Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze Informatiche

Implicit Roll-Up over graph-based data integration system

Relatore:
Matteo Golfarelli

Presentata da:
Filippo Pistocchi

Collaboratori:
Oscar Romero,
Sergi Nadal

Sessione
Anno Accademico

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 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 and stakeholders in companies will have to deal with heterogeneous dataset generated form different sources distributed all around the world. Clearly handling such resources it is not a trivial task and to do so without specific tools it may be 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. As we know from the literature there exists two main kind of architectures for data integration; Physical data integration, is typically grounded in data warehouse and business intelligence systems and its biggest challenge is to structure and prepare data as multidimensional schema in order to analyse study fact as multidimensional models. The aim of this technique is strictly analytical and the operation performed are aggregations over numerical data. Typically this technique is mostly used to gather assets within a corporation, studying the business processes in order to define strategical decisions aimed to the internal improvement. Virtual data integration aim, as opposite, is not grounded in just data aggregations. Virtual data integration systems main goal is gathering huge amount of disconnected and heterogeneous data making easier the data wrangling process to data enthusiasts.

What we would like to do in this project is extend a virtual data integration system (graph-based) giving the possibility to support a multidimensional schema (with the shape of a graph) as data warehouses does. This idea may lead to have both the advantages of virtual and physical data integration over a single system.

1.1 Data integration techniques

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

- Physical data integration
  - Multidimensional model
  - Multidimensional cube $\mathcal{MC}$
    - Tell what $\mathcal{F}$,$\mathcal{D}$, $\mathcal{L}$, $\mathcal{M}$ are.
  - Roll-Up[1] and Drill-Down
- Virtual data integration
- Graph-based virtual data integration
  - Ontology based data access
  - Ontology mediated query
    - LAV mappings.
    - W

1.2 Implicit Roll-Up philosophy

From now on we will work on a graph virtual data integration, ontology mediated query based model, also KG (knowledge graph) based, that we will call $\mathcal{Z}$, in particular we will focus on a concerete implementation of $\mathcal{Z}$ called GFDM, more detail regarding this can be found in section 4, but also in the real source TKDE publication [1]. Let's define now the metadata describing a graph target schema target schema as $\mathcal{G}$ and a pattern matching or visual query over $\mathcal{G}$ as φ. What we would like to do in this project is to develop a multidimensional graph schema that reuses the metadata structure of $\mathcal{G}$ and extend it giving the capability of supporting dimensions, levels and measures as multidimensional cubes $\mathcal{MC}$. Let's identify this metadata structure as multidimensional graph, identified as $\mathcal{MG}$.

The goal of the project then it will be trying to implement the multidimensional Roll-Up operation, one of the most important $\mathcal{MC}$ feature, over $\mathcal{MG}$. This may be very useful for different reasons; Considering that $\mathcal{Z}$ deals with lots of heterogeneous dataset,

[1]https://en.wikipedia.org/wiki/Operation$_Roll - Up$

as declared in the introduction, it will be very likely to have data sources producing information, behaving to the same domain, but recorded in different granularity. Therefore, the extension we would like to do in $\mathcal{Z}$ is, for each φ over $\mathcal{G}$, or more specifically $\mathcal{MG}$, trying to implicitly extrapolate a Roll-Up semantic and when it is possible to do so, deliver a result for φ at the queried data granularity, considering the possibility of pushing up data behaving to the lowest data granularity to the queried data granularity available into $\mathcal{MG}$, aggregating implicitly the numeric study measures. Let's call this operation implicit Roll-Up.

The implicit Roll-Up then may be very useful for data scientists working over $\mathcal{Z}$ since it would come across to a subset of problems that data scientists face off daily. As we can see Figure 1.1 is showing a multidimensional graph $\mathcal{MG}$ and a query φ over $\mathcal{MG}$ as a visual dashed red line. The graph is describing a study fact related to *Sales*, two study measures, respectively the *total revenue* and the *number of unit* and three dimensions (*spatial*, *temporal* and *product*), each one having three granularity levels. Let's focus now just on the *spatial* dimension, that have *City*, *Region* and *Country* as vertexes. Given the semantic of φ, it is very intuitive that the user who triggered φ is interested into the *Region* granularity level for the *spatial* dimension. What we want our system to do is not just giving as result just the data at *Region* granularity level, as GFDM would already do, but a total Roll-Up for the entire *spatial* dimension, for all the lower granularity levels than the queried one, trying to deliver also data at *City* granularity to *Region* where possible, performing implicit aggregations over the queried measures granularity (this operation will clearly have to be done also for all the other queried dimensions).

The advantages of an implicit Roll-Up operation in this case would be twofold; Firstly this operation would save really lots of time to data scientist interested in this kind of operation, providing a huge automation process underneath, delivering correctly aggregated data. Secondly, implicit Roll-Up would also allow to aggregations more accurate and correct since in the aggregation computation will be considered both the queried granularity data measures and also, in addition, 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. Doing this operation we assume that data are disjointed.

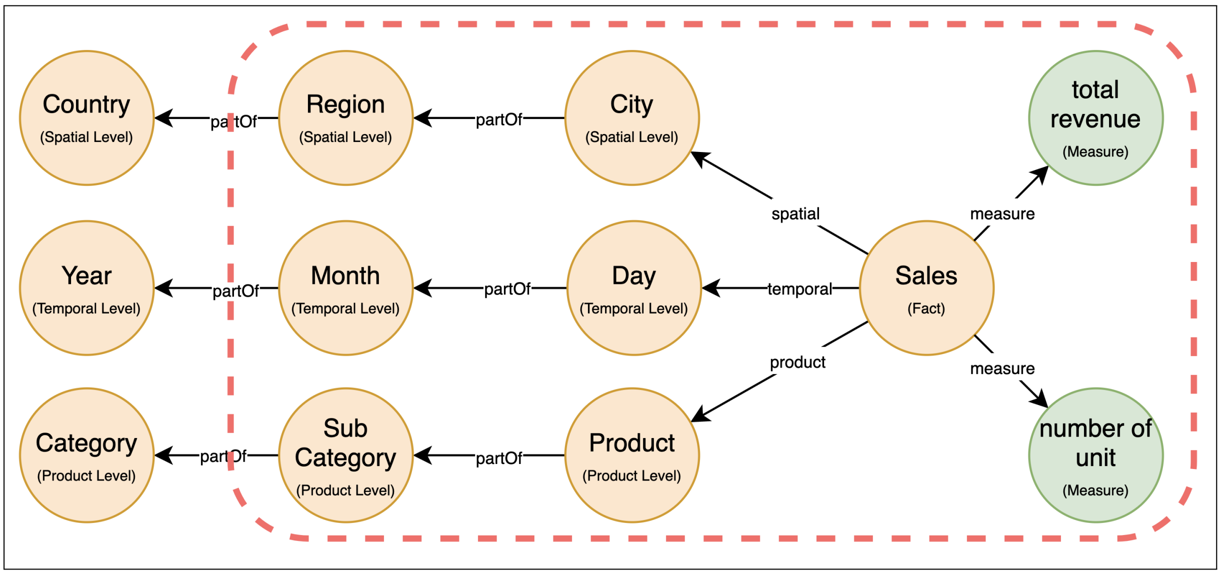

Figure 1.1: Idea of a simplified multidimensional graph $\mathcal{MG}$ having three different granularity level for three dimension and two numerical data. Also a query φ have been depicted as a dashed red line.

Chapter 2

Objectives

This section is a natural follow-up of chapter 1 where all the requirements of the to-be vision will be formalised. All the following functionalities will be built over GFDM as a natural extension of the system. It will follow also a clear distinction of all the different kind of functionality, that will be described as hierarchies.

2.1 Functional Requirements

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

1. Perform implicit Roll-Up on the result of a query φ over a multidimensional graph $\mathcal{MG}$, automatically delivering the right data granularity.
    - 1.1 Understanding if it is possible to execute the implicit Roll-Up operation given φ
        - 1.1.1 If it is not possible to execute the implicit Roll-Up the output of φ would be just the result that GFDM would give.
        - 1.1.2 If it is possible to execute the implicit Roll-Up the output of φ would be the Roll-Up of the output given by GFDM.
    - 1.2 Extraction of group by clauses from φ.
        - 1.2.1 Parsing of the group by clauses.
    - 1.3 Extraction of aggregating clauses from φ.
        - 1.3.1 Parsing of the aggregating clauses.
    - 1.4 Generation of dimension queries.
    - 1.5 Generation Roll-Up queries.

1.6 Generation of the sql query

  1.6.1 Rewriting the Roll-Up queries respecting the minimality of φ

  1.6.2 Conversion in sql of all the rewrited queries.

  1.6.3 Wrapping up all the queries with aggregation clauses.

  1.6.4 Union of all the queries

  1.6.5 Wrapping up the final result with aggregation clauses again.

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

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

   2.1 Definition of the scenario name.

   2.2 Definition of the target schema $\mathcal{MG}$.

   2.3 Definition of the query φ.

   2.4 Definition of the wrappers and attributes.

   2.5 Definition of the aggregation functions.

3. 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.

   3.1 Generation of the target schema triples (global_graph.txt).

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

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

   3.3 Generation of the SPARQL query (query.txt).

   3.4 Generation of the sources graph containing wrappers and attributes triples (source_graph.txt).

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

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

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

2.2 Non Functional Requirements

Here will follow all the requirements that are not functionalities.

1. Keeping the KG syntax of GFDM to model a multidimensional graph $\mathcal{MG}$ without changing it.
2. Avoid to hard-code the query syntax in the design process, keeping the system more flexible as possible.
3. The extension should be most transparent as possible to the user.
4. Multi-platform compatibility (usage of relative paths with ad-hoc system separators).
5. Robustness of the automation pipeline.

2.3 Implementation requirement

Finally here will be listed the requirements related to the implementation work.

1. Reuse modules of GFDM modules to develop implicit Roll-Up.
2. Wrap GFDM modules up adding implicit Roll-Up algorithm.
3. Use a programming language able to describe DSLs.
4. Use a functional programming language in order to speed up the implementation process.

Chapter 3

Related work

Related work to the objectives defined and gaps justifying the need to develop your thesis.

Are there also any other implicit rollUp system in the literature?

Chapter 4

Technology preliminaries

The aim of this section is to describe GFDM system, in order to give to the reader a basic knowledge that will allow to understand all the reasoning of 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 hole 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's now describe the vertexes and the edges of $\mathcal{I}$ and their relationship, keeping Figure 4.1 as a model example of the following formalisation. The integration graph is divided in two main components connected each other; The target 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 variable exposed by each one of them. For the target graph $\mathcal{G}$ there are three main kind of vertexes $V_{\mathcal{G}}$, the concepts C, the features F and also the identifier features F^{id}. As for the edges $E_{\mathcal{G}}$ there may exists just two combination; The connection between C and F always described by the label $hasFeature$ and the connection between C and C

linked via custom labels.

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

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

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

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

The source graph $\mathcal{S}$, that is responsible of 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 each other by the label *hasAttribute*.

$$W \wedge A \sqsubseteq V_{\mathcal{S}} \tag{4.5}$$

$$hasAttribute \sqsubseteq E_{\mathcal{S}} \tag{4.6}$$

Let's finally consider the relationship *sameAs* that will link each A behaving to $\mathcal{S}$ to the corresponding F in $\mathcal{G}$.

$$\mathcal{I}_{\langle S,P,O \rangle} \equiv \mathcal{G}_{\langle S,P,O \rangle} \cup \mathcal{S}_{\langle S,P,O \rangle} \cup sameAsTriples(\mathcal{G}, \mathcal{S}) \tag{4.7}$$

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-portion of $\mathcal{G}$. Let's use $mapping(W)$ denoting the LAV mapping related to W.

$$mapping(W)_{\langle S,P,O \rangle} \sqsubseteq \mathcal{G}_{\langle S,P,O \rangle} \tag{4.8}$$

4.2 Query answering

This section will describe how querying works in GFDMand what output does a query generates. As we introduced before a query is define by φ and it is a connected sub-portion of $\mathcal{G}$.

$$\varphi_{\langle S,P,O \rangle} \sqsubseteq \mathcal{G}_{\langle S,P,O \rangle} \tag{4.9}$$

The algorithm that is responsible for query answering is the *Rewriting Algorithm* (RA) and its structure is clearly explained in paper [1]. Its purpose is to find all the LAV mapping, or possible join combination of them via F^{id}, in $\mathcal{G}$ holding

$$\varphi_{\langle S,P,O \rangle} \sqsubseteq mapping(W_i)_{\langle S,P,O \rangle} \tag{4.10}$$

$$\varphi_{\langle S,P,O \rangle} \sqsubseteq \{mapping(W_1) \bowtie ... \bowtie mapping(W_n)\}_{\langle S,P,O \rangle} \tag{4.11}$$

$$mapping(W_i)_{\langle S,P,O \rangle} \cap \varphi_{\langle S,P,O \rangle} \neq \emptyset \tag{4.12}$$

The LAV mappings holding this rules would then lead the source query construction by the following operation, that will be then converted in relational algebra expressions.

The rewriting algorithm gives also the possibility of expressing minimality constraints (*minimal* property) will guarantee that the relational algebra operations generated are the only one needed to answer φ.

$$mapping(W_i)_{\langle S,P,O\rangle} \cup \{mapping(W_1)\bowtie...\bowtie mapping(W_n)\}_{\langle S,P,O\rangle} \tag{4.13}$$

This algorithm is more precisely described in [1]. The output of RA is a relational algebra query φ_{RA}. Finally, once we have the query, it is possible to convert φ_{RA} in a SQL query φ_{sql}.

Figure 4.1 will show with an example the execution of a query over $\mathcal{I}$, that will have as output the following algebra query.

$$\Pi_{name,price,id,number\_of\_unit}(w1 \bowtie_{id=id} w2) \tag{4.14}$$

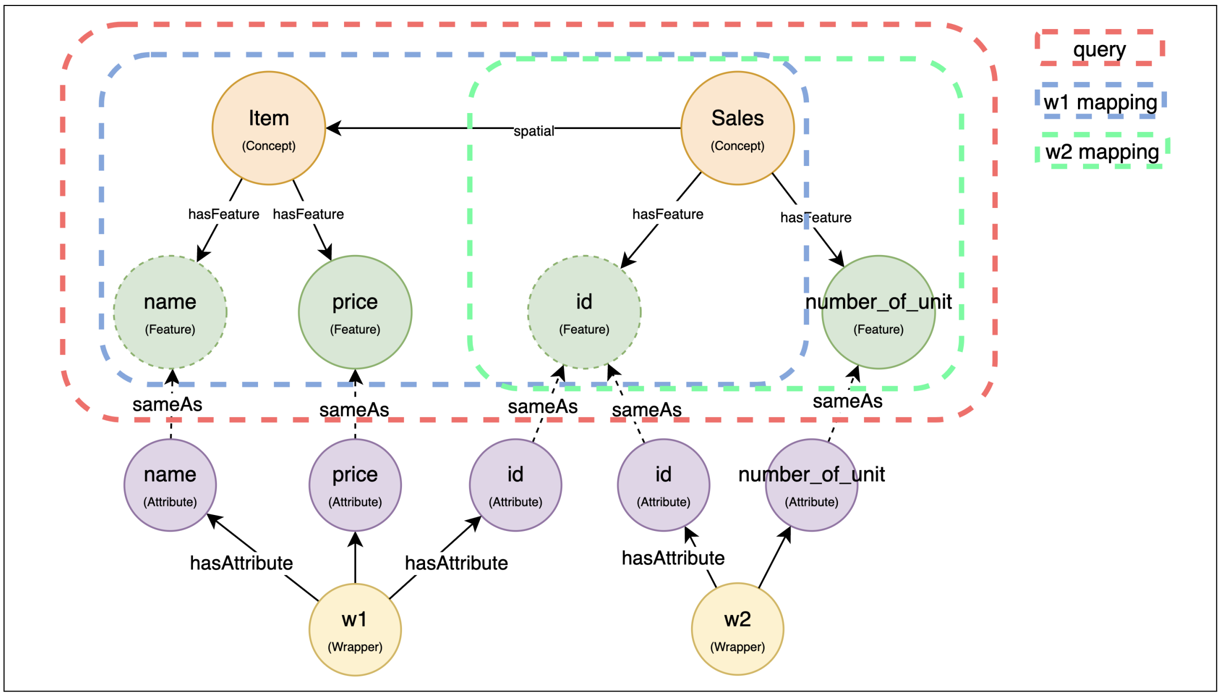

Figure 4.1: A graphical representation of the graph model adopted by GFDM. The example depicts an integration graph $\mathcal{I}$ related to sales, and a query φ over $\mathcal{G}$ (the dashed red line), the LAV mapping of wrapper *w1* (blue dashed line) and the LAV mapping of *w2* (green dashed line).

4.3 Joining dimensional data

In this section we would like to 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 rise up data at a lower granularity to the desired higher granularity and since this feature can actually already be achieved in GFDM this is why it will be explained into this chapter and not in the following one. To execute the join of dimensional data it will be exploited the Rewriting Algorithm described in section 4.2, since it will allow to execute join operations through different wrapper attributes.

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 between dimensions since, for each functional dependency, they will store how a level maps to the following higher granularity level. This kind of approach, to work, will need a preliminary phase in which all the look-up table wrappers W_{lut} have to be loaded into the system, or linked to any multidimensional data repository, otherwise it will not be possible to execute the join and as consequence the Roll-Up.

Let's now start describing a how a dimension will be represented in GFDM with the help of Figure 4.2. As in the multidimensional cube model $\mathcal{MC}$, where each dimension is divided in levels each one associated to an identifier, the same idea is replicated into the multidimensional graph $\mathcal{MG}$. 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, respectively *city*, *region* and *country*.

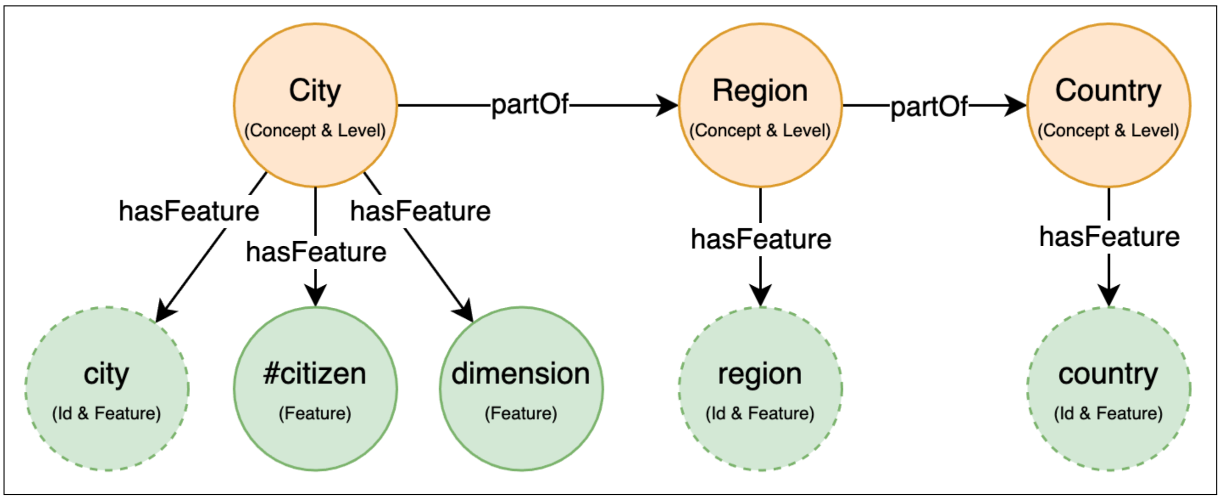

Figure 4.2: How a dimensions will be represented over GFDM.

Once explained how a dimension can be represented in $\mathcal{MG}$ let's now go a bit ahead detailing how multidimensional data are joined **within** a dimension by means of the rewriting algorithm. Figure 4.3 depicts a very simple 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 to join dimensional data. In particular W_{lut} stores all the possible combination describing how each city can be converted into its respective a country. Given a comprehensive query φ that involves for a dimension, all the levels and the respective identifier feature, it will be possible to join all the levels queried with the higher granularity level if there is a W_{lut} bridging all the levels functional dependencies. Finally the expected result from the execution of φ with the rewriting algorithm would be the join between LAV mapping of *w2*,*w3* (the look-up table) and *w1*, having as result the following relational algebra expression.

$$\Pi_{city,country}((w1 \bowtie_{city=city} w3) \bowtie_{country=country} w1) \tag{4.15}$$

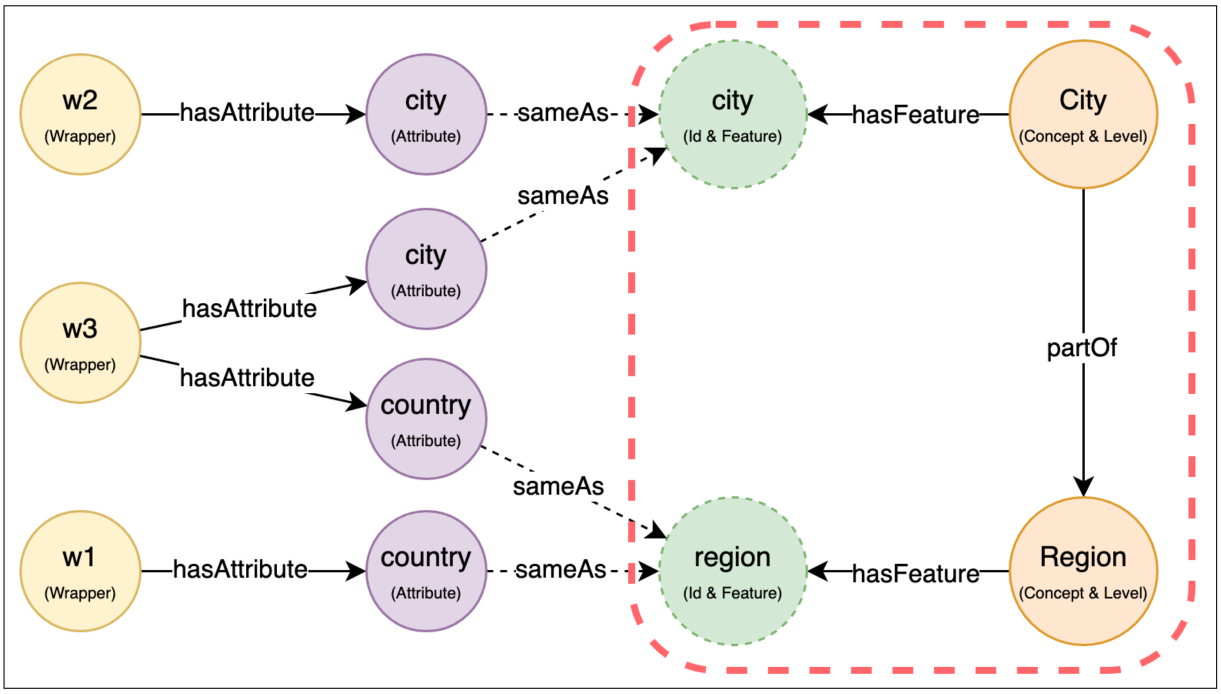

Figure 4.3: How dimensional data are joined within a dimension over GFDM, where the dashed red line is a visual query φ.

Let's now focus on the possible showcases of join **between** dimensions. There are two main possible configuration of dimension representation wrapper side. It is possible that each wrapper contains the entire dimensional variables related to that observation, and it is actually the most likely scenario, but since the project is grounded in a data integration scenario, it may be also possible having disjointed dimensional data, each one exposed by a different wrapper but at the same time related to a single fact observation. Figure 4.4 is representing the first described scenario in which all the dimensional variable are represented inside just one wrapper (*w1*). This scenario is the easiest since it don't need the join of different dimensions variables and it will work over the assumptions we already did. Given the queried feature *region*, *total revenue* and *name*, the rewriting algorithm would return the data behaving to the wrappers having a LAV mapping (or join combination of mapping) where φ is subsumed by the mapping. In figure 4.5, as

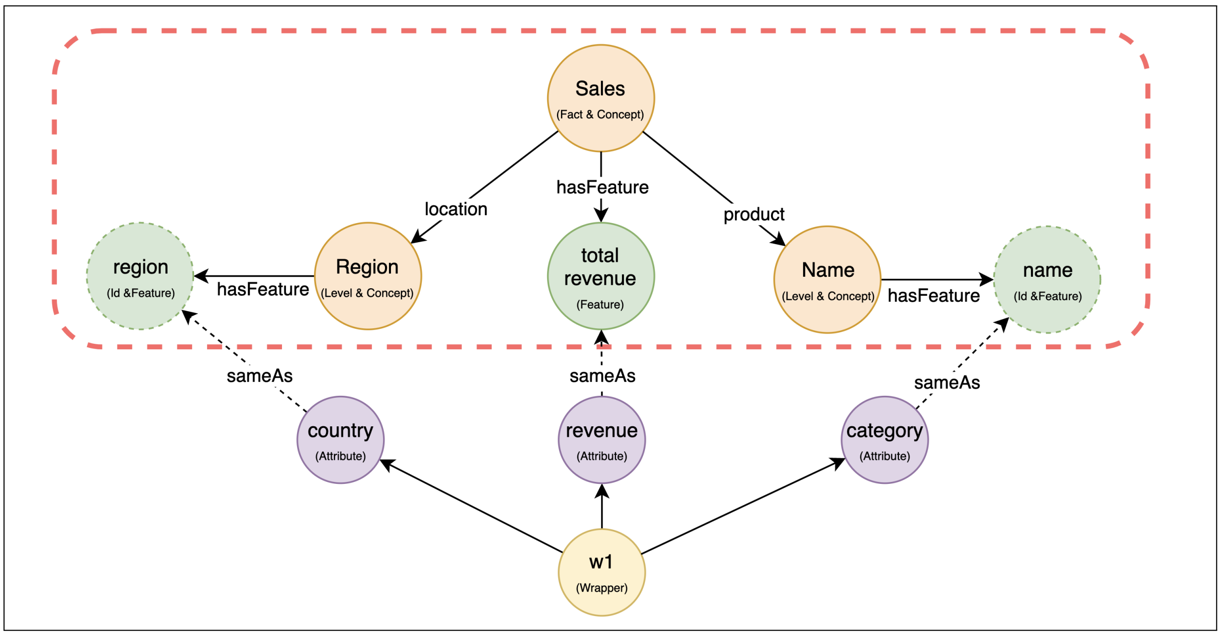

Figure 4.4: Most likely scenario with all the dimensional variable contained in a single wrapper.

opposite, it is depicted the scenario in which different dimensions of a given fact are represented by means of separated wrappers. In this case it is very important to adopt an identifier associated to the studied fact in order to correctly join separated dimensions having observations related to the same fact. The identifier *sale* will play this role.

4.3.1 Aggregation function

This section will show the intuition of how to include aggregation functions into GFDM. Aggregation functions are very important to describe aggregation semantic of a queried feature. In GFDM aggregations will be represented using the graph syntax, making

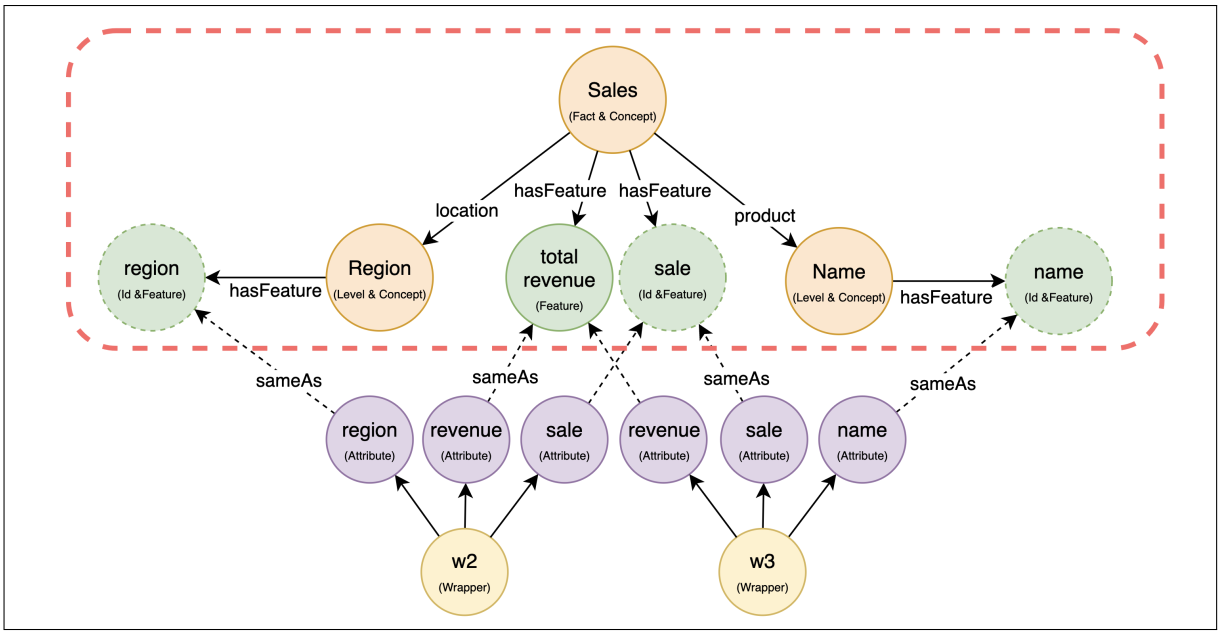

Figure 4.5: Scenario with disjointed wrappers.

pointing a vertex, with the semantic of an aggregation to a feature. In Figure 4.6 have been depicted a clear example, having the feature *number of unit* linked to an aggregating function *Sum* and the feature *total revenue* attached to an *Avg* aggregating function. It is very important having the aggregation functions represented into $\mathcal{G}$ and also $\mathcal{MG}$ since being the Roll-Up performed implicit, it is important for the algorithm knowing the semantic of the aggregation.

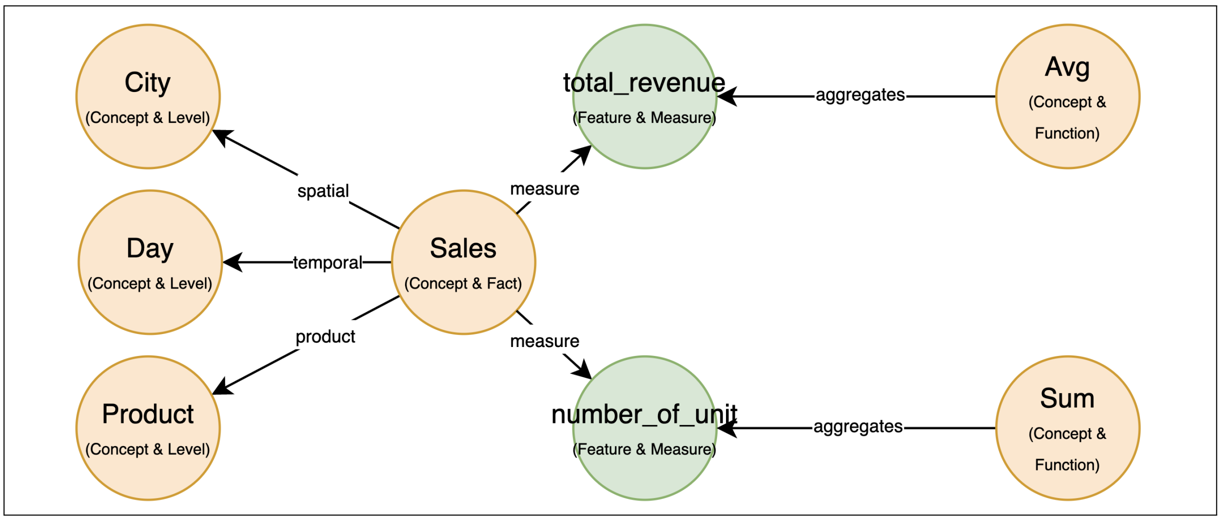

Figure 4.6: Example of aggregation functions *Average* and *Sum*.

Chapter 5

Description of the method

In this chapter it will be firstly described 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 will be very important since it 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 formalise the two model $\mathcal{MC}$ and $\mathcal{MG}$, showing and comparing the characteristics of each one. Let's start defining a multidimensional cube $\mathcal{MC}$ as

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

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 $\mathcal{L}$ behaving to $\mathcal{D}$ and the second describes all the functional dependencies between each level $\mathcal{L}$. Let's now introduce the formalisation of multidimensional graph model $\mathcal{MG}$ as follow.

$$\begin{aligned}\mathcal{MG} \equiv \langle \mathcal{F}, \mathcal{D}, \mathcal{L}, \mathcal{M}, \mathcal{AF}, dimensions : \mathcal{F} \rightarrow \langle \mathcal{D} \rangle, measures : \mathcal{F} \rightarrow \langle \mathcal{M} \rangle, levels : \mathcal{D} \rightarrow \langle \mathcal{L} \rangle,\\ aggregates : \mathcal{AF} \rightarrow \langle \mathcal{M} \rangle, partOf : \mathcal{L} \rightarrow \mathcal{L}', functions : \mathcal{F} \rightarrow \mathcal{AF} \rangle\end{aligned} \tag{5.2}$$

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*. Aggregation functions in $\mathcal{MC}$ model are just related to the kind of measures subject to the Roll-Up operation, therefore the aggregating function is established run time, while in $\mathcal{MG}$ model it 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. $\mathcal{MG}$ model can also describe more then one fact while $\mathcal{MC}$ model describes just one. That's why $\mathcal{MG}$ model have a function *dimensions* that for a given fact $\mathcal{F}$ gets a set of dimensions behaving to $\mathcal{F}$, and for the same reason there is also a set of measures related to $\mathcal{F}$ by the function *measures* and finally a function *functions* that shows all the $\mathcal{AF}$ associated to a given $\mathcal{F}$. Another important difference between $\mathcal{MC}$ and $\mathcal{MG}$ is that the latter, as $\mathcal{G}$, is clearly graph based and it deals with triples (e.g. it is described with a set of triples). From now on it will be introduced the triple notation behind each element of $\mathcal{MG}$ relying to the triple syntax introduced in section 4.1. The following definitions will refer to the Figure 5.1.

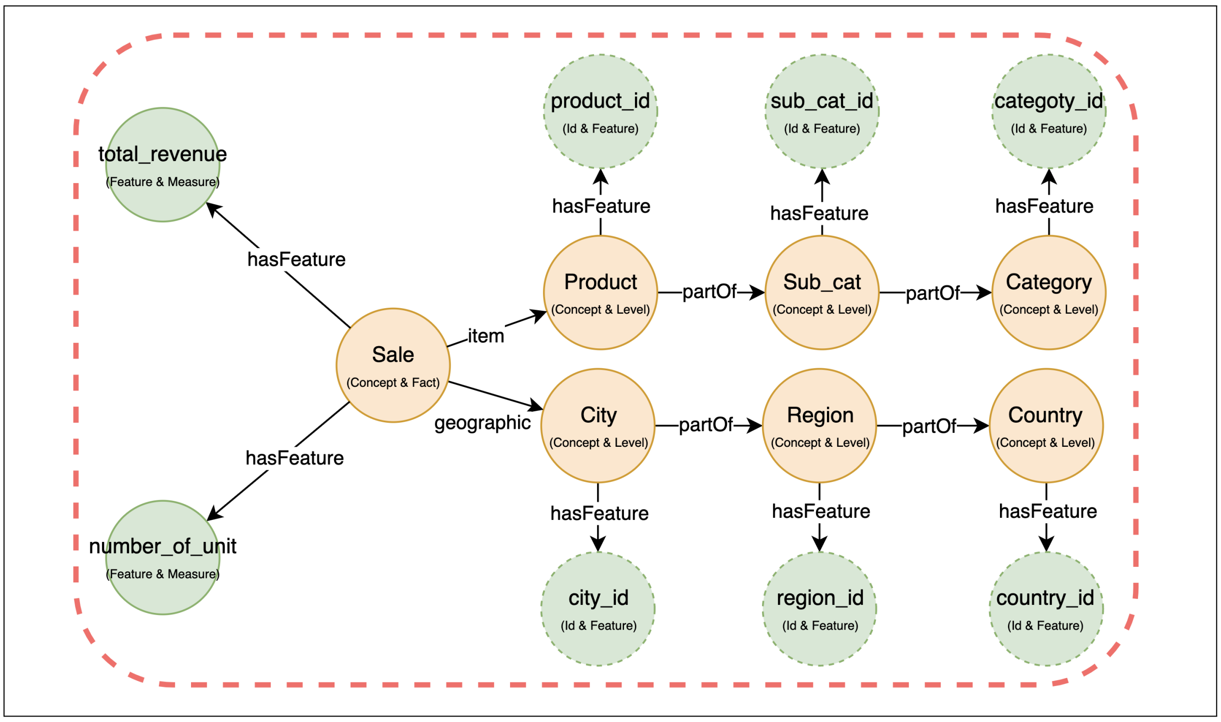

Figure 5.1: Running example, where the dashed red line is the query triggered.

Let's now formalise the triple notation of $\mathcal{MG}$ as the union of a set of fact $\langle\mathcal{F}\rangle$, a set of aggregation functions $\langle\mathcal{AF}\rangle$ and the $\mathcal{G}$ that is a set of triples as well.

$$\mathcal{MG}_{\langle S,P,O\rangle} \equiv \langle\mathcal{F}_{\langle S,P,O\rangle}\rangle \cup \mathcal{G}_{\langle S,P,O\rangle} \cup \langle\mathcal{AF}_{\langle S,P,O\rangle}\rangle \tag{5.3}$$

$$\mathcal{G}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{MG}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{I}_{\langle S,P,O\rangle} \tag{5.4}$$

Introduce now the triple notation for a dimension $\mathcal{D}$, given as the union of all the $partOf$ relationship triples in $\mathcal{D}$, referring to the example we have $\langle City, partOf, Region\rangle$

and $\langle Region, partOf, Country\rangle$ and the union of also all the triples $hasFeature$ defining each level F^{id}, then $\langle City, hasFeature, city\_id\rangle$, $\langle Region, hasFeature, region\_id\rangle$ and $\langle Country, hasFeature, country\_id\rangle$. Formally the triples involving $\mathcal{D}$ are given by the following statements.

$$\langle \mathcal{L}\rangle_{\mathcal{D}} \equiv levels(\mathcal{D}) \equiv \{\mathcal{L}_0, ..., \mathcal{L}_{size(levels(\mathcal{D}))-1}\} \tag{5.5}$$

$$hasFeature_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{size(\langle \mathcal{L}\rangle_{\mathcal{D}})-1}\langle \mathcal{L}_i, hasFeature, id(\mathcal{L}_i)\rangle \tag{5.6}$$

$$partOf_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{size(\langle \mathcal{L}\rangle_{\mathcal{D}})-2}\langle \mathcal{L}_i, partOf, partOf(\mathcal{L}_i)\rangle \tag{5.7}$$

$$\mathcal{D}_{\langle S,P,O\rangle} \equiv partOf_{\langle S,P,O\rangle} \cup hasFeature_{\langle S,P,O\rangle} \tag{5.8}$$

The triple notation for a fact $\mathcal{F}$ is defined as the union of all the dimensions behaving to $\mathcal{F}$, as $\mathcal{D}_{\langle S,P,O\rangle}$ (that have been defined right before) given by $dimensions(\mathcal{F})_{\langle S,P,O\rangle}$. For the fact *Sale* there will be all the triple related to the geographic and also the item dimensional space. Then will have to be considered the triple connecting the fact to each lower granularity level of dimensions, as $\langle Sale, item, Product\rangle$ and $\langle Sale, geographic, City\rangle$. Also all the triples defining the measures $\mathcal{M}$ that behave to the $\mathcal{F}$ have to be considered, such as $\langle Sale, hasFeature, total\_revenue\rangle$ and $\langle Sale, hasFeature, number\_of\_unit\rangle$. Since we want as goal to keep flexibility to the system, both the paths that links $\mathcal{F}$ to $\mathcal{M}$ and $\mathcal{D}$ don't have to be necessarily directed therefore to define the predicate P will be used regular expressions in the form of P^+ to define flexible paths. Then the triple notation of $\mathcal{F}$ can be stated as follow.

$$\langle \mathcal{D}\rangle_{\mathcal{F}} \equiv dimensions(\mathcal{F}) \equiv \{\mathcal{D}_0, ..., \mathcal{D}_{size(dimensions(\mathcal{F}))-1}\} \tag{5.9}$$

$$\langle \mathcal{M}\rangle_{\mathcal{F}} \equiv measures(\mathcal{F}) \equiv \{\mathcal{M}_0, ..., \mathcal{M}_{size(measures(\mathcal{F}))-1}\} \tag{5.10}$$

$$dimLabels_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{size(\langle \mathcal{D}\rangle_{\mathcal{F}})-1}\langle \mathcal{F}, P^+, lowerGranularityLevel(\mathcal{D}_i)\rangle \tag{5.11}$$

$$measuresLabels_{\langle S,P,O\rangle} \equiv \cup_{i=0}^{size(\langle \mathcal{M}\rangle_{\mathcal{F}})-1}\langle \mathcal{F}, P^+, \mathcal{M}_i\rangle \tag{5.12}$$

$$\mathcal{F}_{\langle S,P,O\rangle} \equiv \langle \mathcal{D}\rangle_{\mathcal{F}\langle S,P,O\rangle} \cup dimLabels_{\langle S,P,O\rangle} \cup measuresLabels_{\langle S,P,O\rangle} \tag{5.13}$$

It is also important to state the relationship between $\mathcal{D}_{\langle S,P,O\rangle}$, $\mathcal{F}_{\langle S,P,O\rangle}$ and $\mathcal{MG}_{\langle S,P,O\rangle}$ having the following rule.

$$\mathcal{D}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{F}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{MG}_{\langle S,P,O\rangle} \tag{5.14}$$

Let's finally see the triple notation for an aggregating function, having all the triple connecting the function to the referred measure. Consider that is possible having one $\mathcal{AF}$ linked to one or more $\mathcal{M}$. Referring to the running example let's suppose there are two aggregating function *sum* and *avg*, respectively aggregating *number_ of_ unit* and *total_ revenue*, then there will be the following two triples $\langle sum, aggregates, number\_of\_unit \rangle$ and $\langle avg, aggregates, total\_revenue \rangle$.

$$\langle \mathcal{AF} \rangle_{\mathcal{MG}} \equiv functions(\mathcal{MG}) \equiv \{\mathcal{AF}_0, ..., \mathcal{AF}_{size(functions(\mathcal{MG}))-1}\} \tag{5.15}$$

$$\langle \mathcal{M} \rangle_{\mathcal{AF}_i} \equiv aggregates(\mathcal{AF}_i) \equiv \{\mathcal{M}_0, ..., \mathcal{M}_{size(aggregates(\mathcal{AF}_i))-1}\} \tag{5.16}$$

$$\mathcal{AF}_{\langle S,P,O \rangle} \equiv \cup_{i=0}^{size(\langle \mathcal{AF} \rangle_{\mathcal{MG}})-1} \cup_{j=0}^{size(aggregates(\mathcal{AF}_i))-1} \langle \mathcal{AF}_i, aggregates, aggregates(\mathcal{AF}_i)_j \rangle \tag{5.17}$$

Let's consider now the querying for $\mathcal{MG}$, where φ is a pattern matching or visual query over $\mathcal{MG}$ and φ_{sql} is a query in SQL format. Introduce also the concept of dimension query $\varphi_{\mathcal{D}}$. A dimension query is a sub-graph of a given $\mathcal{D}$ in φ and its purpose is to allow to join dimensional data at low level granularity rising them to the required higher granularity asked by φ through W_{lut} (described in section 4.3). An example of dimension queries for the running example have been depicted in Figure 5.2 containing all the dimension queries for the product and geographical dimension. Let's

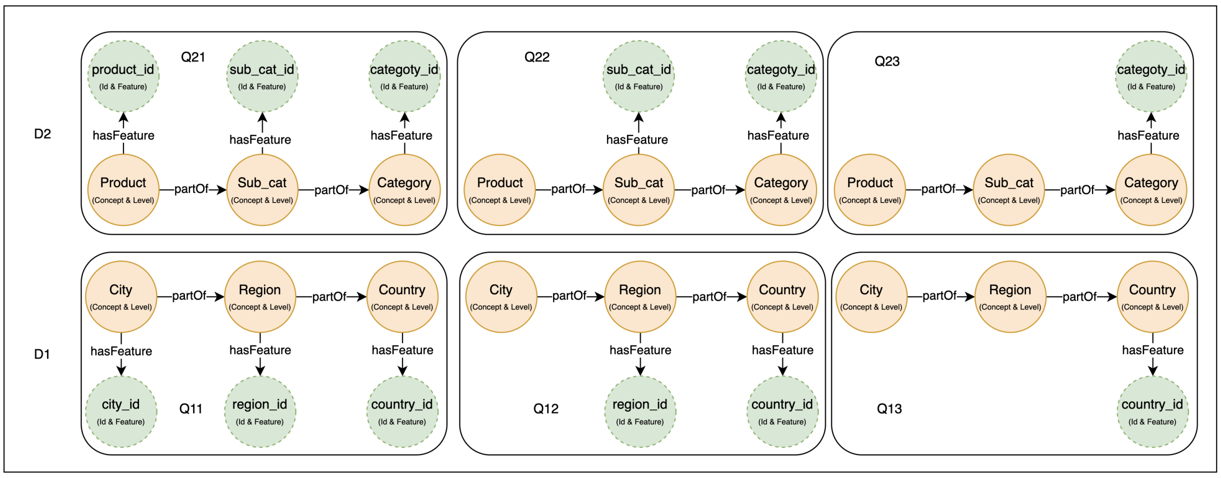
D2
Q21
product_id (Id & Feature)
sub_cat_id (Id & Feature)
categoty_id (Id & Feature)
hasFeature
Product (Concept & Level)
partOf
Sub_cat (Concept & Level)
Category (Concept & Level)
Q22
Q23
D1
City (Concept & Level)
Region (Concept & Level)
Country (Concept & Level)
city_id (Id & Feature)
region_id (Id & Feature)
country_id (Id & Feature)
Q11
Q12
Q13

Figure 5.2: Dimension queries $\varphi_{\mathcal{D}}$ for product and geographical dimension.

take as example the dimension queries for the geographical dimension *D1* in Figure 5.2. 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 (*Q13*), the data at *Region* granularity joined with the relative *Country* (*Q12*) and also the data at *City* granularity joined firstly with *Region* and finally with *Country* (*Q11*). The number of dimension query for each dimension and the total number of dimension queries will be the following.

$$\langle \mathcal{D} \rangle_{\varphi} \equiv dimensions(\varphi) \equiv \{\mathcal{D}_0, ..., \mathcal{D}_{size(dimensions(\varphi))-1}\} \tag{5.18}$$

$$cardinality(\varphi_{\mathcal{D}i}) \equiv size(levels(\mathcal{D}_i)) \tag{5.19}$$

$$cardinality(\varphi_{\mathcal{D}}) \equiv \sum_{i=0}^{size(dimensions(\varphi))-1} cardinality(\varphi_{\mathcal{D}i}) \tag{5.20}$$

A Roll-Up query is a sub-set of φ and it is a φ that has as dimension one possible combination of the dimension queries (e.g. one entry of the Cartesian Product of dimension queries). Referring to the Figure 5.3, there are nine possible combination of Roll-Up query, given by $\{Q21, Q22, Q23\} \times \{Q11, Q12, Q13\}$. The number of the Roll-Up queries for φ is given by the following formula.

$$\langle\mathcal{D}\rangle_{\varphi} \equiv dimensions(\varphi) \equiv \{\mathcal{D}_0, ..., \mathcal{D}_{size(dimensions(\varphi))-1}\} \tag{5.21}$$

$$cardinality(\varphi_{rollUp}) \equiv \prod_{i=0}^{size(\langle\mathcal{D}\rangle_{\varphi})-1} cardinality(\varphi_{\mathcal{D}i}) \tag{5.22}$$

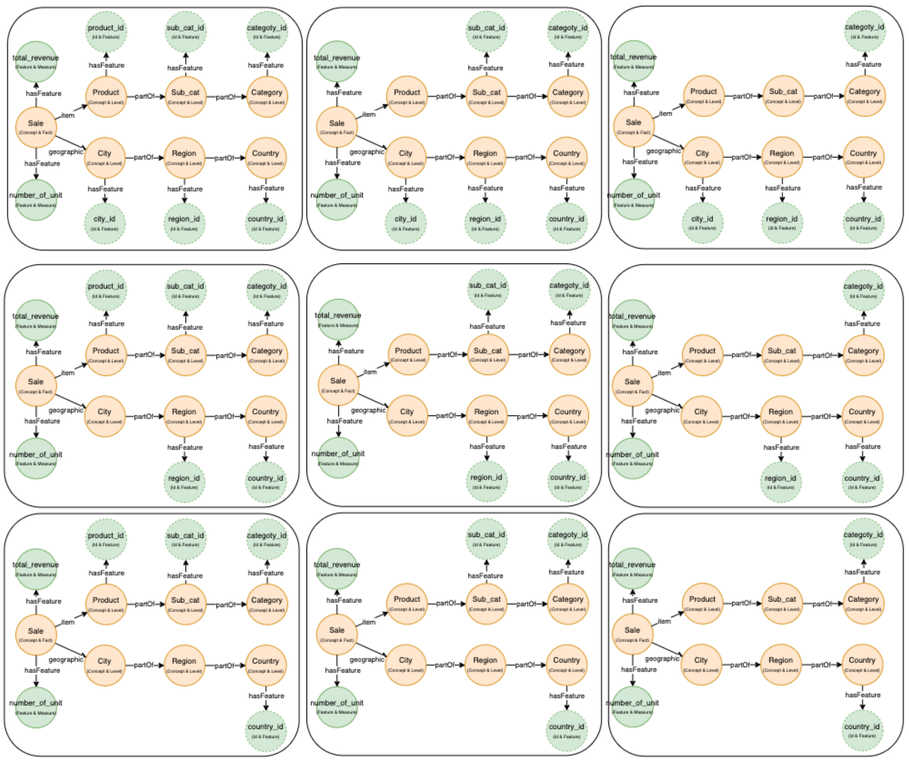

Figure 5.3: All the Roll-Up queries for φ.

Queries have also their triple notation and it is the following.

$$\varphi_{\mathcal{D}\langle S,P,O\rangle} \sqsubseteq \mathcal{D}_{\langle S,P,O\rangle} \tag{5.23}$$

$$\varphi_{\mathcal{D}\langle S,P,O\rangle} \sqsubseteq \varphi_{rollUp\langle S,P,O\rangle} \sqsubseteq \varphi_{\langle S,P,O\rangle} \sqsubseteq \mathcal{G}_{\langle S,P,O\rangle} \sqsubseteq \mathcal{MG}_{\langle S,P,O\rangle} \tag{5.24}$$

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;

$$\mathcal{L} \wedge \mathcal{AF} \sqsubseteq C \sqsubseteq V_{\mathcal{MG}} \tag{5.25}$$

$$\mathcal{M} \sqsubseteq F \sqsubseteq V_{\mathcal{MG}} \tag{5.26}$$

$$partOf \wedge aggregates \sqsubseteq E_{\mathcal{MG}} \tag{5.27}$$

Let's finally also introduce some simple functions that are *traverses* over $\mathcal{MG}$ that will be very useful in the next section for the execution of the algorithm; $measures(\mathcal{G}^{+})$, $features(\mathcal{G}^{+})$, $levels(\mathcal{G}^{+})$, $dimensions(\mathcal{G}^{+})$ are respectively all the measures $\mathcal{M}$, features F, levels $\mathcal{L}$ and dimensions $\mathcal{D}$ in a given $\mathcal{G}$, that means that those function can be both called on φ and $\mathcal{MG}$. Also $id(C)$ are all the F^{id} for a given C. 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 linked to an F^{id}). Finally the function $aggregatingFunctions(\mathcal{MG})$ will describe all the aggregating functions $\mathcal{AF}$ in $\mathcal{MG}$.

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. The *traverses* defined before will be massively used to simplify the understandability of the algorithms but their implementation will not be detailed since it is very related to the implementation technique. The main flow have been depicted in the Algorithm 1, where it is clear which are the main steps of the algorithm and how they are related to each other. The algorithm will take as input the multidimensional graph $\mathcal{MG}$, the global query φ and the function *rewrite* that will execute underneath the rewriting algorithm and finally it will generate a SQL view φ_{sql}. The function *rewrite* takes as argument two queries, φ and φ_{min} executing φ with the respect of the *minimal* property according to φ_{min}. As output *implicitRollUp* will generate a complex SQL query φ_{sql} given by the union of multiple call of *rewrite* if φ has an aggregation semantic. If any aggregation semantic is detected in φ the output of the algorithm will be just the execution of *rewrite* over φ.

Let's now detail all the steps of the algorithm starting defining the first procedure we encounter. 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

Algorithm 1 implicitRollUp

```
Input: $\mathcal{MG}, \varphi, rewrite : (\varphi, \varphi_{min}) \rightarrow \varphi_{sql}$
Output: $\varphi_{sql}$
 1: if $canAggragate(\varphi)$ then
 2:     $\langle\mathcal{L}\rangle \leftarrow extractGroupByClauses(\varphi)$
 3:     $\langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle \leftarrow extractAggregationClauses(\mathcal{MG}, \varphi)$
 4:     $\langle\mathcal{L}\rangle_{sql} \leftarrow parseGroupByClauses(\langle\mathcal{L}\rangle)$
 5:     $\langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql} \leftarrow parseAggregationClauses(\langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle)$
 6:     $\langle\mathcal{D}, \langle\varphi_{\mathcal{D}}\rangle\rangle \leftarrow generateDimensionQueries(\varphi, \mathcal{MG})$
 7:     $\langle\varphi_{rollUp}\rangle \leftarrow crossProductUnion(\varphi, \langle\mathcal{D}, \langle\mathcal{D}_{\langle S,P,O\rangle}\rangle\rangle)$
 8:     $\langle\varphi_{sql}\rangle \leftarrow rewriteAll(rewrite, prune(\varphi))(\langle\varphi_{rollUp}\rangle)$
 9:     $\varphi_{sql} \leftarrow makeSqlQuery(\langle\varphi_{sql}\rangle, \langle\mathcal{L}\rangle_{sql}, \langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql})$
10:     return $\varphi_{sql}$
11: else
12:     $\varphi_{sql} \leftarrow rewrite(\varphi, \varphi)$
13:     return $\varphi_{sql}$
14: end if
```

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 hit at least a dimension $\mathcal{D}$ (e.g. there should be at least a group by clause).
- The query should hit at least a measure $\mathcal{M}$.
- The query shouldn't hit other features than measures and levels identifiers.

Algorithm 2 canAggregate

```
Input: $\varphi$
Output: $Boolean$
 1: return $extractGroupByClauses(\varphi) \neq \emptyset \wedge measures(\varphi) \neq \emptyset \wedge (features(\varphi) -$
    $id(levels(dimensions(\varphi))) - measures(\varphi)) \equiv \emptyset$
```

After understanding if it is possible to have implicit Roll-Up or not, two important algorithm will be executed over the global query φ in order to extract the sql group by semantic; 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 over φ given as input, checking for each dimension in φ which level is the higher granularity one that has also F^{id} (e.g. holds the triple $\langle \mathcal{L}, hasFeature, id(\mathcal{L}) \rangle$). As output 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 to put 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 3 extractGroupByClauses

```
Input: $\varphi$
Output: $\langle \mathcal{L} \rangle$
1: $\langle \mathcal{D} \rangle \leftarrow dimensions(\varphi)$
2: $\langle \mathcal{L} \rangle \leftarrow$
3: for each $\mathcal{D}_i \in \langle \mathcal{D} \rangle$ do
4:     $\mathcal{L} \leftarrow higherGranularityLevel(\mathcal{D}_i)$
5: end for
6: return $\langle \mathcal{L} \rangle$
```

Algorithm 4, *extractAggregationClauses* will be used as opposite to extract the semantic of aggregation 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 in $\mathcal{MG}$ and then intersecting all the measure behaving to the relationship $aggregates(\mathcal{AF})$ to all the queried measures and keeping the solution having a non empty set of measures. As for the algorithm 4 there is a parsing function as well, *parseAggregationClauses* that will generate a formatted string (comma separated) for the aggregation part of the SQL query into the "SELECT" statement.

Another important step of the *implicitRollUp* algorithm will be performed by algorithm 5, *generateDimensionQueries*. The input of the algorithm then will be φ and the output will be a set of set of dimension queries $\langle \langle \varphi_{\mathcal{D}} \rangle \rangle$, each set associated to a dimension $\mathcal{D}$. An example of the output for this algorithm is depicted in Figure 5.2, showing $\langle \langle \varphi_{\mathcal{D}} \rangle \rangle$ for the running example. The main idea of the algorithm is to generate all the dimension queries starting for each $dimensions(\varphi)$ and removing incrementally all the triple $\langle \mathcal{L}_i, hasFeature, id(\mathcal{L}_i) \rangle$ generating step by step each dimension query for each dimension.

Once generated all the dimension query for φ, it will be important to generate all the Roll-Up queries φ_{rollUp}. A Roll-Up query is a sub-graph of φ, generated by the union of φ, without the dimensions, and one entry of the cross product of $\langle \langle \varphi_{\mathcal{D}} \rangle \rangle$. This operation

Algorithm 4 extractAggregationClauses

```
Input: $\varphi, \mathcal{MG}$
Output: $\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle$
 1: $\langle \mathcal{AF} \rangle \leftarrow aggregatingFunctions(\mathcal{MG})$
 2: $\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle \leftarrow$
 3: for each $\mathcal{AF}_i \in \langle \mathcal{AF} \rangle$ do
 4:     $\langle \mathcal{M} \rangle \leftarrow aggregates(\mathcal{AF}_i) \cap measures(\varphi)$
 5:     if $\langle \mathcal{M} \rangle \equiv \emptyset$ then
 6:         $\emptyset$
 7:     else
 8:         $(\mathcal{AF}_i, \langle \mathcal{M} \rangle)$
 9:     end if
10: end for
11: return $\langle \mathcal{AF}, \langle \mathcal{M} \rangle \rangle$
```

Algorithm 5 generateDimensionQueries

```
Input: $\varphi$
Output: $\langle \langle \varphi_{\mathcal{D}} \rangle \rangle$
 1: $\langle \mathcal{D} \rangle \leftarrow dimensions(\varphi)$
 2: $\langle \langle \varphi_{\mathcal{D}} \rangle \rangle \leftarrow$
 3: for each $\mathcal{D}_i \in \langle \mathcal{D} \rangle$ do
 4:     $\langle \varphi_{\mathcal{D}} \rangle \leftarrow$
 5:     for each $\mathcal{L}_i \in levels(\mathcal{D})$ do
 6:         $\varphi_{\mathcal{D}\langle S,P,O \rangle} \leftarrow \mathcal{D}_{i\langle S,P,O \rangle} - \{\cup_{j=-1}^{i-1} \langle \mathcal{L}_j, hasFeature, id(\mathcal{L}_j) \rangle\}$
 7:     end for
 8: end for
 9: return $\langle \langle \varphi_{\mathcal{D}} \rangle \rangle$
```

is performed by algorithm 6, having as parameter φ and all the dimension queries $\langle \langle \varphi_{\mathcal{D}} \rangle \rangle$, and giving as output the Roll-Up queries $\langle \varphi_{rollUp} \rangle$. An example of Roll-Up queries is depicted in Figure 5.3.

Finally once generated all the Roll-Up queries it will be possible to use the rewriting algorithm to translate in relational algebra expressions all the queries and finally convert them in SQL format. The function *rewriteAll* that will perform multiple call of the function *rewrite* will be used to perform this operation over φ_{rollUp}, checking the *minimal* property for φ, that will have to be "pruned" removing all the triple $\langle \mathcal{L}, hasFeature, id(\mathcal{L}) \rangle$ rather then in the higher granularity level of each dimension. This step is very important for the check of the *minimal* property. Algorithm 7 is the responsible of this operation.

Algorithm 6 crossProductUnion

```
Input: $\varphi, \langle\langle\varphi_{\mathcal{D}}\rangle\rangle$
Output: $\langle\varphi_{rollUp}\rangle$
1: $\varphi\prime \leftarrow \varphi_{\langle S,P,O\rangle} - dimensions(\varphi)_{\langle S,P,O\rangle}$
2: $\langle\varphi_{rollUp}\rangle \leftarrow \cup_{\times i=0}^{size(\langle\langle\varphi_{\mathcal{D}}\rangle\rangle)-1} \langle\varphi_{\mathcal{D}}\rangle_{i\langle S,P,O\rangle}$
3: $\langle\varphi_{rollUp}\rangle \leftarrow$
4: for each $r \in \langle\varphi_{rollUp}\rangle$ do
5:     $\varphi_{rollUp} \leftarrow r_{\langle S,P,O\rangle} \cup \varphi'_{\langle S,P,O\rangle}$
6: end for
7: return $\langle\varphi_{rollUp}\rangle$
```

Algorithm 7 rewriteAll

```
Input: $rewrite : (\varphi, \varphi_{min}) \to \varphi_{sql}, \varphi, \langle\varphi_{rollUp}\rangle$
Output: $\langle\varphi_{sql}\rangle$
1: for each $\mathcal{D}_i \in dimensions(\varphi)$ do
2:     $\varphi \leftarrow \varphi_{\langle S,P,O\rangle} - \{\cup_{j=0}^{size(levels(\mathcal{D}_i))-1} \langle\mathcal{L}_{levels(\mathcal{D}_i)_j}, hasFeature, id(\mathcal{L}_{levels(\mathcal{D}_i)_j})\rangle\}$
3: end for
4: $\langle\varphi_{sql}\rangle \leftarrow$
5: for each $\varphi_{rollUp} \in \langle\varphi_{rollUp}\rangle$ do
6:     $\varphi_{sql} \leftarrow rewrite(\varphi_{rollUp}, \varphi)$
7: end for
8: return $\langle\varphi_{sql}\rangle$
```

The algorithm 8 will transform each view $\langle\varphi_{sql}\rangle$ wrapping each of them with a group by clause, after it will make the SQL union operation and then finally it will wrap up the final result again with a final group by statement, generating the result SQL query.

Algorithm 8 makeSqlQuery

```
Input: $\langle\varphi_{sql}\rangle, \langle\mathcal{L}\rangle_{sql}, \langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql}, wrapGroupBy : (\varphi_{sql}, \langle\mathcal{L}\rangle_{sql}, \langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql}) \rightarrow \varphi_{sql}$
Output: $\varphi_{sql}$
1: $\langle\varphi_{sql}\rangle \leftarrow$
2: for each $q_i \in \langle\varphi_{sql}\rangle$ do
3:     $\varphi_{sql} \leftarrow wrapGroupBy(q_i, \langle\mathcal{L}\rangle_{sql}, \langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql})$
4: end for
5: $\varphi_{sql} \leftarrow fold(\langle\varphi_{sql}\rangle)("")\{\varphi^{'}_{sql} + "UNION" + \varphi^{''}_{sql}\}$
6: $\varphi_{sql} \leftarrow wrapGroupBy(\varphi_{sql}, \langle\mathcal{L}\rangle_{sql}, \langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql})$
```

Algorithm 9 wrapGroupBy

```
Input: $\langle\varphi_{sql}\rangle, \langle\mathcal{L}\rangle_{sql}, \langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql}$
Output: $\varphi_{sql}$
1: $"SELECT" + \langle\mathcal{L}\rangle_{sql} + \langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql} + \varphi_{sql} + "GROUP\ BY" + \langle\mathcal{L}\rangle_{sql} + \langle\mathcal{AF}, \langle\mathcal{M}\rangle\rangle_{sql}$
```

Chapter 6

Implementation

In this section 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, a new programmatic graph data model have been defined from scratch to overcome few limitations given from 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 this procedure. A nice domain specific language, that will be detailed in section 6.3, have also been developed in order to define each testing scenario very easily. Once defined a domain specific language, a new model and *traverses* over the model itself to generate $\mathcal{I}$ configuration file, 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 have 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 organisation 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 for first make clear that the organisation 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 behaviour. 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 extension. 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.

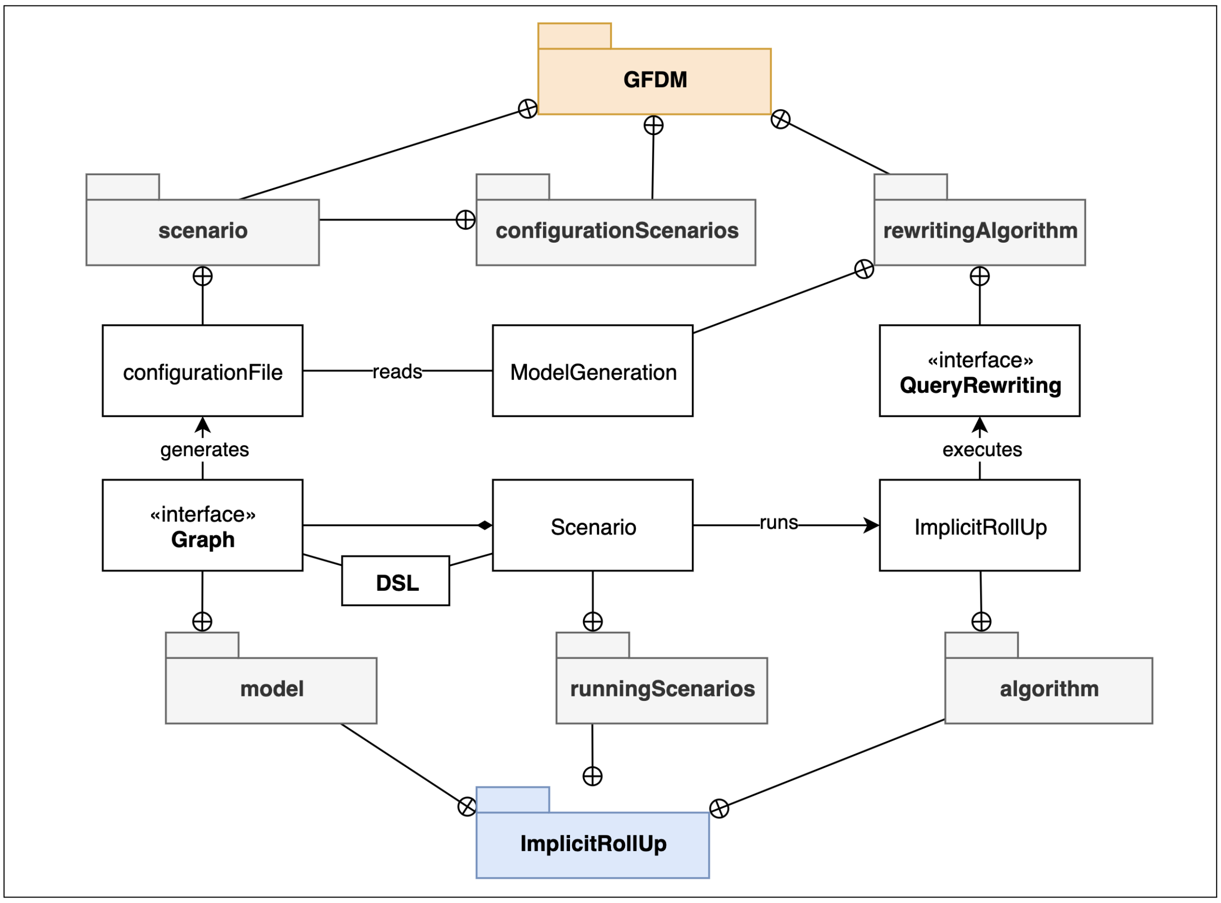

Figure 6.1: UML package diagram.

6.2 Execution flow

The sequence diagram in figure 6.2 is describing 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 in a SQL expression by the module *Sql*.

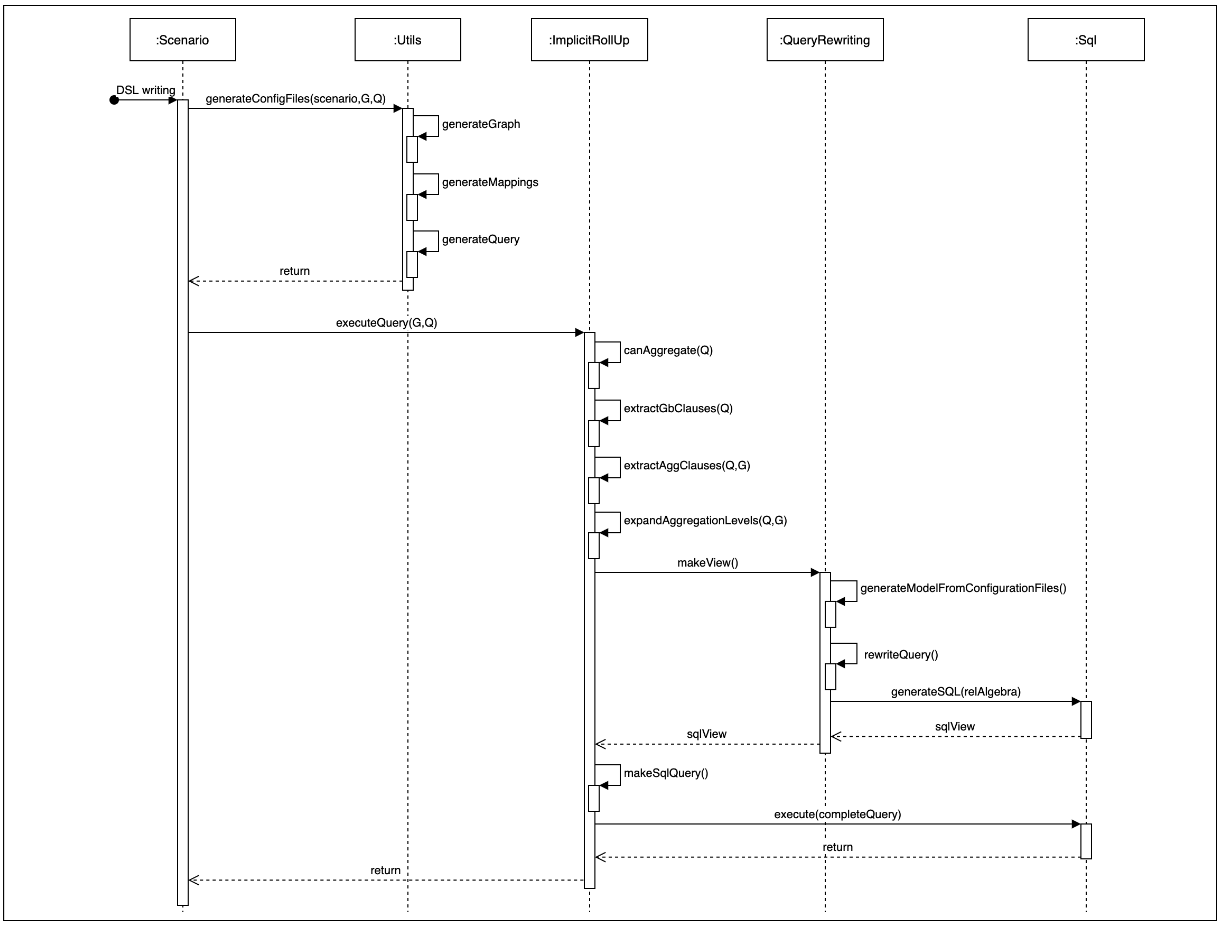

Figure 6.2: UML sequence diagram.

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 kind 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's start with a code snippet that will show how to build $\mathcal{I}$. On the top side of the code will be defined all the feature 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: My Caption

```
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: My Caption

```
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 dealing with the generation of a scenario an 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: My Caption

```
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: My Caption

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

Chapter 7

Experimentation

7.1 Automatic test

7.2 Validation test

Description of the experimental datasets and experiments conducted. Report results and interpret them.

The only factor that matters for performances is the size of the graph.

Chapter 8

Conclusions and future works

Chapter 9

Problems, ***to drop***

- Aggregation functions are separate form $\mathcal{MG}$ in my implementation, I cheat passing to the implicit Roll-Up algorithm a set of aggregation function, that are actually graphs.
- How to chose the right id to expand for a Level if there are multiple... I may need AggregationId but also not (I can cut form the rewriting actually)
- partOf relationship, Dimensions and Facts are very relaxed in my work because are not necessary for the algorithm, but they may be very useful for understanding the model and for visualisation purpose and managing for future works.
- disjointedness of data (assume disjointedness) and compatibility (assume compatibility)
- Data set showcases (assume not to have all the hierarchy filled in one variable), otherwise we may need to enrich the model
- The algorithm is faked a bit in section 5.2 to make it more understandable (eg hide that GFDMworks with configuration files)
- include wrapper name in the result is not implemented

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