSL proposed will help to deal with the generation of a scenario and also the run of the scenario itself. As we know a scenario is composed by the scenario name, a target graph, that can be both $\mathcal{G}$ or $\mathcal{MG}$, all the wrappers, all the aggregation functions and finally the query.

Listing 6.3: Scenario generation

```
class ScenarioName extends Scenario {
  scenario {
    "ScenarioName"
  }

  targetGraph{
    // The target graph
  }

  wrapper {
    // One wrapper, if there are more than one repeat this block
  }

  query {
    // The query
  }

  aggregation {
    // One aggregation function, if there are more than one repeat
       this block
  }

}
```

Finally, once crated the scenario, it will be possible to run it and flag if running implicit Roll-Up or not.

Listing 6.4: Running a scenario

```
object ScenarioRun extends App {
  new ScenarioName().run(executeImplicitRollUp = true)
}
```

Chapter 7

Experimentation

7.1 Performance analysis

In this section, it will follow the description of the overall experimentation phase to understand the implicit Roll-Up algorithm performances. In [1] we already have a detailed performance analysis where the scrutinized variables were:

- The number of features per concept.
- The number of edges covered by a query.
- The overall number of wrappers.
- The number of edges covered by a wrapper.
- The fraction of features in a concept covered by a query.
- The fraction of features in a concept covered by a wrapper.

In this work, the analysis have been carried on considering the three main variables that would affect the performances of the algorithm;

- The number of dimension, as $nDimension$.
- The number of levels, as $nLevel$.
- The number of wrappers for each possible dimensional combination, as $nWrapper$.

According to this variable configuration, it will hold that the total amount of Roll-Up queries $\langle \mathcal{R}\varphi \rangle$ and then also the total amount of possible dimensional combination of data wrappers are given by equation 7.1.

$$nRollUpQueries \equiv nLevels^{nDimension} \tag{7.1}$$

The experimentation have been carried out analyzing the performances of the system studying the performance trend of each of the variable doing studies over multiple values of a variable and fixing the value of the remaining two variables. In doing this the cost of each call of the function *rewrite* (e.g. the rewriting algorithm) have been considered as a constant value by considering very few wrappers, that as we know from [1] is one of the main cost factor of RA. The study will analyze the trend of the computation time required for the rewrite of all the Roll-Up queries $timeRewriteRollUpQuery$, the execution of the SQL query $timeExecuteSQL$ and the execution of the Roll-Up algorithm $timeAlgorithm$ minus the rewriting and SQL execution time. The analysis will also consider the variable $nRollUpQueries$ given by equation 7.1 because according to our expectation this variable is the main cost factor.

Number of dimensions

Let us now study the complexity curve of the algorithm by analyzing the variable $nDimension$ that represents the number of dimension of the experiment multidimensional graph. We will then fix the value of variables $nLevel = 2$ and $nWrapper = 1$, executing the algorithm with a linear increment of $nDimension$ as $\langle nDimension \rangle \equiv \{1, ..., 10\}$. Then we would have the number of Roll-Up query given by $nRollUpQueries \equiv 2^{\langle nDimension \rangle_i}$. Figure 7.1 shows clearly that the main complexity factor of the algorithm is the time of rewriting all the Roll-Up queries $timeRewriteAllRollUpQuery$ (blue curve), and such complexity has the exponential trend described in equation 7.1. The figure also shows that growing the variable $nDimension$ there is an exponential growth also for the number of Roll-Up queries $nRollUpQueries$.

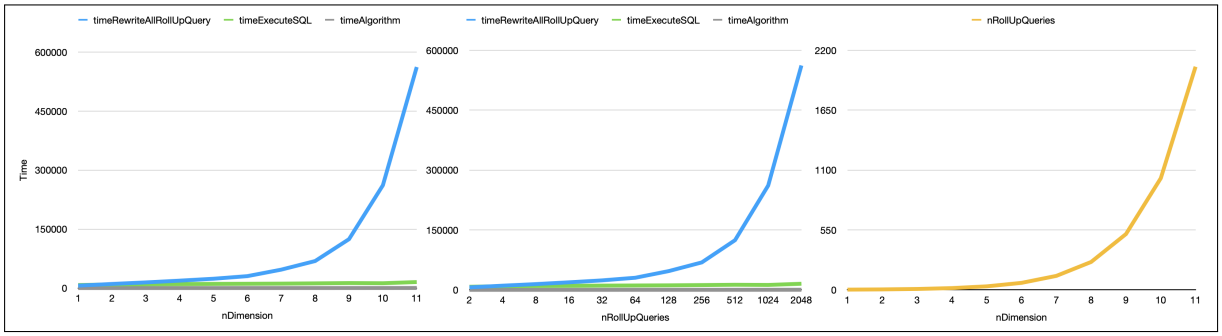


Figure 7.1: Trend of $timeRewriteAllRollUpQuery$ (blue), as the time spent for the execution of the RA for each Roll-Up query, $timeExecuteSQL$ (green) as the time spent for the execution of the Roll-Up SQL query, $timeAlgorithm$ (grey) as the time spent for the execution of the implicit Roll-Up algorithm and the increment of the number of Roll-Up queries $nRollUpQueries$ (yellow), incrementing the variable $nDimension$.

Number of levels

Let now study the variable $nLevel$ representing the number of levels of each dimension. The analysis have been carried on fixing the value of the variable $nDimension = 2$ and $nWrapper = 1$ and having a linear increment for the variable $nLevel$ as $\langle nLevel \rangle \equiv \{1, ..., 10\}$. Then the number of Roll-Up query for each experiment would be given by $nRollUpQueries \equiv \langle nLevel \rangle_i^2$. As we can see from the chart depicted in Figure 7.2 the trends respect the expectations having a quadratic growth for the time for the execution of all the Roll-Up queries $timeRewriteAllRollUpQuery$ (bluecurve) being still the most relevant cost for the algorithm.

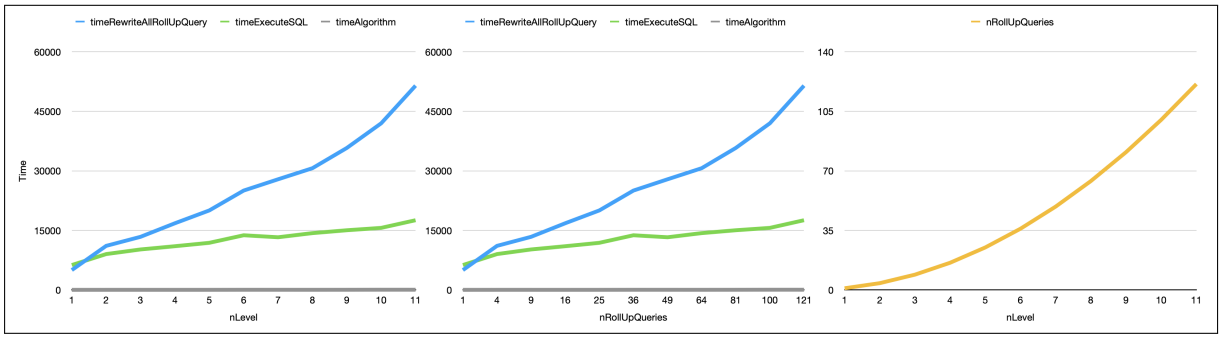


Figure 7.2: Trend of $timeRewriteAllRollUpQuery$ (blue), as the time spent for the execution of the RA for each Roll-Up query, $timeExecuteSQL$ (green) as the time spent for the execution of the Roll-Up SQL query, $timeAlgorithm$ (grey) as the time spent for the execution of the implicit Roll-Up algorithm and the increment of the number of Roll-Up queries $nRollUpQueries$ (yellow), incrementing the variable $nLevel$.

Number of wrappers

Let us finally try to see how the number of wrappers can affect the implicit Roll-Up algorithm. What we expect is that the variable $nWrapper$ affects just the RA costs, having any influence on the implicit Roll-Up algorithm itself rather than over the $rewrite$ method. The experiments have been carried on adopting the variables $nDimension = 2$ and $nLevel = 2$, increasing linearly the variable $nWrapper$ as $\langle nWrapper \rangle \equiv \{1, ..., 10\}$. As we can see in Figure 7.3, the total number of wrappers would then be calculated summing the number of data wrappers and the total of look up wrappers, having $nTotalWrapper \equiv nWrappers * nLevels^{nDimension} + (nLevel - 1) * nDimension \equiv nWrappers * 2 + 2$. According to $timeRewriteAllRollUpQuery$ we can clearly see that it is way higher increasing the variable $nWrapper$ then increasing $nDimension$ and $nLevel$, then we can state that $nWrapper$ has a strong impact on RA.

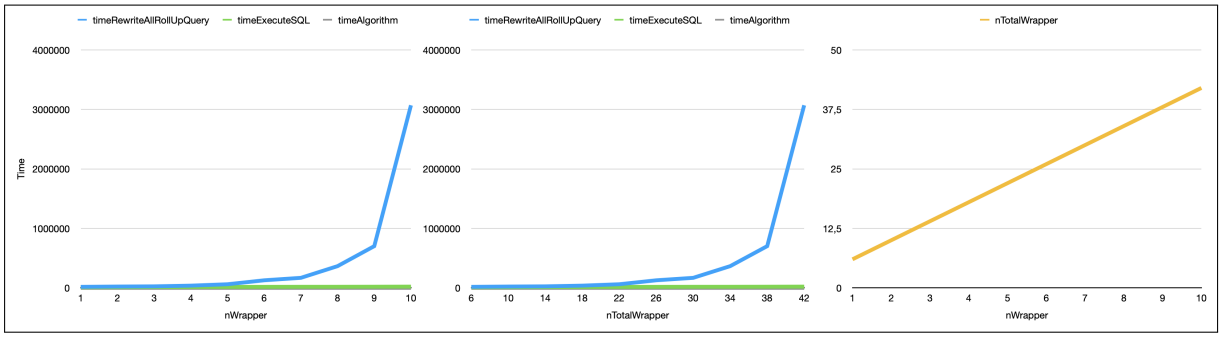


Figure 7.3: Trend of *timeRewriteAllRollUpQuery* (blue), as the time spent for the execution of the RA for each Roll-Up query, *timeExecuteSQL* (green) as the time spent for the execution of the Roll-Up SQL query, *timeAlgorithm* (grey) as the time spent for the execution of the implicit Roll-Up algorithm and the increment of the number of Roll-Up queries *nRollUpQueries* (yellow), incrementing the variable *nWrapper*.

Trend analysis

Let finally show how each variable affect the time of execution of the implicit Roll-Up algorithm itself without considering the external costs as the RA and the SQL execution. Figure 7.4 shows how each variable affects the performance of the algorithm. What we see from Figure 7.4 is that the variable *nWrapper* and as consequence *nTotalWrapper* doesn't affect the performance of the implicit Roll-Up algorithm, while the number of dimension *nDimension* and the number of levels for each dimension *nLevel* does respectively with an exponential and a quadratic growth.

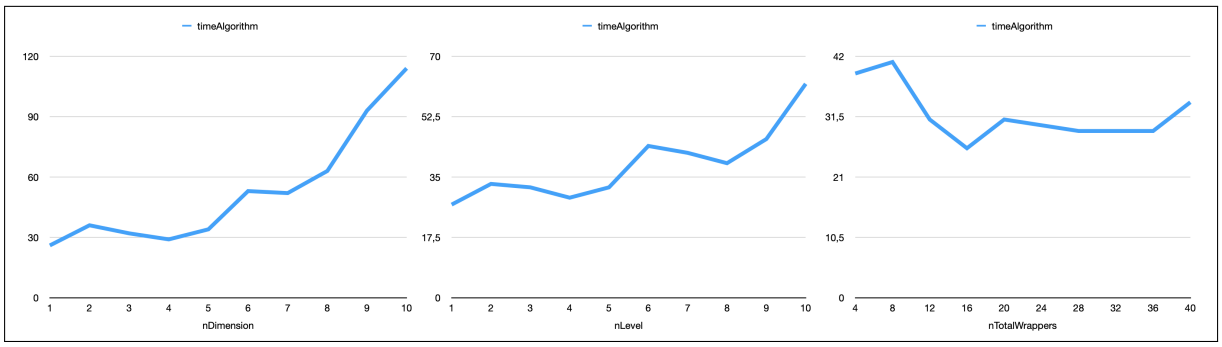


Figure 7.4: How each variable affects the implicit Roll-Up algorithm.

7.2 Acceptance test

This section will show the procedure adopted for the validation of the implicit Roll-Up algorithm developed. The correctness of the algorithm have been asserted by means of Acceptance Tests. During the design phase, we developed experiments aimed to test the correctness of the algorithm over the following circumstances:

- Correctness of the implicit data aggregation within wrappers, aggregating each wrapper data to the queried granularity.
- Correctness of the overall data aggregation aggregating data between wrappers leading to a final correct result.

Then, by developing experimental scenarios by using the DSL as shown in section 6.3 it has been possible to define acceptance tests for both of the previous scenarios, where several tests have been run with several configurations of the variables representing the number of dimensions $nDimension$, the number of levels $nLevels$, the number of wrappers $nWrapper$ and the configuration of the wrappers itself.

Chapter 8

Conclusions

In this work, we presented a framework built over GFDM able to perform automatic Roll-Up operation, delivering data at the correct queried granularity level. During the first part of the work, we studied the graph model proposed by GFDM, able to define integration graphs. Then we proposed a model able to represent multidimensional schema through GFDM syntax, annotating with the graph semantic facts, dimension, levels and aggregations. Such model have been called Multidimensional Graph. Afterwards, once chosen the data model, it has been possible to define the automatic data aggregation algorithm (e.g. the Roll-Up operation). This operation consists of three key operation; For first, each data source will be aggregated to the correct data granularity through the execution of the Rewriting Algorithm and dimensional data will be joined by using look-up tables. Secondly, the query will be parsed, extracting the aggregation semantic. Finally, all the queries generated by the Rewriting Algorithm will then be converted in SQL expressions, wrapped up with the aggregation semantic, glued up into a single SQL query and wrapped up again with the final aggregation semantic. According to the goals of this project, we can state we have successfully completed all the requirements and delivered a working artifact able to address such functionalities. We can finally consider implicit Roll-Up operation feasible and the experimental results shows which are the criticism of this approach.

8.1 Future improvement

In this section it will be listed some possible improvements for this project.

OLAP functionalities

In this work we presented a flexible multidimensional model built on top of GFDM graph syntax. As we introduced in Section 1 our approach is the first one to extend a graph-based virtual data integration system to incorporate OLAP-like aggregations, and

also according to Section 3 we do so defining annotations over the schema (the TBOX) and not directly over the data sources (the ABOX). A possible improvement of this project may be reusing the existing multidimensional graph structure that is used for the definition of the TBOX for the implementation of any other OLAP operator directly over GFDM, such as Drill-Down, Drill-Across, change base and slicing, enriching the functionalities of the framework proposed. Another possible approach of extension of this framework is adopting an approach similar to what [2] did, implementing OLAP operators not directly over the original system, but parsing the multidimensional graph in order to generate a separate model similar to a data cube.

Aggregation function semantics

A possible improvement of this work is guaranteeing correctness of aggregation function implicitly. According to what we did, the aggregation operation have been divided in two main steps. Firstly, data are aggregated to the queried data granularity within wrappers through implicit aggregation functions pointing to measures annotated into the TBOX. Secondly, aggregated source data are glued together and aggregated a second time by using an explicit aggregation function to preserve correctness in the aggregation. This two phases aggregation is clearly due to the necessity of preserving the correctness property of aggregation operators, since some operators are not commutative and associative. These properties may be annotated into the multidimensional graph to automatically detect how operators have to be applied.

Bibliography

[1] Nadal, S., Abello, A., Romero, O., Vansummeren, S., and Vassiliadis, P. (2021). Graph-driven Federated Data Management. IEEE Transactions On Knowledge And Data Engineering, 1-1. doi: 10.1109/tkde.2021.3077044

[2] Ghrab, A., Romero, O., Skhiri, S. et al. TopoGraph: an End-To-End Framework to Build and Analyze Graph Cubes. Inf Syst Front 23, 203–226 (2021). https://doi.org/10.1007/s10796-020-10000-z

[3] Golfarelli, M., Rizzi, S. (2009). Data warehouse design: Modern principles and methodologies. New York: McGraw-Hill.

[4] Halevy, A. Answering queries using views: A survey. The VLDB Journal 10, 270–294 (2001).

[5] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, "Linking data to ontologies," Journal on Data Semantics, vol. 10, pp. 133–173, 2008.

[6] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev, "Reasoning over extended ER models," in ER, 2007.

[7] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom, "The TSIMMIS approach to mediation: Data models and languages," J. Intell. Inf. Syst., vol. 8, no. 2, pp. 117–132, 1997.

[8] M. T. Roth, M. Arya, L. M. Haas, M. J. Carey, W. F. Cody, R. Fagin, P. M. Schwarz, J. Thomas, and E. L.Wimmers, "The Garlic Project," in SIGMOD, 1996.

[9] D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori, and M. Vincini, "Information integration: The MOMIS project demonstration," in VLDB, 2000.

[10] M. Arenas, P. Barceló, L. Libkin, and F. Murlak, Foundations of Data Exchange. Cambridge University Press, 2014.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi, "Query processing under GLAV mappings for relational and graph databases," Proc. VLDB Endow., vol. 6, no. 2, pp. 61–72, 2012.

[12] A. Y. Levy, A. Rajaraman, and J. J. Ordille, "Querying heterogeneous information sources using source descriptions," in VLDB, 1996.

[13] O. M. Duschka, M. R. Genesereth, and A. Y. Levy, "Recursive query plans for data integration," J. Log. Program., vol. 43, no. 1, pp. 49–73, 2000.

[14] R. Pottinger and A. Y. Halevy, "Minicon: A scalable algorithm for answering queries using views," VLDB Journal, vol. 10, no. 2-3, pp. 182–198, 2001.

[15] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao, "Ontop: Answering SPARQL queries over relational databases," Semantic Web, vol. 8, no. 3, pp. 471–487, 2017.

[16] F. Priyatna, O. Corcho, and J. F. Sequeda, "Formalisation and experiences of r2rml-based SPARQL to SQL query translation using morph," in WWW, 2014

[17] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo, "The MASTRO system for ontology-based data access," Semantic Web, vol. 2, no. 1, pp. 43–53, 2011.

[18] C. Beeri, A. Y. Levy, and M. Rousset, "Rewriting queries using views in description logics," in PODS, 1997.

[19] F. Goasdoue and M. Rousset, "Answering queries using views: A KRDB perspective for the semantic web," ACM Trans. Internet Techn., vol. 4, no. 3, pp. 255–288, 2004.

[20] J. Baget, M. Lecl'ere, M. Mugnier, S. Rocher, and C. Sipieter, "Graal: A toolkit for query answering with existential rules," in RuleML, 2015.

[21] N. Konstantinou, E. Abel, L. Bellomarini, A. Bogatu, C. Civili, E. Irfanie, M. Koehler, L. Mazilu, E. Sallinger, A. A. A. Fernandes, G. Gottlob, J. A. Keane, and N. W. Paton, "VADA: an architecture for end user informed data preparation," J. Big Data, vol. 6, p. 74, 2019.

[22] M. Buron, F. Goasdoue, I. Manolescu, and M. Mugnier, "Ontologybased RDF integration of heterogeneous data," in EDBT, 2020.

[23] Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.S. (2009). Graph OLAP: a multi-dimensional framework for graph data analysis. Knowledge and Information Systems, 21(1), 41–63.

[24] Zhao, P., Li, X., Xin, D., Han, J. (2011). Graph cube: on warehousing and OLAP multidimensional networks. In Proceedings of the 2011 ACM SIGMOD international conference on management of data (pp. 853–864): ACM.

[25] Denis, B., Ghrab, A., Skhiri, S. (2013). A distributed approach for graph-oriented multidimensional analysis. In 2013 IEEE international conference on big data workshops (pp. 9–16): IEEE.

[26] Yin, M., Wu, B., Zeng, Z. (2012). HMGraph OLAP: a novel framework for multi-dimensional heterogeneous network analysis. In Proceedings of the 15th international workshop on data warehousing and OLAP (pp. 137–144): AC

[27] Wang, Z., Fan, Q., Wang, H., Tan, K.-l., Agrawal, D., El Abbadi, A. (2014). Pagrol: parallel graph OLAP over large-scale attributed graphs. In 2014 IEEE 30th international conference on data engineering (ICDE) (pp. 496–507): IEEE.

[28] Wang, P., Wu, B., Wang, B. (2015). TSMH graph cube: a novel framework for large scale multi-dimensional network analysis. In 2015 IEEE international conference on data science and advanced analytics (DSAA) (pp. 1–10): IEEE.

[29] Wu, X., Wu, B., Wang, B. (2017). PD graph cube: model and parallel materialization for multidimensional heterogeneous network. In 2017 International conference on cyber-enabled distributed computing and knowledge discovery (CyberC) (pp. 95–104): IEEE.

[30] Kang, S., Lee, S., Kim, J. (2019). Distributed graph cube generation using spark framework. J Supercomput, 1–22.