Data & Knowledge Engineering 69 (2010) 917–942

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Automatic validation of requirements to support multidimensional design

Oscar Romero [a,*], Alberto Abelló [b]

[a] Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya (BarcelonaTech), c/ Jordi Girona no 1-3, edifici Omega, despatx 117, 08034, Barcelona, Spain

[b] Departament d'Enginyeria de Serveis i Sistemes d'Informació, Universitat Politècnica de Catalunya (BarcelonaTech), c/ Jordi Girona no 1-3, edifici Omega, despatx 125, 08034, Barcelona, Spain

ARTICLE INFO

Article history:
Received 7 October 2008
Received in revised form 9 March 2010
Accepted 11 March 2010
Available online 27 March 2010

Keywords:
Data warehouse
OLAP
Multidimensional design
Requirements
Design by examples

ABSTRACT

It is widely accepted that the conceptual schema of a data warehouse must be structured according to the *multidimensional* model. Moreover, it has been suggested that the ideal scenario for deriving the multidimensional conceptual schema of the data warehouse would consist of a hybrid approach (i.e., a combination of data-driven and requirement-driven paradigms). Thus, the resulting multidimensional schema would satisfy the end-user requirements and would be conciliated with the data sources. Most current methods follow either a data-driven or requirement-driven paradigm and only a few use a hybrid approach. Furthermore, hybrid methods are unbalanced and do not benefit from all of the advantages brought by each paradigm.

In this paper we present our approach for multidimensional design. The most relevant step in our framework is *Multidimensional Design by Examples* (MDBE), which is a novel method for deriving multidimensional conceptual schemas from relational sources according to end-user requirements. MDBE introduces several advantages over previous approaches, which can be summarized as three main contributions. (i) The MDBE method is a fully automatic approach that handles and analyzes the end-user requirements automatically. (ii) Unlike data-driven methods, we focus on data of interest to the end-user. However, the user may not be aware of all the potential analyses of the data sources and, in contrast to requirement-driven approaches, MDBE can propose new multidimensional knowledge related to concepts already queried by the user. (iii) Finally, MDBE proposes meaningful multidimensional schemas derived from a validation process. Therefore, the proposed schemas are sound and meaningful.

1. Introduction

Data warehousing systems were designed to support decision-making within organizations. These systems homogenize and integrate data in a huge repository (i.e., the *data warehouse*) to create a single, detailed representation of the organization from which relevant knowledge can be extracted and applied in the organization's decision-making processes.

It is widely accepted that the conceptual schema of a data warehouse must be structured according to the multidimensional model. The multidimensional (MD) conceptual view of data is distinguished by the *fact/dimension* dichotomy and represents data as if placed in an n-dimensional space, which facilitates the interpretation and analysis of data in terms of *facts* (the subjects of analysis) and *dimensions* showing the different perspectives from which a subject can be analyzed.

Since a data warehouse is the result of homogenizing and integrating relevant data in a single, detailed view, it is assumed that the MD conceptual schema of the data warehouse must be derived from the organization's data source schemas. Traditionally, this process has been performed manually, but automation is essential as it removes the dependency on an expert's ability to properly

* Corresponding author. Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya (BarcelonaTech), c/ Jordi Girona no 1-3, edifici Omega, despatx 117, 08034, Barcelona, Spain.
E-mail addresses: oromero@lsi.upc.edu (O. Romero), aabello@essi.upc.edu (A. Abelló).

0169-023X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2010.03.006

apply the method chosen and the need to analyze the data sources, which is a tedious and time-consuming task (which can be unfeasible when working with large databases). In recent years, several approaches have been proposed for automating this process, most of which follow a data-driven model in which data sources are analyzed thoroughly to derive the data warehouse schema in a reengineering process that overlooks the end-user MD requirements. However, as discussed in [36], a requirement analysis phase is crucial in ensuring that end-user's needs and expectations are met. Otherwise, end-users may become frustrated as they would not be able to analyze data of interest to them, which would result in the failure of the whole system. The literature contains several requirement-driven methods, but all of them must be carried out manually. Automating requirement-driven approaches would require the formalization of end-user requirements (i.e., translating them into a computer understandable language), whereas current methods handle requirements that are mainly stated in languages (such as natural language) which lack the required degree of formalization.

As discussed in the literature [24,36], the ideal scenario for deriving the data warehouse conceptual schema would consist of a hybrid approach (i.e., a combination of data-driven and requirement-driven paradigms). Therefore, the resulting MD schema would satisfy end-user requirements and be conciliated with the data sources simultaneously (i.e., capturing the analytical potential in the data sources and able to be populated with data within the organization). However, current automatable methods follow a fully data-driven approach, and current requirement-driven approaches are not automatable because they tend to work with requirements at a high level of abstraction.

In this paper we present a largely automated approach for supporting MD design based principally on *Multidimensional Design By Examples* (MDBE), which is an automated method conciliating both types of paradigms. Unlike other hybrid approaches, MDBE does not carry out two well-differentiated phases (i.e., data-driven and requirement-driven) that need to be conciliated a posteriori but instead performs both phases simultaneously. Consequently, each paradigm benefits from feedback obtained by the other, and eventually MDBE is able to derive more valuable information than approaches in which the two phases are carried out sequentially (a detailed list of the main advantages of MDBE over previous approaches is given in Section 2).

In our approach we derive MD *conceptual* schemas from *relational sources* according to end-user *requirements*. There are two steps: requirement formalization and the MDBE method (see Fig. 1). As in previous requirement-driven methods (or requirement-driven stages within hybrid methods), a prior requirements elicitation step is required. However, our approach is not based on a step-by-step manual process in which the requirements and data sources that will eventually derive the MD schema are analyzed in details, but rather on a largely automatable approach.

Requirements are typically expressed at a high level of abstraction and need to be formalized prior to automation of the analysis step. In our framework, requirements are expressed as SQL queries over the relational data sources (i.e., at the logical level over the data sources). SQL queries provide a clearly defined structure that will facilitate full automation of the MDBE method (the second step in our approach). Although requirement formalization must be performed manually, translating requirements into SQL queries requires considerably less effort than carrying out any of the step-by-step requirement-driven approaches in current use (see Section 2 for further discussion of this issue). In our approach we have reduced the amount of manual operations as much as possible (i.e., removing ambiguous semantics by formalizing the requirements) and delegated most of the design workload to the MDBE method, which will use the semantics captured in the requirements and the data sources to automate the rest of the process.

The inputs of the MDBE method are the end-user information requirements (expressed as SQL queries) and the *integrated* logical model of the data sources. The output is a *constellation schema* [18] (i.e., a conceptual schema for each fact identified) derived from the data sources and capable of retrieving data requested in the input requirements. Briefly, MDBE validates whether each input SQL query represents a valid MD query (i.e., if the query retrieves data that can be analyzed from a MD perspective). Note that we translate requirements into regular SQL queries over the transactional data sources and do not require a specific translation that would make MD sense. MDBE analyzes each input SQL query to validate whether it represents an MD requirement and notifies if it is able to derive at least one *multidimensional schema* that can retrieve data requested in the SQL query. Conciliation of the schemas proposed for each query produces the output constellation schema.

To illustrate a practical application of our approach we now introduce the TPC benchmark H (TPC-H) [1]. TPC-H is a decision support benchmark that introduces a relational database logical schema (see Fig. 2) and a suite of 22 business-oriented queries.

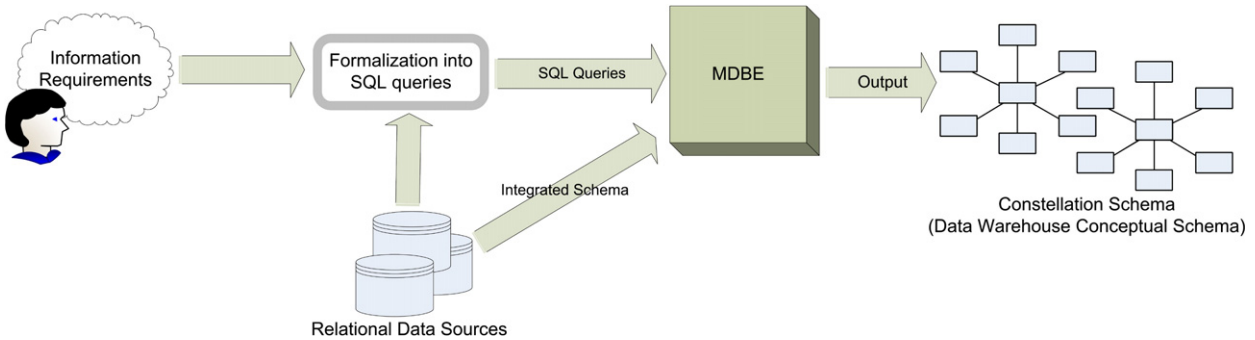

Fig. 1. Overview of our approach.

This benchmark was designed to represent a real-world information system, so its database schema and queries have been chosen for their industry-wide relevance. The database schema presented portrays the activity of a wholesale supplier. TPC-H does not represent the activity of a particular business sector but rather that of any industry in which a product needs to be managed, sold or distributed internationally (e.g., car rental, food distribution, parts, suppliers, etc.). Queries presented in the benchmark have been given a realistic context and were chosen to be representative and to answer to real-world questions. The queries are defined by the following components:

- A high-level description of the business question, which illustrates the context in which the query could be used. For example, "*report the amount of business that was billed, shipped, and returned*" (Q1), "*list the revenue volume done through local suppliers*" (Q5), "*determine the value of goods shipped between certain nations to help in the re-negotiation of shipping contracts*" (Q7) or "*identify customers who might be having problems with the parts that are shipped to them*" (Q10).
- The functional query definition, which uses the SQL-92 language to define the function to be performed by the query. As an example, business query #5 (Q5) is expressed in SQL as:

```
SELECT n_name, sum(l_extendedprice*(1-l_discount)) as revenue
FROM customer, orders, lineitem, supplier, nation, region
WHERE c_custkey=o_custkey and l_orderkey=o_orderkey and
l_suppkey=s_suppkey and c_nation key=n_nationkey and
s_nationkey=n_nationkey and n_region key=r_regionkey and
r_name='[REGION]' and o_orderdate>='[DATE]' and
o_orderdate<'[DATE]'+'1' year
GROUP BY n_name
ORDERBY revenue desc;
```

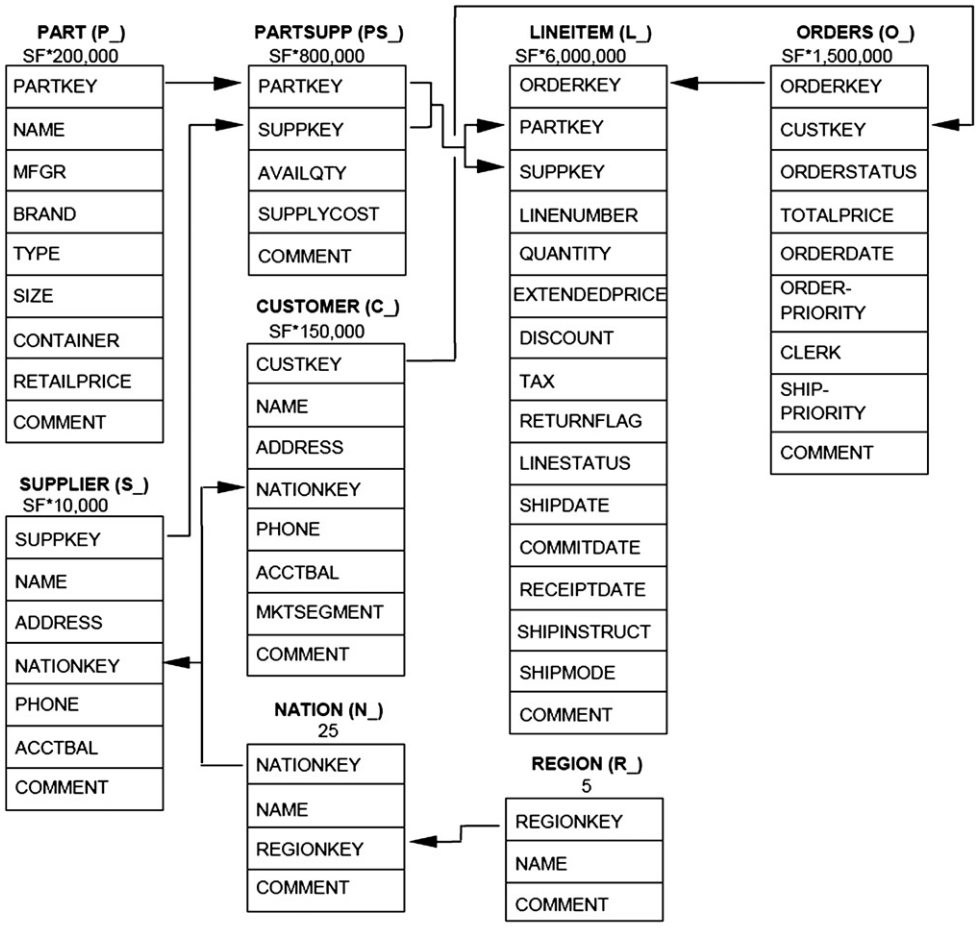

Fig. 2. TPC-H relational schema.

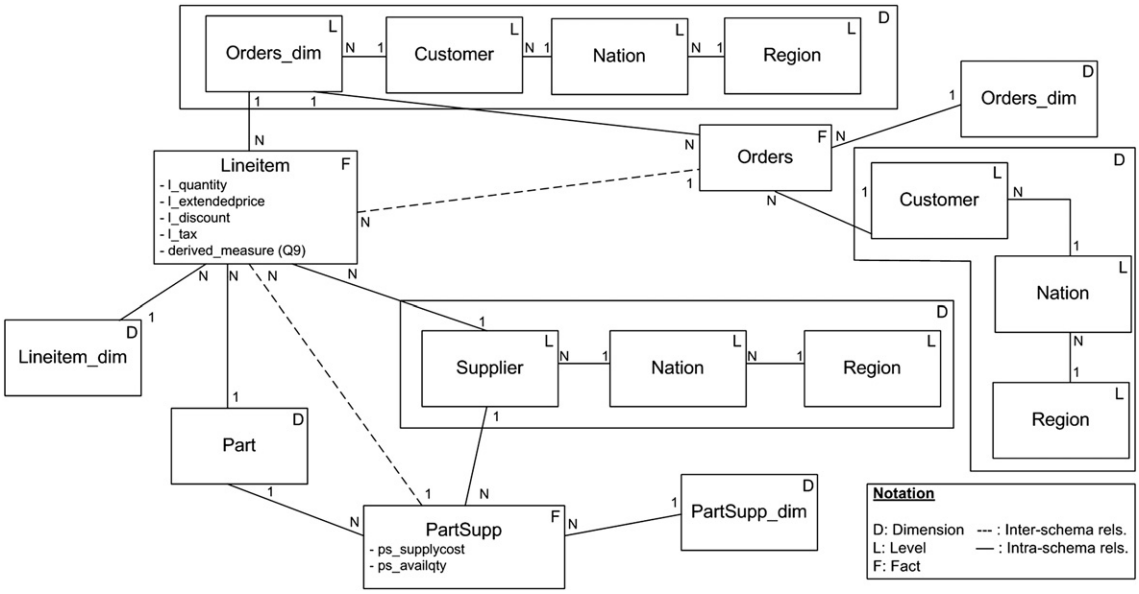

Fig. 3. Constellation schema derived from the TPC-H benchmark case study.

TPC-H is a decision support benchmark, and it would make sense to propose a MD schema (i.e., develop a data warehouse) for analyzing this data. This has already been considered in data warehouse research. One example is the *Star-Schema benchmark* (SSB) [27], which was devised from the TPC-H benchmark and introduces an MD schema derived manually from the TPC-H relational schema. In this paper we use TPC-H to demonstrate how to derive the MD schema with our approach. We then confirm the reliability of the result obtained by comparing it to that of the MD schema proposed in SSB (see Section 5.2).

We consider the TPC-H relational schema as the integrated relational schema of the data sources, and the high-level descriptions of the business queries as the end-user *information requirements* (gathered in the requirements elicitation step). TPC-H also provides the SQL query for each end-user requirement (i.e., the requirement formalization required in our approach). Consequently, it provides all of the inputs needed to launch the MDBE method. Eventually, MDBE will generate a set of MD schemas (see Fig. 3) from the data sources (in this case, the TPC-H logical schema) that meet the end-user requirements (the 22 TPC-H SQL queries).

In summary, MDBE has three main benefits: (i) It is a fully automatic approach that handles and analyzes the end-user requirements automatically. (ii) Unlike data-driven methods, we focus on data of interest to the end-user. However, the user may not be aware of all the potential analyses of the data sources [13,24,36] and, in contrast to requirement-driven approaches, MDBE can propose new MD knowledge related to concepts already queried by the user. (iii) Finally, MDBE proposes meaningful MD schemas derived from a validation process. Input queries are validated to determine whether they make MD sense, so the schemas proposed are sound and meaningful. As such, MDBE could be used as a validation tool for MD requirements in addition to its design function.

In this paper we provide a detailed description of our approach. In Section 2 we discuss related work in the literature and highlight the main advantages of MDBE over alternative approaches. In Section 3 we present our framework and discuss the foundations of our approach in depth. Section 4 focuses on the core of our approach, the MDBE method. Finally, in Section 5 we present the statistics of our approach over the TPC-H case study and compare the results with those presented in the Star-Schema Benchmark.

2. Related work and main contributions

The literature contains descriptions of several methods for deriving conceptual MD schema from the data sources. However, most of them must be carried out manually (following a step-by-step guide) and only a few automate the process. According to Winter et al. [36], these methods can mainly be classified in a *supply-driven* or *demand-driven* framework:

- Supply-driven: These approaches start from a detailed analysis of the data sources to determine the MD concepts in a reengineering process. Many methods presented in the literature follow this paradigm, including [14,17,26], among others.
- Demand-driven: These approaches focus on determining the end-user MD requirements (as typically performed in other information systems) to later map them on to data sources, as described in [13,16,30,36], among others.
- Hybrid approaches: Some authors have proposed combining the two approaches to design the data warehouse from the data sources taking into account end-user requirements. Examples of methods combining the two approaches can be found in [9,13,24,28], among others.

Most of these approaches do not automate (most of) the definition process and simply give a set of steps (i.e., guidelines) to be followed by an expert for deriving the MD conceptual schema. Most of the methods use different patterns or heuristics to discover concepts likely to play an MD role, so experts must have access to well-documented data sources (such as up-to-date conceptual schemas)

in order to implement these approaches. However, in a real organization, documentation on data sources may be incomplete, incorrect or even non-existent [14], and it would be difficult for a non-expert designer to follow these guidelines. To solve these problems, automatable methods [13,14,17,24,28] work directly over relational database logical schemas (i.e., getting up-to-date data) and always rely on a thorough analysis of the relational sources. Most methods of this type have three common limitations:

End-user requirements not considered. Although end-user requirements are essential for satisfying the expectations of end-users [36], they are generally not considered in these methods. In most cases a set of design patterns is introduced to identify the MD role that each relational concept may play, and the MD schema is eventually derived from the relational schemas by applying these patterns in a reengineering process. Among the automatable approaches, only [13,24] consider requirements, but the demand-driven stages must be performed manually (as in any other demand-driven approach) and only the supply-driven stages are automated.

Design patterns to identify facts. Identifying facts automatically is a complex task [28], and some of these methods rely on heuristics such that may identify false facts or overlook real ones [11,20]. Specifically, [28] identifies facts from tables with numerical fields whereas [17] identifies facts semi-automatically (i.e., involving the user) by means of table cardinalities and the presence of measures (which are identified by a bayesian network). The other approaches require facts to be identified manually [13,14,24].

Dependency on normalization. Design patterns used to identify dimensional data are mainly based on "foreign" (FK) and "candidate key" (CK) constraints. In MD design, it is well-known that facts and dimensions must be related by many-to-one relationships (i.e., one fact instance is related to just one instance of each dimension; see Section 3.1 for further information). Therefore, the accuracy of results depends on the degree of normalization of the logical schema, since some FKs and CKs are lost if we do not use a schema up to third normal form.

In addition, supply-driven approaches generally risk wasting resources by handling much unnecessary information [36]; since they overlook the MD requirements, they must apply their patterns across all data sources. Each automatable method in the literature is described in detail below:

- Golfarelli et al. [14] introduced the first semi-automatable method for deriving the MD conceptual schema. However, facts have to identified manually. Once this step has been completed, a conceptual schema is derived following the many-to-one relationships for each fact proposed, i.e., dimensions are identified following FK constraints. As such, this method cannot handle denormalized schemas or requirements.
- Phipps and Davis [28] proposed a supply-driven method that is validated a posteriori by a demand-driven stage. They propose potential MD schemas that are then validated by end-user requirements expressed as MDX queries [25]. This approach fully automates the supply-driven stage but the demand-driven stage must be performed manually. Moreover, requirements are not used to guide the process but only to filter results, so new knowledge cannot be derived from requirements and the results are simply pruned in the demand-driven stage. Furthermore, this approach relies on a weak heuristic to identify facts: any relational table containing numerical fields is identified as a potential fact, and dimensional data are identified following FK chains from those tables identified as facts. Although this method relies mainly on FK constraints to identify dimensional data, to our knowledge this is the only method that partially supports denormalized input relational schemas, as each remaining attribute in a table identified as a non-numerical and non-key fact is considered to be an interesting analysis dimension for that fact. However, this approach generates too many results, and filtering them through a manual demand-driven stage would be a difficult and time-consuming task.
- Jensen et al. [17] presented a method in which data mining techniques are used to analyze the data sources. Assuming that the database does not contain composite keys, this method derives valuable metadata such as functional and inclusion dependencies and key or cardinality information, which identifies potential *snowflake schemas* [18]. To infer the metadata, the authors access the instances and apply data mining techniques, which could be unsuitable for large data sources. Moreover, since they are looking for snowflake schemas, they rely on "foreign key"–"candidate key" relationships to identify functional and inclusion dependencies, as in the methods discussed previously. Finally, the complexity of this approach may become problematic due to the high number of permutations computed when searching for inclusion dependencies (as all permutations of potential candidate keys and foreign keys are constructed with the consequent computational cost), since requirements are overlooked and patterns introduced must be computed for all instances.
- Giorgini et al. [13] presented a demand-driven approach for deriving the conceptual MD schema, although they claim that it can also be used as a hybrid approach. They propose to gather MD requirements as described in the demand-driven stage and then map them on to the data sources in a conciliation process, but the degree of automation achieved is relatively low. Facts, dimensions and measures identified during the requirements analysis must be mapped manually over the data sources. Once this has been done, the aggregation hierarchies are completed using an automatic algorithm similar to the one presented in [14]. Finally, the authors propose a refinement step in which the MD schema is rearranged to better suit the user's needs; this information can then be used to reorder dimensions or try to find new directions of analysis, although this process must be performed manually.
- Mazón et al. [23,24] presented a method in which requirements and data sources are conciliated. To our knowledge, this is the first method to introduce a balanced hybrid approach, which makes it the closest to our own proposal. An MD conceptual schema is derived from the end-user requirements, and a set of Query/View/Transformation (QVT) relations is then applied to guarantee that the conceptual schema obtained is consistent with the data sources. However, the conceptual schema derived from requirements

(i.e., the demand-driven stage) is obtained manually. In a second step, a second schema that is consistent with the data sources is derived automatically from the initial schema taken from the end-user requirements. In our case, both steps are carried out automatically and simultaneously, which improves the final result, as discussed in the following subsection. Finally, in contrast to our approach, Mazón et al. assume that the data sources are normalized up to third normal form.

Finally, [5] and [34] present two largely automatable approaches working at the conceptual level. Both approaches overlook requirements (i.e., follow a supply-driven paradigm) and present a similar approach to the one presented in [28]. These works automate the analysis of the sources ([5] from domain ontologies and [34] from ER schemas) in order to extract as much multidimensional knowledge as possible. They later use this knowledge to simplify and guide the compulsory demand-driven stage that we will need to perform.

These approaches are complementary to MDBE, as they can be used in other scenarios where our approach does not apply. Thus, they cannot be smoothly compared to our method. Nevertheless, the contribution of MDBE regarding them is clear: it considers requirements as first-class citizens while they do not. Consequently, i) our output schema fits better to the end-user necessities and ii) similar to [28], their demand-driven stage can be even unfeasible for large ontologies or ER schemas.

2.1. Contributions of the MDBE method

For the sake of clarity we divide this section into two subparts: one discussing our contributions regarding demand-driven approaches (or demand-driven stages within hybrid approaches) and another focusing on automatable approaches working from relational sources.

2.1.1. Demand-driven approaches

In general, matching requirements over the data sources demands a good knowledge of the data sources. As discussed, this is the major drawback of current demand-driven approaches, and is compounded by the time required to analyze the data sources. In our approach, although we also need a reasonable knowledge of the sources to create the SQL queries (i.e., formalize the requirements), the required effort is considerably lower. We use a well-established language such as SQL to facilitate the mapping of the requirements over the data sources. Consequently, we only need an expert to create these queries, and many real organizations will have access to someone with these skills. Our framework has two main advantages over current demand-driven approaches:

- MDBE, like current automatable methods, works exclusively with relational data sources. This means that the organization has a relational transactional system, which makes it very likely that a number of employees will possess the skills required to create the queries; the database administrator, for example, would have a perfectly suited profile. On this point, we make the same assumptions as used in current automatable approaches.
- Although our approach follows a hybrid paradigm, most of the tasks are automated. Briefly, this means that the user is not required to *deeply analyze* the sources. This contrasts with current demand-driven approaches, which require detailed and exhaustive analysis of the data sources; hints and tips are given (in case of the most formal methods, MD patterns) that the data warehouse expert must then search for manually across all data sources (logical or conceptual schemas, depending on the approach chosen). The time-consuming nature of this task can render it unfeasible when large databases are used.
  For example, [26] introduces a commonly cited method for deriving the MD schema from an ER schema. It requires each ER entity to be classified as a *transactional* (the basis for fact tables), *component* (details or components of business events that will produce dimensions) or *classification* (that will be used to shape the dimension hierarchies) entity. The authors give advice on how these entities can be identified. Thus, "transactional entities must describe events that happens at a point in time and contain measures or quantities that can be summarized". Formal rules are given for each type of entity to give shape to the MD schema. For example, a typical rule is discovering *functional dependencies* (FDs) to identify dimensional data. Manual discovery of FDs is an unfeasible task for most systems [7,11,35], and automatic methods for identifying FDs need to address this task at the physical level (i.e., using the instance semantics). These methods have various drawbacks, propose solutions that are computationally expensive, and register drops in performance when a large number of attributes or instances are processed [15,22,33,35].
  Other important demand-driven approaches, such as [16], use a formal framework to derive the MD schema. However, little information is given about how to identify the MD concepts over the data sources. This scenario is repeated in the most recent demand-driven approaches; for example, [30] derives the MD conceptual, logical and physical (using the Oracle MOLAP Tool) with a UML-based method that introduces a metamodel and a set of transformations to perform the mapping between each metamodel. However, this approach suffers from the same drawback as previous approaches, and analysis of the sources may still be unfeasible if it has to be performed manually. More comprehensive descriptions of these methods and how they work can be found in [8] and [12].
  In contrast, our approach introduces a manual formalization step but the user does not need to analyze the data sources or perform the complete mapping of the requirements over the data sources. Importantly, the exhaustive analysis and iterative application of the MD patterns are delegated to the MDBE method. In other words, the counterpart to our method would require the entire process described in Section 4 to be carried out manually (by properly applying the MD patterns introduced in Sections 3.1 and 3.2), which is the case of current demand-driven methods. Therefore, the manual workload in our proposal is considerably lighter than in previous approaches.

2.1.2. Automatable approaches

The MDBE method was designed to overcome the limitations shared by current automatable methods. To our knowledge, (i) MDBE is the first method with an automated demand-driven stage. Our approach requires end-user requirements to be formalized as SQL queries, after which MDBE validates each SQL query to determine whether it makes MD sense (see Section 3.1 for further information). The main contribution in this area is that (ii) MDBE validates the explicit and implicit MD knowledge in the query. For example, relationships between concepts depict the potential MD role that each concept could play, and *joins* stated in the WHERE clause identify relationships (i.e., *concept associations*) explicitly stated by the user that, in some cases, may not be in the logical schema of the data sources. For example, consider a database overlooking foreign keys. In these cases, previous approaches that rely on primary key–foreign key relationships would overlook this information. In contrast, in our approach, these missing relationships will be stated (if they are of relevance for the user) by means of concept associations (i.e., joins) in the SQL query. Thus, if a join attribute is identified as dimensional data, this MD role is propagated to its join counterpart (like a supply-driven approach would do if the proper primary key–foreign key relationship were defined). Another example would be a denormalized database. In this case, if the query performs data grouping (i.e., it contains a GROUP BY clause) or contains comparison clauses in the WHERE clause, the attributes involved in these clauses are identified as dimensional data. In other words, the SQL queries may provide additional relevant knowledge to that captured in the sources. Nevertheless, we also harness the knowledge contained in the data sources (as in supply-driven approaches), such as foreign key and candidate key constraints, if present. In addition, (iii) MDBE works at the *attribute level* (SQL queries handle attributes), whereas other automatable methods work at the table level. Consequently, relational attributes can be labeled as dimensional or factual data and, in turn, relational tables are identified as dimensional data, factual data or tables containing factual data and dimensional data [18]. We can therefore identify the role played by each attribute in each relation and split it into different concepts in the resulting MD schema. Thanks to these contributions, (iv) MDBE is able to handle denormalized relational schemas to some extent. The analysis of requirements at the attribute level allows MDBE to identify dimensional or factual attributes that previous approaches would overlook. However, regarding dimensional data identified from denormalized relations, MDBE cannot automatically generate the dimension hierarchies as the domain FDs needed to shape hierarchies are missing in the source schema. In other words, each requirement (i.e., SQL query) will identify attributes representing interesting analysis perspectives, but the relationships between these attributes (i.e., the dimension hierarchies) cannot be extracted from denormalized data sources. In these cases, the designer will be responsible for restructuring this kind of dimensional data. However, note that the automatic methods for identifying FDs can help in this task, since the set of attributes to work with is considerably smaller than in the general case.

MDBE also provides the advantage of carrying out the demand-driven and supply-driven stages simultaneously in many aspects. This means that we are able to produce more and better-quality outputs than methods in which the two stages are performed sequentially. For example, (v) MDBE can derive implicit knowledge according to the input query and the data sources. Some attributes in the query may not play a relevant role in the output produced, in which case they could be overlooked. However, we analyze all of the potential alternatives, as well as metadata in the logical schema, and consider how these alternatives would affect the output schema, in some cases deriving interesting alternatives overlooked by the user. This contribution is important because it is often assumed in data warehouse modeling that the user may not recognize the analytical potential of all the data sources and, therefore, may overlook potentially useful analytical alternatives. However, analyzing all of the data sources can be expensive and produce too much noise in the final result [36]. In this paper we present an intermediate solution, in which concepts are analyzed to determine their analytical potential if they are implicitly related to concepts already stated in the end-user requirements (see step 6 in Section 4.1 for further details).

In addition, (vi) MDBE can derive new concepts that are not stated in the logical schemas. Since we handle requirements automatically, we can analyze them in depth and identify information such as concept specializations or newly derived measures (see Section 3.1.1 for further details). (vii) MDBE also keeps track of relevant metadata extracted from the requirements, which will be relevant in the implementation stage: specifically, interesting data granularity within a fact (see Step 2 in Section 4.1) and data summarizability properties (see Steps 1 and 3 in Section 4.1).

It is important to note that the method presented in this paper is a natural development of the one presented in [4]. We have improved many aspects of our previous work: MDBE can now handle denormalized logical schemas, and we have improved the conciliation between the demand-driven and supply-driven stages (see (vi)). We have also relaxed some of the theoretical patterns introduced in the preliminary version, to cope with practical issues. Finally, we present a detailed case study and relevant statistics about our process obtained using the MDBE tool.

3. Our approach

The main aim of our approach is to support the data warehouse design process. It consists of two steps: requirement formalization and the MDBE method (as shown in Fig. 1). Furthermore, as in any classical design process, a requirement elicitation pre-process is needed. Although this pre-process falls outside the scope of this paper, some relevant features should be noted here. Data warehousing systems differ in various aspects from conventional operational systems (since they are designed to support decision-making) and need specialized requirement elicitation processes [24,36]. However, this issue has been studied in depth, and there are several methods that can be used in preliminary step (for example, [13,24,29,32,36]). Nevertheless, note that we gather *information requirements* in this step. Information requirements [36] are designed to meet end-user information necessities, which is the objective of a data warehouse [24]. Unlike in other systems, end-users can easily determine their information necessities because they consist of data that is required in their decision-making processes. Consequently, information requirements can be stated in the end-users' own

words and closely reflect their reality. For example, "*examine stocks provided by suppliers*" or "*analyze customer purchases with regard to region, product and time*" would be typical information requirements.

The next step in our approach formalizes the requirements gathered. As discussed previously, we aim to automate the manipulation of requirements (i.e., integrate them in a fully-automated method), so they must be translated into a computer understandable language. In our approach, end-user requirements are expressed as SQL queries over the relational data sources (i.e., at the logical level over the data sources). This step must be carried out by a database expert capable of lowering the level of abstraction of the input requirements to the logical level (see Section 2.1.1 for a detailed discussion of the advantages and disadvantages of this step).

As shown in Fig. 1, the next step in our approach is to apply the MDBE method, which has two inputs: the end-user information requirements (expressed as SQL queries) and the logical model of the data sources. As output, MDBE presents an MD schema derived from the data sources, which allows the user to retrieve data demanded in the input requirements. In this step, MDBE determines *whether each input SQL query represents a valid MD query*, i.e., if the query retrieves data that can be analyzed from a MD perspective; this is the case if the input SQL query represents a valid set of multidimensional operators over a MD schema (i.e., if the query represents data retrieved from a MD schema after performing valid data manipulations according to the MD model). For this purpose, we carried out a study to identify which constraints should be guaranteed by a query in order to represent a combination of MD operators (see Section 3.1 for further information). These constraints can be summarized as follows: data retrieved should be (1) free of data summarizability anomalies, and (2) placeable in an MD space. If these constraints are satisfied, we may find a set of MD operators which would retrieve that data from the proposed MD schema. Finally, note that each query (i.e., each MD requirement) produces a potential MD schema. The last step in the MDBE method would allow the user to conciliate those results into a minimal set of conceptual schemas that meet all of the requirements (i.e., a constellation of MD schemas).

3.1. Foundations

As there is not yet a standard MD notation, we now introduce the notation used in this paper.

[Notation] Multidimensional modeling: Multidimensionality is based on the fact/dimension dichotomy. **Dimensional concepts** produce the MD space in which the **fact** is placed. **Dimensional concepts** are those concepts likely to be used as a new analytical perspective, which have traditionally been classified as **dimensions**, **levels** and **descriptors**. Thus, we consider that a **dimension** consists of a hierarchy of **levels** representing different granularities (or levels of detail) for studying data, and a **level** containing **descriptors** (i.e., **level** attributes). In contrast, a **fact** contains **Cells** which, in turn, contain **measures**. As in [26], we consider that a **fact** may contain not just one but several different materialized levels of data granularity. Therefore, one **Cell** represents individual cells of the same granularity that contain data relating to the same **fact** (i.e., a **Cell** is a "Class" and cells are its instances). Specifically, a **Cell** of data is related to one **level** for each of its associated **dimensions** of analysis. For example, consider Fig. 3. The lineitem **fact** contains several **Cells**. We may analyze this **fact** from its finest data granularity (i.e, {lineitem_dim × part × supplier × orders_dim}) or from a coarser data granularity (for example, {lineitem_dim × part × nation × customer}). Each level of data granularity available for a given **fact** gives rise to a **Cell** within that **fact**, and any of the factual instances within the **Cell** is called a cell. For example, every lineitem_dim, part, nation and customer tuple univocally identifies a cell of data (i.e., a l_quantity, l_extendedprice, l_discount, l_tax and a derived_measure). Finally, one **fact** and several **dimensions** for its analysis produce a star-schema.

Next, we present the criteria upon which the study is based, namely those used to validate the input SQL query (i.e., the information requirement) as a suitable MD requirement. A query *makes multidimensional sense* if it retrieves data that can be analyzed from a MD perspective; in other words, if the data retrieved form a *data cube* [18]. We carried out a study [6] to identify the constraints that an SQL query must satisfy in order to make MD sense.

Data manipulation in the MD model should be restricted to the *multidimensional operators*. Unfortunately, we do not yet have a standard MD algebra, and several MD operators have been suggested in the literature. To overcome this problem, we surveyed all of these MD operators and analyzed how they should be translated into SQL queries in a relational implementation of the data warehouse. We found that MD data manipulation (i.e., multidimensionality) focuses on two aspects: (i) the placement of data in an MD space; and (ii) the correct summarizability of the data. If the data retrieved satisfy both constraints they can be depicted as a data cube (i.e., orthogonal dimensions fully functionally determining the fact) free of summarizability problems [6]. In other words, this query would represent the translation of a set of MD operators into SQL. Below, we provide definitions of the basic axioms identified in our study.

Definition 1. The cube-query template

The standard SQL'92 query template to retrieve a Cell of data from the RDBMS was first presented in [18]:

```
SELECT l1.ID, …, ln.ID, [ F( )c.Measure1[ ) ], …
FROM Cell c, Level1 l1, …, Leveln ln
WHERE c.key1=l1·ID AND … AND c·keyn=ln·ID [AND li·attr Op. 𝒦]
[GROUP BY l1.ID, …, ln·ID ]
[ORDER BY l1.ID, …, ln·ID ]
```

The FROM clause contains the "Cell table" and the "Level tables". These tables are properly linked in the WHERE clause by "joins" that represent *concept associations*. The WHERE clause also contains logical clauses restricting a specific **level** attribute (i.e., a **descriptor**) to a

constant using a comparison operator. The GROUP BY clause shows the identifiers of the **levels** at which we want to aggregate data. Those columns in the grouping must also be in the SELECT clause to identify the values in the result. Finally, the ORDER BY clause is designed to sort the output of the query.

Note that we talk about **Cells** instead of **facts**. The reason is that every SQL query will produce a single data cube (i.e., a specific level of data granularity) and in our method, we will first identify **Cells** of interest and later, **facts**.

Definition 2. The multidimensional space arrangement

Dimensions arrange the MD space (i.e., the data cube) in which the **Cell** under study is depicted. Each instance of factual data is identified (i.e., placed in the MD space) by a point in each of its analysis **dimensions**. Conceptually, this means that a **Cell** must be related to each analysis **level** by a to-one relationship; that is, every instance of the **Cell** (i.e., every cell) is related to exactly one instance of an analysis **dimension**, and every **dimension** instance may be related to many instances of the **Cell**.

Definition 3. The base concept

We use **base** to denote a *minimal* set of **levels** functionally determining a **Cell**. This guarantees that two different instances of data cannot be placed in the same point of the MD space; in other words, given a point in each of these **dimensions**, they determine only one instance of data. In addition, **dimensions** (and thus, **levels**) that produce a **base** must be *orthogonal* (i.e., functionally independent) [2]. Otherwise, we would use more **dimensions** than strictly needed to represent data, which would generate empty meaningless zones in the space. In a relational implementation of the data warehouse, the **base** concept would be implemented as the **Cell** primary key.

Definition 4. The necessary conditions for correct summarizability

Data summarization must be correct, which is ensured by applying three necessary conditions (which, intuitively, are also sufficient) [21]: (1) **Disjointness** (the sets of objects to be aggregated must be disjoint); (2) **Completeness** (the union of subsets must constitute the entire set); and (3) **Compatibility** of the **dimension**, the type of measure being aggregated and the aggregation function. Compatibility must be satisfied, since certain functions are incompatible with some **dimensions** and types of measures. For example, we cannot aggregate Stock over Time **dimension** by means of sum, as some repeated instances would be counted. Unfortunately, compatibility cannot be automatically verified. Nevertheless, our method keeps track of the compatibility information extracted from the requirements (see Steps 1 and 3 of Section 4.1; there, data summarizability properties identified from the query are considered and properly stored).

The MDBE validation process checks if each SQL query can produce a meaningful MD cube. For our purpose, *an SQL query is considered to make MD sense if it satisfies the following semantic constraints* (which can be easily derived from the definitions presented above):

- [C1] *Multidimensional compliance*: The SQL query must follow the cube-query template. Thus, any attribute involved in the query must play either a dimensional or a factual role. In other words, every attribute must be labeled as one of the MD concepts described in our notation (see Definition 1).
- [C2] *Star-schema*: **Cells** are related to **levels** by to-one relationships (see Definition 2).
- [C3] *Uniqueness*: Every two different data instances retrieved by the query must be placed in different points of the MD space (see Definitions 1 and 3).
- [C4] *Orthogonality*: The set of concepts that produce the MD space must be orthogonal (see Definition 3).
- [C5] *Completeness*: Data summarization performed in the query must be complete. Thus, the conceptual relationships involved in the query should not allow NULL values when relating factual to dimensional data (see Definitions 2 and 4).
- [C6] *Disjointness*: Data summarization performed in the query must be disjoint. Thus, the conceptual relationships involved in the query should avoid double-counting instances; for example, Cartesian products (see Definitions 2 and 4).
- [C7] *Restricted selection*: Joins performed in the query cannot be used to select data. In the MD model, selections must be performed through comparisons in the WHERE clause (see Definition 1).

These constraints are applied to identify the MD role played by each relational concept and to guarantee that the schemas proposed by our method will be able to retrieve (by using MD operators) data demanded in the requirements. Specifically, [C2], [C3] and [C4] guarantee that the SQL query produces a valid MD space; [C5] and [C6] preserve data summarizability; and [C7] guarantees that data manipulation is restricted to the MD operators.

3.1.1. Additional considerations

As stated above, the constraints are used to validate the final output. If they are not satisfied in a given query, we can end the process and inform the user that the current requirement does not make MD sense. Otherwise, the final result forms a data cube and we can say that the input query is a valid MD requirement. However, even if [C5], [C6] and [C7] are not satisfied, a valid data cube of interest to the end-user may still be retrieved.

3.1.1.1. Relaxing completeness and disjointness. Our method can identify when disjointness and completeness are not preserved in the logical schema of the data sources. However, end-user requirements may include new concepts that have not been captured in the data sources but which may still be derived from them. Specifically, (i) in the SQL query, the user may require a concept specialization not preserving completeness in the logical schema or ii) a derived measure not preserving disjointness. Consider

Fig. 2. The first case would apply if, for example, a lineitem does not require a suppkey when inserting data (i.e., if suppkey allows NULL values). Note that it would make sense, as we may assign the supplier later, by means of another task that, for example, minimizes expenses. In this case, if we want to use supplier as an analytical perspective of lineitem it would not preserve completeness regarding the data sources. According to the constraints discussed in this section (see [C5]), NULL values should not be allowed when relating factual data to dimensional data. However, this relationship (and thus, this query) would make sense if we are interested in analyzing only those lineitem with supplier (i.e., a specific specialization of lineitem not depicted in the sources). Thus, we may relax [C5] and produce this result if the user is interested in this specialization.[1] Nevertheless, it is important to note that [C5] is relaxed regarding the data sources, but it is guaranteed by considering the specialization.

In the second case, two values not satisfying the disjointness constraint may produce a meaningful derived measure; for example, if the measure is properly weighted in the query. TPC-H Q9 is an example of this case. This query is expressed in SQL as:

```
SELECT nation, o_year, sum(amount) as sum_profit
FROM (SELECT n_nameasnation, extract(year from o_order date) as o_year,
l_extended price*(1-l_discount)-ps_supply cost*l_quantity as amount
FROM part, supplier, lineitem, part supp, orders, nation
WHERE s_suppkey=l_suppkey AND ps_suppkey=l_suppkey
AND ps_partkey=l_partkey AND o_orderkey=l_orderkey
AND s_nationkey=n_nationkey AND p_name like '%[COLOR]%) as profit
GROUP BY nation, o_year
ORDER BY nation, o_year desc;
```

See the graph in the right side of Fig. 8. This query selects factual data from two different nodes (lineitem and partsupp). However, these nodes are related by means of a many-to-one relationship that, in principle, will produce double-counting (factual data from partsupp may be considered several times when joining it to factual data in lineitem). According to our criteria (see [C6]), double-counting must be forbidden to preserve disjointness. However, double-counting may happen and yet make multidimensional sense. Indeed, this is the case of Q9. This query, weights the l_quantity attribute of lineitem with the ps_supplycost of partsupp (one ps_supplycost may be related to many l_quantity values) and this value is taken away from the income obtained from the sale. In other words, it is calculating the profit obtained on a given line of parts. Clearly, despite not preserving disjointness, the semantics of Q9 does make MD sense. In general, our method produces results which satisfy [C5] and [C6]. However, in the MDBE method we apply the following rule:

R1: If we cannot produce an output by satisfying [C5] *and* [C6]*, our method tries to identify relevant derived* **measures** *or concept specializations (not captured in the data sources) by relaxing these constraints.*

In this paper we will clearly note those steps in which this assumption stands. Steps affected by this assumption will try to guarantee both constraints, but if no result is produced the steps have to be relaunched and [C5] and [C6] relaxed. In other words, if [C5] and [C6] are not preserved regarding the data sources, we propose alternative solutions preserving them (i.e., considering specializations or derived measures not captured in the sources). In such cases, MDBE may produce schemas that do not preserve the completeness or disjointness of the data sources, and we inform the user that results are only correct if either a concept specialization or a new derived measure is considered.

3.1.1.2. Allowing selections by means of joins. In some cases, the input SQL query may use joins to select data that, according to [C7], would not make MD sense. However, this could occur if the person creating the queries is not sufficiently skilled for the task. For example, by means of alternative *join paths* between two concepts.

Consider Fig. 2 and a query containing two different join paths between lineitem and nation in the WHERE clause. For example, one path following the lineitem–orders–customer–nation foreign key–primary key relationships, and another following the lineitem–partsupp–supplier–nation foreign key–primary key relationships. Clearly, these joins are used to select lineitems having as customer and supplier people living in the same nation (i.e., we select factual data having the same value for both paths). But this query would not make multidimensional sense: there is no multidimensional operator performing such operation and thus, there is no OLAP tool that could produce this query. Consequently, MDBE should disregard this query. However, it may happen that both paths are equivalent (i.e., both values of nation always coincide for each and every lineitem). For example, it would be the case if the following integrity constraint holds within the organization: *every customer will be provided with items supplied by a supplier of his/her own country*. Indeed, such constraint would make sense in many organizations.

For this reason, if the query contains alternative join paths between two concepts we may distinguish two cases: whether the user guarantees that both paths are equivalent (i.e., instances selected through each path are exactly the same) then, we can rewrite it in such a way it preserves [C7]. Otherwise, the query should be disregarded. In the first case, the query must be rewritten by defining two different *alias* for nation (for example, nation *n*1 and nation *n*2) and use each in one of the paths. The resulting query would be semantically equivalent to the previous one and it will make MD sense (i.e., this query could be generated by OLAP tools).

In the implementation of our method, the user is responsible for relaxing this constraint (the MDBE tool has a check-box for activating or deactivating the constraint).

[1] Note that this type of context-sensitive summarizability is also considered in the multidimensional normal forms introduced in [19].

3.2. Internals

In this section we show how *multidimensional concepts* are identified from the *relational concepts* involved in each SQL query. MDBE aims to analyze the knowledge available from the SQL query and the relational schema to infer the MD role played by each relational concept. For this purpose, it uses a graph to store information elicited from the overall process, which we will refer to as the *multidimensional graph*.

Briefly, *the MD graph represents the relational schema fragment captured in the SQL query (and not the whole relational sources)*. It is composed of *nodes*, which represent relational tables involved in the query, and *edges*, which relate nodes joined in the query (i.e., keep track of *concept associations*). Furthermore, each node contains information about the relational *attributes* involved in the query. Importantly, note that a node does not fully represent a relational table. *Nodes capture those table attributes of relevance for the query analyzed* (i.e., the table fragment of interest). Similarly, edges represent the relationships between graph nodes (and not between relational tables).

In our approach, MDBE aims to validate the MD graph (i.e., the input query) as a suitable MD requirement. It *labels* each graph node and its attributes as MD concepts, in such a way that the whole labeling satisfies the MD constraints described in Section 3.1. Thus, MDBE tries to find a MD meaning for the relational schema fragment captured in the graph (roughly speaking, we may say that it looks for a *MD interpretation* of the graph).

This section discusses how each relational concept may be labeled. Here, we present the criteria used to identify the MD role of attributes, nodes and edges according to our foundations (see Section 3.1). Later, Section 4.1 introduces an algorithm that applies these criteria. The algorithm analyzes the SQL query and the relational schema to look for the labeling criteria introduced in this section and accordingly, labels the graph.

3.2.1. Attribute labeling

A given relational attribute of MD interest may play a dimensional or a factual role. If it has a useful analytical value it will be labeled as a **measure** (i.e., factual data) and if it represents an interesting analytical perspective for the MD data it will be labeled as a **dimensional concept** (i.e., dimensional data). When an attribute is labeled as a **dimensional concept**, depending on its semantics, it may be identified as a **level** or a **descriptor**.

We use the 7 constraints introduced in Section 3.1 to identify the MD role played by an attribute in the SQL query. Note that a given attribute may be labeled as both a **dimensional concept** and a **measure**. In their MD model, Agrawal et al. [3] proposed handling **measures** and **dimensions** *uniformly* (they presented two MD operators to transform **measures** into **dimensions** and vice versa). This idea was also incorporated into later MD models (see [6]). MDBE allows multiple labeling of a relational attribute (which we will refer to as *dual attributes*), so the final MD schema will contain a **measure** and a **dimensional concept** derived from the same attribute. It will happen if a given attribute is labeled as factual data according to any of our criterion, and as dimensional data by another.

3.2.2. Node labeling

At this point, it is important to remark the subtle difference between relational tables and graph nodes. Nodes do not represent the whole relational table but the set of table attributes involved in the query and, according to the kind of attributes they contain, can be labeled as either dimensional data or factual data (or even both, as discussed later in this subsection):

- *Dimensional data* (*L*): If the node contains attributes representing a useful analytical perspective for the MD data, it will be labeled as a **level** (i.e., as *L*).
- *Factual data* (*CM* or *C*): If the node contains factual data, we label it as a **Cell**. However, we distinguish between two different types of **Cells**:
  - *Cell With Measures* (*CM*): These nodes represent **Cells** that contain **measures**. According to [C3], these nodes will also contain **dimensional concepts** that determine the MD space in which to place the data. There are three possible cases: the **Cell** directly contains (i) the MD **base**, (ii) a *candidate* **base** (i.e., a set of attributes preserving a one-to-one relationship with the MD **base**) or (iii) a set of attributes fully determining the MD **base**.
    To preserve [C3], in the (i) and (ii) cases, it means that either the MD **base** (candidate **base**) corresponds to a table CK (also represented in the node) or, if performing data aggregation in the query, the GROUP BY clause contains attributes from the node. In the latter case (iii), consider the TPC-H business query #5 introduced in Section 1, and the TPC-H relational schema shown in Fig. 2. In this query, the name attribute (from the nation node) forms the MD **base** and lineitem plays a **Cell** role. To

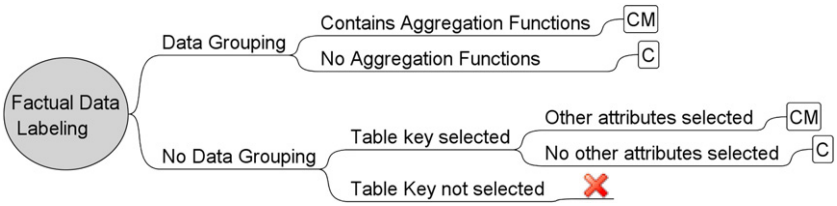

Fig. 4. Decision diagram for labeling nodes representing factual data.

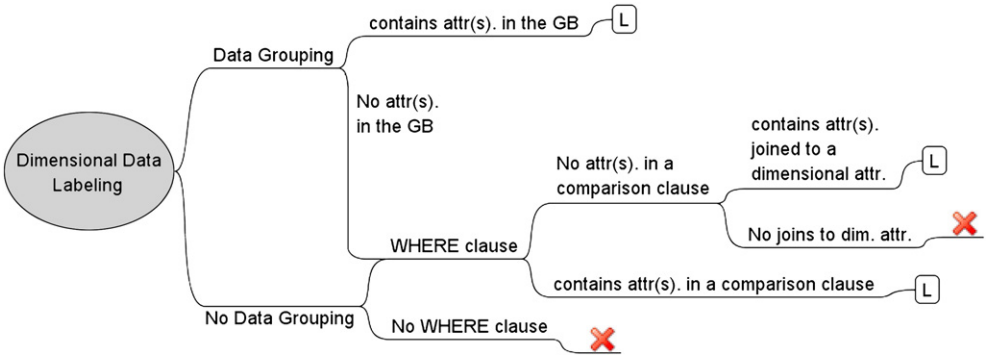

Fig. 5. Decision diagram for labeling nodes representing dimensional data.

preserve [C3], lineitem is properly *linked* to name in such a way that every instance of factual data is related to just one nation name value. In other words, lineitem functionally determines the nation of the supplier (indeed, this dependency is properly captured in the relational schema by means of FKs). In this paper, we use *link attributes* to denote the **dimensional concepts** contained in a **Cell** placing factual data in the MD space (i.e., the (i), (ii) or (iii) cases discussed).

– *Cell* (*C*): These nodes represent "factless facts" [18]. This definition is equivalent to the previous one, but this type of **Cell** does not contain **measures**. These facts are very useful for describing events and coverage and can be used to formulate many interesting questions [18].

To determine the factual label of a node, we follow the decision diagram shown in Fig. 4, which generates questions about the query and the table metadata. These questions derive directly from constraints introduced in Section 3.1, and we distinguish between two possible scenarios: one in which the current input query performs data grouping (i.e., it contains a GROUP BY clause) and another in which it does not. In the first case, and according to [C1], if the SELECT clause contains an *aggregated attribute* (i.e., summarized by an aggregation function), that attribute will play a **measure** role. Consequently, the node is labeled as *CM*. Otherwise, if no aggregated attribute is selected, it is labeled as a factless fact *C* (i.e., the **Cell** does not contain **measures**).

Similarly, if no data grouping is performed in the query but we are able to produce an MD space (i.e., a table CK is selected), the node will be labeled as a **Cell**: *CM* if attributes other than the key are selected (i.e., if it contains **measures**); otherwise, *C*. According to [C6], any other alternative would not make multidimensional sense as a **Cell** (depicted in the figure by the *X* mark).

When checking if any **measure** other than a table key is selected, we do not only consider numerical attributes. Traditionally, numerical attributes produce **measures** because they are perfectly additive but, as discussed in [18], semi-additive or non-additive values could be of interest to the end-user. Moreover, there are some areas in which non-numerical values are additive. For example, the *spatial databases* area contains algorithms for the aggregation of text values representing geographical information (see [10]).

Note the MD semantics involving each alternative in the decision diagram discussed above. **Cells** identified without grouping will represent "atomic factual data" [2] (i.e., the finest granularity of data in the data warehouse), whereas those **Cells** identified by data aggregation will represent "aggregated factual data" (i.e., coarser data granularities of interest).

Similarly, to determine the dimensional role of a node we follow the decision diagram shown in Fig. 5. Again, it generates questions about the query and the table metadata. First, we check if the input query performs data grouping, and according to [C3], attributes in the GROUP BY (i.e., contains attributes being part of the MD **base**) will play a **level** role. Consequently, the nodes containing these attributes are labeled as *L*. If no data grouping is performed or none of the node attributes are used to group data, we check the WHERE clause. We distinguish between two possible scenarios: if any of the node attributes are involved in a *comparison clause* or in a *join*. In the first case, according to [C7], selections are performed over dimensional data and thus, that attribute will be identified as dimensional data. In the second case, according to [C1], joins in the WHERE clause represent *conceptual associations*. Thus, if a node contains an attribute joined to a dimensional attribute (i.e., an attribute already identified as dimensional data) then, both attributes represent dimensional data. Any other scenario would not make sense as dimensional data and the node is not labeled (see the *X* mark in the figure).

3.2.2.1. Supporting denormalization. As discussed in Section 2, our method can handle denormalized input schemas, which implies that a given node may play a factual and dimensional role simultaneously. This scenario occurs when a graph node is identified as factual data by the decision diagram shown in Fig. 4, and as dimensional data by the decision diagram shown in Fig. 5.

In this case, we introduce two new labels to identify *hybrid nodes* containing factual *and* dimensional data. Note, however, that a factual node (i.e., those labeled as *CM* or *C*) always contains dimensional data forming the MD space (i.e., the *link attributes*). However, hybrid nodes contain *additional* dimensional data: either attributes playing a degenerated dimension [18] role and/or attributes playing the role of denormalized dimensional data (i.e., partial or whole denormalized dimension hierarchies):

- *Cell With Measures and Additional Dimensional Data* (*CDM*): This label is equivalent to the *CM* label (this node therefore contains the *link attributes* as well as **measures**) with additional dimensional data. The additional dimensional data represent other analytical **levels** and **descriptors** that form other analytical perspectives.

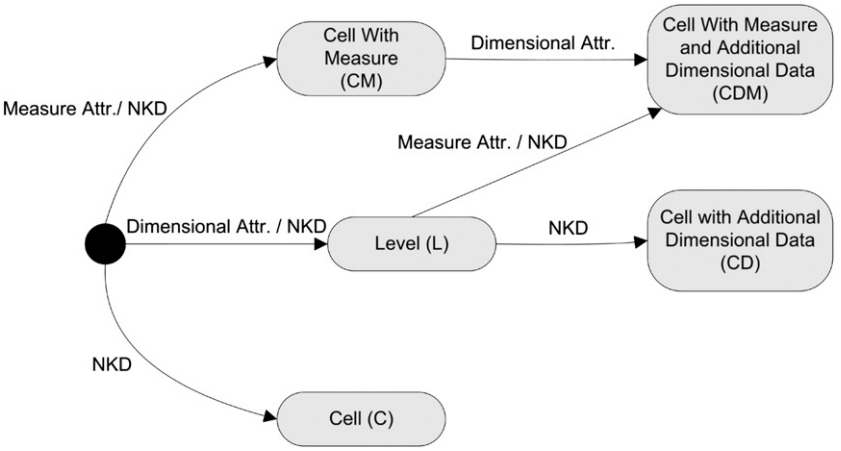

Fig. 6. State diagram showing the transition between node labels.

- *Cell with Additional Dimensional Data* (*CD*): Similarly, nodes representing factless facts with additional dimensional data are labeled as *CD*.

For example, consider the TPC-H business query Q5 introduced in Section 1. If this query contained an additional comparison clause such as l_shipdate = '12-02-2009' in the WHERE clause, MDBE would identify lineitem as a hybrid node: according to the decision diagram shown in Fig. 4, lineitem is labeled as *CM* (because the query contains grouping and lineitem contains two aggregated attributes — i.e., l_extendedprice and l_discount — in the SELECT clause) and, according to the decision diagram shown in Fig. 5, it will also identify lineitem as dimensional data (because lineitem does not contain any attribute in the GROUP BY clause, but it contains l_shipdate, which is involved in a comparison clause of the WHERE). As result, MDBE labels this node as a hybrid node (*CDM*) since it contains **measures**, the *link attributes* and also an additional dimensional attribute.

Once we know how to label attributes (by means of the 7 constraints introduced in Section 3.1) and nodes (by means of the decision diagrams previously introduced in this section), the node labeling state diagram can be produced, as shown in Fig. 6. The transitions between possible labels are shown. Every node is unlabeled in its initial state (i.e., at the beginning of the labeling process) and the label is then updated according to the explicit knowledge extracted from the query. For example, from the initial state, we can label each node as either *CM* (if one of its attributes is identified as a **measure**) or *L* (if one of its attributes is identified as a **dimensional concept**). From the *CM* state, we can keep the same label if any other **measure** is identified or update it to *CDM* if an attribute playing a dimensional role and not part of the *link attributes* is identified (i.e., if this node contains factual and dimensional data).

Some transitions shown in the state diagram are labeled with the *NKD* (*New Knowledge Discovery*) tag. In MDBE, a state transition can take place due to either the explicit knowledge extracted from the query or the implicit knowledge derived both from the input query and the data source metadata. The latter case represents a scenario in which either the query does not *explicitly* establish a node role (thus, the node is not yet labeled) or the implicit knowledge available suggests an *alternative* labeling. In these cases we analyze every labeling alternative for the node in question. As discussed in Section 2.1 and presented in detail in Section 4.1 (see Step 6), this process is used to derive new MD knowledge that is not stated in the requirements.

3.2.3. Edge labeling

Edges relate nodes and keep track of joins in the WHERE clause of the query. They provide information about how relational concepts are related in the relational schema fragment captured in the query. For our purpose, a given edge is labeled according to the MD conceptual relationship it may represent (i.e., the MD interpretation we may infer). We consider four potential labels: *Cell–Cell*, *Cell–Level*, *Level–Cell* and *Level–Level*. For example, a *Cell–Level* edge label would mean that the relationship could relate factual data (i.e., a node playing a **Cell** role) to dimensional data (i.e., a **level**). Note that edge labels only depict the conceptual role that each node may play relative to a given edge. Therefore, these labels show how factual and dimensional data may be related but, as previously discussed, MDBE has different labels to identify factual and dimensional nodes. Specifically, a node playing a factual role may be labeled as *CM*, *C*, *CDM* or *CD* whereas a node playing a dimensional role can only be labeled as *L*. In other words, regarding edges, hybrid nodes can only play a **Cell** role, as justified later in this section.

Next, we introduce the edge labeling process:

(i) For each join between tables in the WHERE clause, we first infer the relationship multiplicity with regard to the schema constraints of the join attributes (i.e., FKs, CKs or Not Null values). In the relational model, the multiplicity of a relationship depends on how attributes involved are defined in the schema: Whether they (as a whole, since we consider multi-attribute joins) play the role of a relation CK and/or if they are defined as a FK to the other attribute(s) and/or if they allow null values. Joining to a CK guarantees to match at most one instance of the relation.[2] Otherwise it may match many of them.

[2] We assume, as all systems do, that a FK can only point to a CK set of attributes.

Table 1
Summary of rules used to infer relationship multiplicities.

CK_{n1}	CK_{n2}	FK_{n1}	FK_{n2}	NN_{n1}	NN_{n2}	Relationship	Multiplicity
×	×	×	×	?	?	*Attr.* → *Attr.*	*N* − *M*
✓	×	×	✓	✓	✓	*CK* → *FK* + *NN*	1 − o *N*
✓	×	×	?	✓	?	*CK* → *Attr.*	1 o − o *N*
×	✓	✓	×	✓	✓	*FK* + *NN* → *CK*	*N* o − 1
×	✓	?	×	?	✓	*Attr.* → *CK*	*N* o − o 1
✓	✓	✓	✓	✓	✓	*CK* + *FK* → *FK* + *CK*	1 − 1
✓	✓	✓	×	✓	✓	*CK* + *FK* → *CK*	1 o − 1
✓	✓	×	✓	✓	✓	*CK* → *CK* + *FK*	1 − o 1
✓	✓	×	×	✓	✓	*CK* → *CK*	1 o − o 1

Similarly, an attribute not allowing null values and being defined as FK will surely match one and just one instance. Otherwise, it may introduce zeros. Table 1 summarizes all those relationship multiplicities that we may find in the relational model with regard to the attributes metadata. There, each row represents a specific relationship between nodes (i.e., a kind of join). Notation used is the following: the first six columns represent all possible combinations with regard to the constraints of join attributes (the subscripts *n*1 and *n*2 refer to each one of the attribute sets joined): As CK, as a FK pointing to the other attribute(s) or as NN (*not null*) attribute(s). If an specific cell is ticked (i.e., ✓), it means that that attribute is constrained according to that column. Otherwise, it is marked with a × mark. Notice that not all the combinations are allowed and some columns determine the following ones. For instance, CK attribute(s) cannot accept null values. Moreover, a cell is marked with a ? mark if previous columns already determine a certain multiplicity, meaning that this constraint does not affect the obtained multiplicity. Finally, the last two columns inform about the specific join depicted as well as the multiplicity inferred. There, an Attr. represents unconstrained attribute(s); that is, not defined either CK nor FK and allowing null values.

(ii) Next, according to the semantics of the multiplicity inferred, we label each edge with those MD relationships it could represent (i.e., the MD concepts it could relate). Potential edge labels are shown in Table 2, and those combinations making MD sense (according to [C2], [C5] and [C6]) are marked with a ✓. For example, a many-to-one relationship, depending on zeros, could represent a *Cell–Level*, *Cell–Cell* or a *Level–Level* relationship but not a *Level–Cell* relationship, since it would not satisfy [C2]. However, completeness could eventually be relaxed to identify concept specializations, as explained in Section 3.1.1. These cases, in which completeness would be relaxed a posteriori, are shown in Table 2 as $\checkmark_c$.

It can also be seen that many-to-many relationships would not generally produce a valid labeling. According to the constraints presented in Section 3.1, a many-to-many relationship is meaningless in the MD model. Nevertheless, there is one case in which we may consider many-to-many relationships, since we could eventually relax disjointness to identify derived measures, as explained in Section 3.1.1; this exception, in which disjointness would be relaxed a posteriori, is shown in Table 2 as $\checkmark_d$.

Finally, and as previously stated, we would like to remark that a node required to play a dimensional role by an edge label, can only be labeled as *L* and not as *CDM* or *CD*. Although these two labels represent hybrid nodes (and thus, they also contain dimensional data), their semantics are different from those of the *L* label. Importantly, edges relate nodes, and they determine the role that the related nodes may play according to the join conditions. Consider again Table 2. A node may play a **level** role whether: (i) it is placed in the to-one end of a relationship (see second, fourth and fifth column) or (ii) it is placed in the to-many end of a *Level–Level* one-to-many relationship (see second column).

By definition, the cardinality of factual data within a hybrid node is greater than (or in a degenerate case, equal to) that of the dimensional data it contains. Thus, in the (i) case, when an edge relates a node *n* to a hybrid node *h* by means of a to-one

Table 2
Valid multidimensional relationships in a relational schema.

Multiplicity	*Level–Level*	*Cell–Cell*	*Level–Cell*	*Cell–Level*
1 − 1	✓	✓	✓	✓
1 o − 1	✓	✓	$\checkmark_c$	✓
1 o − o 1	✓	✓	$\checkmark_c$	$\checkmark_c$
N − 1	✓	✓	×	✓
N o − 1	✓	✓	×	✓
N o − o 1	✓	✓	×	$\checkmark_c$
N − o 1	✓	✓	×	$\checkmark_c$
N − *M*	×	$\checkmark_d$	×	×
N − o *M*	×	$\checkmark_{dc}$	×	×
N o − *M*	×	$\checkmark_{dc}$	×	×
N o − o *M*	×	$\checkmark_{dc}$	×	×

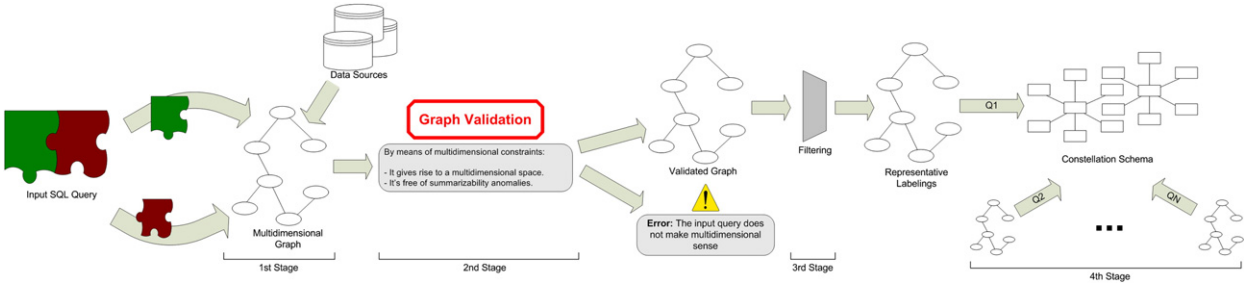

Fig. 7. Summary of the MDBE process.

relationship, the link relates *n* to the factual data in *h*. Otherwise, if the link were relating *n* to the dimensional data in *h*, it would not raise the to-one multiplicity. For this reason, hybrid labels cannot be used in this case. In the (ii) case the reason is subtler. According to Table 2, the node in the to-many end may represent a **level** (see second column) or a **Cell** (see third and fifth column). However, labeling it as a hybrid node entails that this node contains factual and dimensional data and, by the same reasoning as in the previous case, we are relating the factual data in *h* (i.e., the hybrid node) to the data in *n* (i.e., its counterpart node) by means of a many-to-one relationship. For this reason, the semantics of this edge would capture a *Cell–Level* or a *Cell–Cell* relationship (depending on the role of *n*), but never a *Level–Level* relationship. Indeed, a *Level–Level* relationship can only be obtained by considering *h* to play a strict dimensional role (i.e., labeling it as *L*).

Summing up, from the perspective of the edge labeling process and concerning hybrid nodes, factual data is of more relevance than dimensional data. Note that this is sound with the hybrid node definition: they contain factual data (and thus, like any other **Cell**, the *link attributes*) and additional dimensional data (that in the general case will introduce redundancy of data).

4. The MDBE method

The MDBE method has two inputs: the end-user information requirements (expressed as SQL queries) and the logical model of the data sources. As output, our method produces a constellation schema from the data sources, which allows the user to retrieve the data requested in the input requirements. In this scenario, each query is analyzed to derive an MD schema that meets the information requirements. This automatic process is depicted in Fig. 7 and can be divided into four different stages:

- For each input query, the first stage (see Section 4.1) extracts the MD knowledge contained in the query (i.e., the MD role played by each concept in the query and the conceptual relationships between concepts), which is properly stored in the MD graph. For this purpose, we apply the labeling methods discussed in Section 3.2. In this stage, the role played by the data sources will be crucial in inferring the conceptual relationships between concepts.
- The second stage (see Section 4.2) validates the MD graph created in the first stage according to the constraints introduced in Section 3.1. The aim is to check whether the concepts and relationships stated in the graph collectively produce a data cube. From the graph building perspective, the first stage of the MDBE method is designed to derive an MD labeling (i.e., label attributes, nodes and edges) to be validated in the second stage (i.e., checking the overall soundness of the graph). Therefore, this stage determines whether we would be able to use a set of MD operators to retrieve data requested in the input query from the MD schema represented by the MD graph. If the validation process fails our method ends, since the required data cannot be analyzed from a MD perspective (i.e., we are not be able to retrieve the requested data simply by using MD operators). Otherwise, the resulting MD schema is directly derived from the MD graph.
- The third stage (see Section 4.3) finds the most representative results among those obtained. The step in which new MD concepts are discovered may introduce new results (i.e., labelings) of potential interest, and we introduce a rule for determining which results should be presented to the user.
- Finally, the fourth stage (see Section 4.4) conciliates the MD schemas obtained for each query. The result is a minimal constellation schema subsuming each of the schemas obtained for the input queries.

Importantly, MDBE establishes a framework that can be used incrementally: by launching queries we can see the impact on the final conceptual schema. This feature facilitates the maintenance of the MD conceptual schema.

4.1. First stage: concept labeling

The first stage is designed to build the MD graph in 6 steps by applying the labeling standards introduced in Section 3.2. In this section, we introduce a detailed algorithm in pseudo-code (the *MDBE algorithm*) for implementing the first MDBE stage. This algorithm is followed by a brief explanation and an example of the execution of each step (based on the TPC-H schema). For the

purposes of the study, the comprehensibility of the pseudo-code took priority over its performance (nevertheless, some optimizations have already been applied for its implementation in the MDBE tool):

```
declare MDBE ALGORITHM as
1. For each table in the FROM clause do
   (a) Create a node and Initialize node properties;
2. For each attribute in the GROUP BY clause do
   (a) Label attribute as Level;
   (b) node=get_node(attribute); Label node as Level;
   (c) For each attr2 in follow_conceptual_relationships (attribute, WHERE clause) do
       i. Label attr2 as Level;
       ii. node=get_node(attr2); Label node as Level;
3. For each attribute in the SELECT clause not in the GROUP BY clause do
   (a) Label attribute as Measure;
   (b) node=get_node(attribute); Label node as Cell with Measures selected;
4. For each comparison in the WHERE clause do
   (a) attribute=extract_attribute(comparison);
   (b) if !(attribute labeled as Level) then
       i. Label attribute as Descriptor;
       ii. node=get_node(attribute); Label node as Level;
   (c) For each attr2 in follow_conceptual_relationships(attribute, WHERE clause) do
       i. if !(attribute labeled as Level) then
          A. Label attribute as Descriptor;
          B. node=get_node(attribute); Label node as Level;
5. For each join in the WHERE clause do
   (a) /* Notice a conceptual relationship between tables may be modeled by several equality clauses in the WHERE*/
   (b) set_of_joins=look_for_related_joins(join);
   (c) multiplicity=get_multiplicity(set_of_joins); relationships fitting={};
   (d) For each relationship in get_allowed_relationships(multiplicity) do
       i. if !(contradiction_with_graph(relationship)) then
          A. relationships fitting=relationships fitting+{relationship};
   (e) if !(size of (relationships fitting)) then return notify_fail("Node relationship not allowed");
   (f) Create an edge(get_join_attributes(set_of_joins)); Label edge to relationships fitting;
   (g) if (unequivocal_knowledge_inferred(relationships_fitting)) then propagate knowledge;
6. for each g in New_Knowledge_Discovery(graph) do
   (a) output+=validation_process(g); //A detailed pseudo-code of this function may be found in Section 4.2
   return output;
```

The algorithm analyzes each query clause according to Definition 1:

Step 1: Each table in the FROM clause is represented as a node in the MD graph. As presented in Section 3.2, MDBE will try to label every node, attribute and edge depicted in the query. Each node will keep track of relevant metadata inferred during the process. Specifically, we retain relevant metadata related to the query and referring to the data cube retrieved (if it makes MD sense): the data cube base and *compatibility* information.

Example: Consider the TPC-H business question #5 (Q5) that *"lists the revenue volume done through local suppliers"*. We will present, a detailed view of each step for Q5. The SQL translation of this query can be found in Section 1. In this first step, the graph initially has six nodes: customer, orders, lineitem, supplier, nation and region.

Step 2: This step is designed to find explicit dimensional data used to arrange the MD space. According to [C3], the GROUP BY clause (see [C1]) must fully functionally determine data. Thus, fields in this clause represent interesting perspectives from which to base data analyses. In addition, fields joined to these attributes in the WHERE clause will also be labeled as dimensional data (since joins represent conceptual associations stated in the end-user requirements [C1]).

Current methods rely on foreign keys to identify dimensional data, so results depend on the degree of normalization of the data sources (see Section 2 for further information). In our approach we are not tied to design decisions affecting the data source logical schemas and can identify them from the requirements. For example, when the user states relationships not depicted in the logical schemas of the data sources (for instance, data grouping). Consequently, every attribute identified in this step is labeled in the MD graph as an interesting **level** of analysis.

In these steps, each time an attribute is labeled, the label of the node to which it belongs will be properly updated according to the decision diagram shown in Fig. 6. Finally, we add the identified data cube base to the graph metadata.

Example: Attribute n_name from node nation is labeled as a **level** and accordingly (see Fig. 6), nation is labeled as a node containing dimensional data (i.e., *L*). Furthermore, to propagate that knowledge, we verify any concept association in the WHERE clause in which n_name is involved. However, there is no join involving that attribute. If c_nationkey had been used in the GROUP BY

clause instead of n_name, s_nationkey and n_nationkey would have been identified as **dimensional concepts** as well since there are two joins in the WHERE clause relating all of the attributes (i.e., *c_nationkey = s_nationkey* and *s_nationkey = n_nationkey*). Finally, we store the n_name as the data cube **base** in the graph metadata.

Step 3: This step is designed to find explicit factual data. Aggregated attributes in the SELECT clause (see [C1]) play a **measure** role. However, if the input query does not contain a GROUP BY clause we do not have to aggregate **measures** in the SELECT clause, and this step cannot identify them (these types of **Cells** and those not containing **measures** will be identified in Step 6). If the query does not perform a GROUP BY, we store the primary key used as the data cube **base** in the graph metadata (see Fig. 4 for further details). Finally, we also track the *compatibility* information identified in the node metadata.

Example: In this step, l_extendedprice and l_discount are identified as **measures**, and accordingly, table lineitem is labeled as a **Cell with measures** (*CM*). We also add to the graph metadata the compatibility information stated in the query: the (l_extendedprice*(1 − l_discount)) can be summarized by using sum function for all the **dimensions** in the data cube **base** (i.e., n_name; see previous step).

Step 4: This step is designed to find explicit dimensional data used to restrict the MD space. Since a selection (i.e., a comparison between an attribute and a constant value) must be carried out over dimensional data (see [C1] and [C7]), this step labels attributes as **dimensional concepts** looking for comparisons in the WHERE clause, following the concept association criteria presented in step 2. Attributes identified in this step are labeled as **descriptors** unless they have been used to arrange the MD space (in this case they would have been labeled as **levels** in Step 2).

Example: The SQL query #5 contains three comparison clauses between attributes and constants in the WHERE clause (*r_name = '[REGION]'*, *o_orderdate >= '[DATE]'* and *o_orderdate < '[DATE]' + '1' year*). Consequently, r_name and o_orderdate are labeled as **descriptors**. Accordingly, orders and region are labeled as dimensional data (*L*). In this step, we again verify joins in the WHERE clause involving any of these attributes to propagate the MD knowledge through concept associations. However, none of the attributes are involved in a join.

Step 5: The previous steps are aimed at creating and labeling nodes and their attributes whereas this step creates and labels edges (i.e., concept associations). Conceptual relationships are depicted in an SQL query by joins in the WHERE clause (see [C1]). In the MD graph joins are represented as edges, and this step is designed to label them following the process described in Section 3.2.3.

A list of potential edge labels is inferred according to the multiplicity inferred for a conceptual association in the WHERE clause (see Table 2). These alternatives are checked prior to labeling the edge, and a label is overlooked if it contradicts current knowledge depicted in the graph. For example, this may occur if a node has already been labeled and the edge label requires it to be relabeled in an incompatible way. An incompatible labeling happens when a node labeled as *C*, *CM*, *CD* or *CDM* is required to play a dimensional role. Once every alternative has been validated there are two potential scenarios: we will either have been able to label that edge with at least one alternative, or we will not. In the first case the algorithm continues, and if we have been able to infer unequivocal knowledge for a given edge (i.e., if a unique edge label stands) this knowledge is propagated in cascade to the rest of the graph. However, in the second case the algorithm stops since we have identified a conceptual relationship that does not make MD sense.

Example: (i) We infer the relationship multiplicity for each conceptual relationship in the WHERE clause. In this example, each conceptual relationship is defined by a single-attribute join, although in practice they might be depicted by multi-attribute joins; for example, *l_orderkey = o_orderkey* represents a relationship between lineitem and orders. According to Table 1 (second row), this join produces a many-to-one relationship between lineitem and orders that allows zeros in the to-many side of the relationship (since o_orderkey is defined as the primary key of orders and l_orderkey is defined as a foreign key to o_orderkey).

(ii) Next, according to Table 2, this one-to-many relationship may represent a *Level–Level*, a *Cell–Cell* or a *Level–Cell* relationship. However, the *Level–Level* relationship contradicts current knowledge in the graph since lineitem has been labeled as *CM* and this edge label requires it to be labeled as dimensional data. In contrast, the *Cell–Cell* relationship is consistent with current knowledge depicted in the graph. Although orders has already been labeled as dimensional data in Step 4, according to Fig. 6 it could also be considered a hybrid node (see the NKD transition), which means that it could also be labeled as either *CDM* or *CD*. In this case, according to Fig. 4 it should be labeled as *CD* (since the query performs data grouping but there is no orders attribute aggregated in the SELECT clause). Finally, the *Level–Cell* relationship is allowed, so the current edge is labeled with both possibilities (*Level–Cell* and *Cell–Cell*). A graphical representation of the MD graph after Step 5 can be seen in Fig. 8.

Once these steps have been completed the MD graph has been deployed. Tables (i.e., nodes), attributes (i.e., node attributes) and their conceptual relationships (i.e., edges) are depicted in the graph, and every edge has been labeled. However, some nodes (if none of their attributes have been labeled) may have not been labeled. Specifically, explicit concepts requested by the user (and nodes to which they belong) will be labeled after Step 5. This is because when writing the SQL query of a given requirement we may need to introduce *intermediate* concepts to relate explicit concepts stated by the user. In general, nodes containing intermediate concepts remain unlabeled after Step 5 (unless they have been labeled by the propagation rule of Steps 2 and 4). In addition, some nodes already labeled after Step 5 may have potentially interesting alternatives, which can occur if the query structure does not clearly identify **measures** (see Step 3) or if we are looking for interesting factless facts. In this paper, we will refer to intermediate nodes and nodes with interesting alternative labels as *implicit* nodes.

As discussed in Section 2.1.2, we propose an intermediate solution for automatically derive new MD knowledge not considered by the user. In our approach, we focus on the implicit concepts of the query, and analyze the available labeling alternatives. The aim of this step is to determine how these alternatives would affect the output schema, deriving (in some cases) interesting analytical alternatives that may have been overlooked by the user.

Step 6: This step is designed to derive new MD knowledge from unlabeled nodes or, according to the NKD transitions in Fig. 4, to test alternative labels for nodes already labeled. Each unlabeled node can be considered to play a dimensional role (i.e., labeled as *L*) or a factual role (labeled as *C* or *CM*, according to Fig. 4). However, nodes with potential alternatives of interest will introduce an alternative label. For each possible combination of new labels, an *alternative graph* is created if the labels do not contradict knowledge already depicted in the graph. Subsequently, each of these graphs will be validated as explained in Section 4.2, and only those that make MD sense will finally be considered. Therefore, a query could produce several valid MD graphs. In that case, MDBE would be able to derive multiple MD schemas for a single query.

Essentially, this step guarantees that all of the possible MD labelings for the input requirements will be generated (each one represented as an alternative MD graph). As such, it is possible for all of the nodes in a given graph to be labeled as dimensional data. However, this type of graph is directly disregarded by our method because an MD graph must contain at least one **Cell** [C1].

Example: We have two unlabeled nodes (i.e., customer and supplier, labeled as *?* in Fig. 8) and three nodes that, according to Fig. 6, may play a factual role in addition to their current dimensional role (i.e., orders, nation and region, marked with an * in Fig. 8). For each combination that does not contradict knowledge depicted in the current graph, an alternative graph is generated.

After Step 6, we have 5 nodes with two potential labeling alternatives that produce 8 different MD graphs. Note that we do not generate 32 graphs (i.e., 2^5 combinations) because many of them are meaningless in the MD model. From Step 5, a given labeling is overlooked if it contradicts knowledge depicted in the graph. For example, consider the following labeling alternative in which customer, orders, supplier and region are labeled as *C*, whereas nation is labeled as *L*. According to the edge between region and nation, if region is labeled as *C* then nation should also be labeled as *C*; otherwise, it would not make MD sense (see Table 2). This type of contradiction removes 24 of 32 possible combinations. The remaining 8 combinations (shown in Table 3) will then be validated in the second stage of MDBE. As we will see, most of these will be invalidated, and only two will eventually be found to make MD sense (the last column of Table 3 shows which step invalidates which combination).

4.2. Second stage: multidimensional graph validation

In this stage we validate each of the MD graphs generated in the previous stage. The validation process also guarantees the *multidimensional normal forms* presented in [19] and [20] for validating the output MD schema. Again, we use a detailed algorithm in pseudo-code (the *validation_process algorithm*) to implement our method, followed by a brief explanation and an example of each one of its steps. This algorithm is called once for each alternative graph generated in Step 6 (see Step 6a of the *MDBE algorithm* in the previous section):

```
   declare VALIDATION_PROCESS as
7. If !connected (graph) then return notify_fail("Aggregation problems because of cartesian product.");
8. For each subgraph of Levels in the multidimensional graph do
   (a) if contains_cycles(subgraph) then
       i. /* Alternative paths must be semantically equivalent and hence raising the same multiplicity.*/
       ii. if contradiction_about_paths_multiplicities(subgraph) then return notify_fail("Cycles
           cannot be used to select data.");
       iii. else ask user for semantic validation;
```

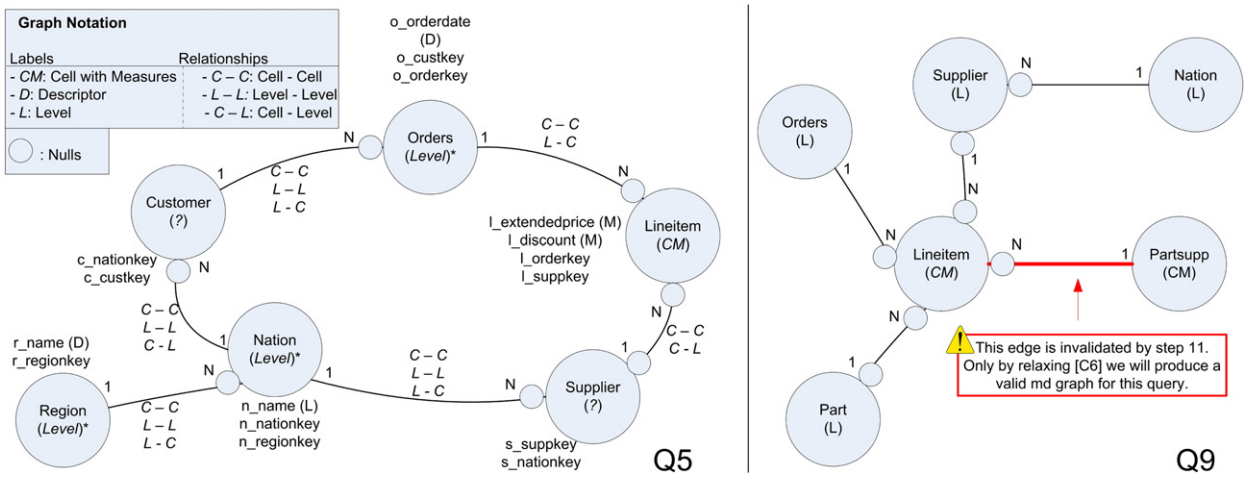

Fig. 8. Left, the MD graph for Q5 after Step 5; right, the MD graph for Q9 after Step 12.

Table 3
Graph labelings generated after the first stage of MDBE.

Id	Lineitem	Customer	Orders	Supplier	Nation	Region	Step
1	*CM*	*C*	*CD*	*C*	*CD*	*CD*	10
2	*CM*	*C*	*CD*	*C*	*CD*	*L*	10
3	*CM*	*C*	*CD*	*C*	*L*	*L*	9
4	*CM*	*C*	*CD*	*L*	*L*	*L*	9
5	*CM*	*L*	*CD*	*C*	*L*	*L*	9
6	*CM*	*L*	*CD*	*L*	*L*	*L*	OK
7	*CM*	*L*	*L*	*C*	*L*	*L*	OK
8	*CM*	*L*	*L*	*L*	*L*	*L*	8b

(b) **if** *exists_two_Levels_related_same_Cell*(*subgraph*) **then** return *notify_fail*("Non-orthogonal Analysis Levels");
(c) **For each** relationship **in** *get_1_to_N_Level_Level_relationships*(*subgraph*) **do**
i. **if** *left_related_to_a_Cell_with_Measures*(*relationship*) **then** return *notify_fail*("Aggregation Problems.");
9. **For each** Cell pair **in the** multidimensional graph **do**
(a) **For each** *1_1_correspondence*(*Cellpair*) **do Create** context edge between Cell pair;
(b) **For each** *1_N_correspondence*(*Cellpair*) **do Create** directed context edge between Cell pair;
(c) **If** *exists_other_correspondence*(*Cellpair*) **then** return *notify_fail*("Invalid correspondence between Cells.");
10. **if** *contains_cycles*(*Cells path*) **then**
(a) **if** *contradiction_about_paths_multiplicities*(*Cells path*) **then** return *notify_fail*("Cycles cannot be used to select data.");
(b) **else** ask user for semantic validation; **Create** context nodes(Cells path);
11. **For each** element **in** *get_1_to_N_context_edges_and_nodes*(Cells path) **do**
(a) **If** *CM_at_left*(*element*) **then** return *notify_fail*("Aggregation problems between Measures.");
12. **If** *exists_two_1_to_N_alternative_branches*(*Cells path*) **then** return *notify_fail*("Aggregation problems between Cells.");

Step 7: The MD graph must be *connected* to avoid the "Cartesian Product" ([C6]). Furthermore, the MD graph should be composed of valid edges that produce a path between **Cells** (factual data) and connected subgraphs of **levels** (dimensional data) surrounding it — these constraints will be properly checked in the following steps.
Example: In our example, the 8 MD graphs to be validated (see Table 3) are connected.
Step 8: This step validates **levels** subgraphs (i.e., subgraphs only containing **level** nodes) with regard to **Cells** placement: According to [C4], two different **levels** in a subgraph cannot be related to the same **Cell** (Step 8b); to satisfy [C5] and [C6], **level–level** edges raising aggregation problems in **Cells** with selected **measures** must be forbidden (Step 8c). Finally, every subgraph must represent a valid **dimension** hierarchy (i.e., not being used to select data) [C7]. Thus, we must be able to identify two nodes in the **level** subgraph which represent the *top* and *bottom* **levels** of the hierarchy, and if there is more than one alternative path between these nodes, they must be semantically equivalent (8a). As discussed in Section 3.1.1, this step may eventually relax [C5] and [C6] (i.e., disjointness) if required in Step 8c.

Example: i) Step 8a: In our example, none of the graphs contain a cycle within a **level** subgraph so that all of them satisfy this step. ii) Step 8b: Consider the alternative graph depicted in row 8 of Table 3. All of the nodes except for lineitem are labeled as **levels**. Therefore, this alternative does not preserve 8b, since orders and supplier belong to the same **level** subgraph and both are related to lineitem. Consequently, the validation process fails and this alternative is discarded because the **Cell** is related to two different points of the same analysis perspective (which does not make MD sense). iii) Step 8c: In our example, lineitem is the only node labeled as **Cell** with **measures**, but it does not raise any aggregation anomalies in any of the graphs.
Step 9: Cells determine MD data and must be related in the graph to produce a single **Cell** path. If this is not the case, they cannot retrieve a single data cube [C1]. For every pair of **Cells** in the graph, we aim to validate the paths between them as a whole, inferring and validating the multiplicity raised as follows: (i) if a one-to-one correspondence between two **Cells** exists, we replace all of the relationships involved in that correspondence with a one-to-one *context edge* between the two **Cells** (i.e., a context edge replaces the subgraph representing the one-to-one correspondence). As shown in Fig. 9.1, this means that a whole **Cell** CK is linked by one-to-one paths to the whole CK of the other **Cell**. (ii) Alternatively, if both CKs are related by one-to-many paths or the first CK matches the second one partially, we replace the relationships involved with a one-to-many directed context edge (see Fig. 9.1). (iii) From a data source perspective, many-to-many relationships between **Cells** should be invalidated because they do not preserve disjointness [C6]. Nevertheless, this step may eventually relax disjointness, as discussed in Section 3.1.1.

Example: In our example 3 of the 7 remaining labeling alternatives produce incoherent context graphs that do not satisfy the MD constraints. For example, the labeling alternative shown in the third row of Table 3 would produce a

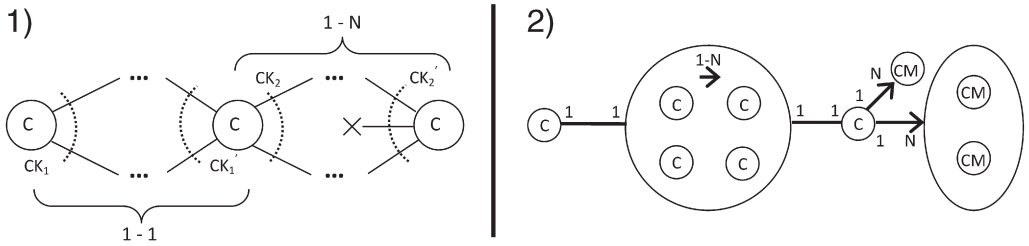

Fig. 9. Examples of cells paths in a context graph.

forbidden many-to-many relationship between customer and supplier in the context graph. There are only four viable alternatives for this step: if every node in the graph cycle is considered as factual data (rows 1 and 2 of Table 3) or if orders (row 6) or supplier (row 7) are considered to play a factual role. Any other alternative would produce an invalid context graph.

Steps 10, 11 and 12: The previous step validated the correspondences between **Cells**, whereas these steps validate the **Cell** path (MD data retrieved) as a whole: According to [C7], Step 10 validates cycles in the path of **Cells** to ensure that they are not used to select data, similarly to the validation of **levels** cycles (see 8a). Once the cycle has been validated, the **Cells** involved are clustered in a *context node* labeled with the cycle multiplicity, as shown in Fig. 9.2. According to [C5] and [C6], Steps 11 and 12 look for potential aggregation problems; the first looks for **Cells** with **measures** selected at the left side of a one-to-many context edge or node, and the second looks for alternative branches with one-to-many context edges or nodes each, which would produce a forbidden many-to-many relationship between the **Cells** involved (as depicted in the right side of Fig. 9.2). Finally, as in any step involving [C5] and [C6], this step may eventually relax disjointness as discussed in Section 3.1.1.

Example: i) Step 10: In our example, this would be the case if every node in the graph cycle was considered to play a factual role (row 1 of Table 3) or if all of them except region played a factual role (row 2 of Table 3). In both cases, identified cycles would not make MD sense because they do not preserve disjointness of lineitem (which contains **measures**). Consequently, both alternatives are discarded. ii) Steps 11 and 12: At this point, we only have two valid alternatives (rows 6 and 7), but neither produces aggregation problems with lineitem.

Another interesting example is the graph obtained from Q9 (see Fig. 8), in which a one-to-many relationship is shown between two **Cells** (lineitem and partsupp). According to Table 2 this edge is allowed between **Cells** but Step 11 invalidates it. This query does not preserve disjointness [C6] between both **Cells** with **measures** (a **Cell** with **measures** is selected at the left side of a one-to-many context edge). In this case, no result would be produced and, according to R1 (see Section 3.1.1 for a detailed definition of this rule) we relax [C6]. Now, we obtain a valid MD graph, but the user is informed of the situation and asked to validate the result obtained. In this example, according to the query semantics, we are calculating a derived measure (weighting the l_quantity with the ps_supplycost), which determines the profit made on a given line of parts (see [1]). Clearly, this derived measure (and thus, this query) makes sense and must be considered a valid MD data cube.

The second MDBE stage would eventually have validated each graph as corresponding a data cube, and only those guaranteeing every step discussed above would be presented to the user.

Example: At the end of the validation process, two of the eight initial labeling alternatives (see rows 6 and 7 of Table 3) are sound and make MD sense. Therefore, MDBE would produce two different MD schemas that would satisfy Q5 (these can be determined from Fig. 8 and rows 6 and 7 of Table 3).

4.3. Third stage: finding representative results

Step 6 in the first stage of the MDBE method may produce several alternative graphs for the same query. Unlabeled nodes (and those with interesting alternatives, according to Fig. 6) are proved to be factual and dimensional data in alternative graphs, which are validated in the second MDBE stage. Eventually, those graphs that make MD sense will be considered in the conciliation process (see next section). Consequently, more than one MD schema can be produced for a given query. However, an alternative graph could make MD sense but not represent a new and potentially interesting analytical perspective. Indeed, dimensional data could always be considered as an alternative factless fact, although in most cases it will not be relevant to the end-user. Therefore, this step is designed to determine the representativeness of new alternatives produced by Step 6, according to the following rule:

R2: If, for a given query, we obtain two sibling graphs that suggest analyzing a given dimensional node as a factless fact, we disregard the potential factual role of that node.

Two sibling graphs differ only in the labeling of one node. Therefore, they have exactly the same labels except for one node, which is considered to play a factless fact role in one graph and a strict dimensional role in the other. As an

example, consider the following table, which depicts the alternative graphs obtained after the validation step for a given query:

Id	Node *A*	Node *B*	Node *C*	Node *D*
1	CM	CD	C	L
2	CM	L	C	L
3	CM	L	C	CD

According to the previous definition, alternative Graphs 1 and 2 (which only differ in the label of B), and Graphs 2 and 3 (differing in the label of D) are siblings. In this case, and according to *R2*, for the first sibling relationship we disregard the first graph and choose Graph 2 as the most representative; for the second pair we disregard Graph 3 and choose Graph 2 again. Eventually, this query will produce a single MD schema. In short, sibling graphs do not provide new interesting analytical perspectives. MDBE uses them to analyze the potential factual data that a **dimension** may contain. However, in most cases, the end-user would not be interested in this type of analysis. Knowledge inferred from Step 6 is therefore disregarded when it produces sibling graphs and is only considered and presented to the user in one of two cases. (i) In the first case, the knowledge identifies a dimensional node that may also play a factual role with **measures**. This scenario can only arise in a query without data grouping, in which case Step 6 would identify an atomic **Cell** with **measures** (see Section 3.2.2 for further details). Note that this type of node is relabeled in Step 6 as *CDM* and, as such, does not fit the sibling definition and will not be pruned in this step. (ii) In the second possible case, we have a factless fact that cannot play a dimensional role (i.e., there is no sibling graph for this labeling).

Example: The latter case (ii) occurs in the Q5 validation process. Consider Table 3. The two valid labels are shown in rows 6 and 7; since they do not have sibling graphs, none will be pruned in this step. Essentially, MDBE highlights that Q5 will make MD sense if either supplier or orders plays a factless fact role (the query semantics can be checked to confirm that this is consistent with the query definition).

4.4. Fourth stage: conciliation

MDBE validates each input requirement and obtains a potential set of MD schemas for each query (see the three previous stages presented above). In this section we present an algorithm that conciliates the results for the input queries into a minimal set of schemas covering all of the queries.

Before proceeding to the conciliation, a pre-process must be carried out to *normalize* the MD graphs; each hybrid node in every MD graph is normalized. This means that any node labeled as *CDM* or *CD* will produce two different nodes: according to the discussion introduced at the end of Section 3.2.3, hybrid nodes contain factual data (and thus, like any other **Cell**, the *link attributes*) and additional dimensional data (that in the general case will introduce redundancy of data). In fact, hybrid nodes could be represented as factual data (i.e., a node labeled as *CM* or *C*) related by a many-to-one (or in a degenerate case, a one-to-one) relationship to dimensional data (i.e., a node labeled as *L*); in other words, we could normalize them. We then apply the following algorithm (for clarity, the comprehensibility of the algorithm took priority over its performance):

- (1) MDBE looks for all the facts identified in the MD graphs, and creates a new factual class[3] for each one (every class will eventually produce an MD schema at the end of the conciliation process). Two other tasks are performed in this step: i) we enrich each class by adding the measures identified in the graphs as attributes of the factual class; and ii) we draw the conceptual relationships between facts depicted in the graphs by semantic relationships between classes.
  Example: Consider a simplified scenario of the TPC-H case study in which we only need to conciliate the MD graphs created for Q5 and Q9 (see Fig. 8). First, we create four factual classes (lineitem, orders, supplier and partsupp) for each node labeled as either *CM* or *C*. Then, we add the measures identified in these graphs to each class. Consequently, l_extendedprice, l_discount (from Q5 and Q9) and l_quantity (from Q9) are added to the lineitem class and ps_supplycost (from Q9) is added to partsupp. The remaining classes will not contain measures as they were identified as factless facts (see Q5).
  In addition, since partsupp is related to lineitem in Q9, we keep track of this conceptual relationship by drawing a semantic relationship between the two classes. The same is done with lineitem and order, and lineitem and supplier (Q5).
- (2) Next, we conciliate the dimension hierarchies identified by the input queries. We first look for compatible hierarchies. Two hierarchies are compatible if they share their *atomic level*.[4] Every set of compatible hierarchies must be conciliated (i.e., produce a single dimension subsuming all of them). This process is carried out by checking the *hierarchies graphs*. From the perspective of the MD graph, a hierarchy is represented by the subgraph containing the nodes that form the dimension. For example, the customer → nation → region hierarchy identified in Q5 (see Fig. 8 and Table 3) is directly derived from the subgraph formed by these three nodes.

[3] In this step we are devising the MD conceptual schema. We therefore talk about classes and attributes in this section, but we could use the notation from any conceptual MD model. For example, [2].

[4] An atomic level is the finest granularity level within a dimension hierarchy and is directly related to the fact [2].

Therefore, a hierarchy *h subsumes* a hierarchy h' if the subgraph representing h' is contained (except for the descriptors) in the subgraph representing *h*. At this point, it should be noted that a one-to-one relationship is contained in a one-to-many or a many-to-one relationship. Having said that, we conciliate a set of compatible dimensions by applying the following properties iteratively:

- (2.1) If a given hierarchy *h* subsumes a hierarchy h' and h' also subsumes *h*, both hierarchies are equivalent and we only need to keep one of them aligning all of the descriptors of both dimensions. The other hierarchy must be removed from the set of compatible hierarchies.
- (2.2) Alternatively, if *h* subsumes h' and h' does not subsume *h*, the descriptors of h' are mapped to *h*, and h' is removed from the set.
- (2.3) Finally, if *h* does not subsume h' and h' does not subsume *h*, they are conciliated as follows: i) first, we conciliate (by keeping the common structure and aligning their descriptors) the overlapping part shared by the hierarchies (note that, by definition, they will share at least their atomic levels; see the *compatible hierarchies* definition above); second, ii) we draw two *alternative branches* in the resulting hierarchy, one branch for each disjoint part of the subgraphs.

Example: In our example, Q5 and Q9 provide two sets of compatible dimensions (i.e., the first set contains the supplier → nation → region from Q5 and supplier → nation from Q9, and the second set contains orders_dim from Q9 and Q5, and orders_dim → customer → nation → region from Q5). In this scenario, conciliation of the two sets corresponds to the second case presented above: one hierarchy is contained in the other but the reverse is not true. Therefore, we keep the richest hierarchy and enrich it with the descriptors of the discarded one.

The conciliated dimension hierarchies and those that are not compatible with any other are depicted in the MD schema. For example, consider the orders_dim → customer → nation → region dimension; its atomic level was related to lineitem and orders. Consequently, we relate this new conciliated dimension to these two factual classes. By carrying out this process, we will obtain a star-schema for each factual class identified. Note that conciliated dimensions enrich the conceptual schema: they provide other factual classes with new analytical perspectives considered in other star-schemas. For example, orders only considered the orders_dim level, whereas it now has a detailed conciliated hierarchy.

- (3) Finally, a pruning step is carried out. MDBE identifies those star-schemas that are semantically poor. We can also introduce a non-representative requirement, which would produce an unneeded star, for example: *every star-schema composed of just one dimension is proposed to be disregarded* (note that we could use any other criterion introduced in the literature [5,34], if desired). However, the final decision is taken by the user, since the star-schema is derived from the end-user requirements and he/she must decide if it really makes sense or it was an error.

  Example: In the TPC-H case study, this would be the case of supplier and customer. Both have been identified as factless facts during the process, but their star-schemas are rather simple (one **dimension** each). After considering the requirements from which they were derived, we may decide to eliminate them (as was the case in the final schema shown in Fig. 3).

Two main points should be made about this process. First, it does not introduce a *summarizability* problem, because we are only merging compatible labels (i.e., factual data and only fully compatible dimensional data). It is also very important to note the relevance of semantics in the conciliation process. In a data warehousing design task, semantic relationships must be carefully considered. For example, two different relationships between the same concepts *A* and *B* must produce two different perspectives. The reason is clear: each relationship relates a different set of instances from the two classes and, therefore, produces two different analytical perspectives. This explains why two dimension hierarchies such as A → B and A → C → B cannot be conciliated as A → C → B. Had we proceeded like this, we would have lost semantics. It should be considered that we are working with relational sources, so if we travel from A to B along two different paths there must necessarily be two different conceptual paths between them. As explained in Step 2.3 of the conciliation process, the hierarchies should be conciliated as: B ← A → C → B; i.e., with two alternative branches starting from A (the common part).

The second point is that the *orthogonality* of the MD spaces that may be produced is not lost, since we keep track of the metadata inferred from each query at the constellation level (see Section 4.1; steps 2 and 3). Note that each input query represents a data cube of interest. Consequently, our output schema retains the metadata about these datacubes: the MD space depicted (i.e., the cube base) and the information about the compatibility of the data summarization performed (i.e., which function may be used for their measures and in which dimensions). This type of information will be relevant for the OLAP tool once it has been implemented.

Finally, this stage, like the three previous stages, is fully automatic and we therefore obtain a star-schema for each fact identified; this, as a whole, produces a constellation schema (see Fig. 3). Note that this figure only shows **facts**, **measures** and **dimension hierarchies** identified in the process, whereas **descriptors** have been overlooked to avoid disrupting the final result. Nevertheless, it should be stressed that MDBE works at the attribute level and keeps track of the role assigned to each attribute when deriving partial schemas from each query. Consequently, we are able to split some tables (for example, orders produced two different concepts in the MD schema, since the dimensional attributes contained in the relational orders table are represented explicitly in orders_dim).

5. Analysis of the TPC-H empirical results

In this section we discuss several statistics about the overall TPC-H case study. MDBE was carried out for the 22 TPC-H queries that together produced the constellation schema shown in Table 3. Below, we focus on four interesting aspects of this case study: the amount of input queries needed to obtain an interesting output, the output correctness, the extra knowledge obtained in the output thanks to the novel contributions of the MDBE method, and the computational complexity of the algorithm.

5.1. Input queries needed

One interesting aspect of the study is the number of queries needed to produce the resulting conceptual schema. In the output schema we identify 3 factual classes (containing 9 measures) and 9 dimension hierarchies (containing a total of 18 level classes and 39 descriptors):

- In the worst case, we would need 11 queries to identify all of the factual classes and dimension hierarchies in the MD model. In other words, some queries are redundant and are not relevant to the final result. Had we executed them in the worst possible order, we would have identified all the MD classes and most of the attributes even with 11 queries (8 out of 9 measures and 17 out of 39 descriptors).
- In contrast, in the best case, we would have been able to identify all of the factual classes and dimension hierarchies with just 4 queries (and 6 out of 9 measures and 10 out of 39 descriptors — it would also be possible to give more relevance to descriptors and then identify 16 out of 39 descriptors but 5 out of 9 measures, also with 4 queries). For example, Q5 is a key requirement as it identifies 4 dimension hierarchies and 2 factual classes. Indeed, Q5 and Q9 identify all three factual classes and 4 measures of the resulting MD schema.

If we considered a random order of input queries (i.e., without any consideration other than choosing the order of the query execution at random) we would need an average of 8 queries to identify the main structure of the schema (i.e., facts, measures and dimension hierarchies). This result is sound as it is relatively easy to identify the MD classes with only a small number of queries. Indeed, the MD design task proposed in this paper is incremental, and it is up to the user to decide when to stop adding new queries. Once most of the structure has been defined, it can be customized as in traditional approaches.

For example, consider a case in which we are satisfied with the number of facts, measures and dimension hierarchies identified by MDBE. Suppose that we have identified the 3 factual classes, the whole dimension hierarchies and the 9 measures. In an average case, these concepts can be defined with 8 queries and we would have approximately 14–19 descriptors (depending on the input queries used). To proceed further, it would be easier to identify the rest of the descriptors among the dimensional data table attributes than by launching new queries. Note that MDBE can easily support this last step: we can browse the attributes of each level identified and let users add those that are of interest to them.

To continue with our example, at this point the region level class would contain r_regionkey and r_name but the r_comment attribute in the relational table would not have been selected yet. However, it would be easier to browse the region attributes and add r_comment to the output schema than to launch a new query specifically for the purpose.

5.2. Output correctness

As discussed in Section 1, the Star-Schema Benchmark (or SSB) [27] presents a MD logical schema that is derived *manually* from the TPC-H schema. This schema was devised to improve the querying performance of the data warehouse by denormalization. Data denormalization, achieved by implementing a logical star-schema [18], is fairly common in data warehouse systems and is used to speed up certain queries [18]. Unlike SSB, the MDBE method produces a conceptual schema, but the SSB logical schema can be obtained by applying the same design decision taken by the SSB authors: to implement the output schema as a logical star-schema (i.e., denormalize dimensional data as much as possible). Therefore, we can obtain the SSB *logical* schema in one of the following ways:

- By denormalizing the **dimensions** of analysis as much as possible. Therefore, nation and region data will be denormalized within supplier, partsupp and customer. Dimensional attributes that produced new level classes (i.e., orders_dim, lineitem_dim and partsupp_dim) must also be denormalized.
- By merging the three schemas that form our constellation (since we have three factual classes) into a single schema. Therefore, lineitem, orders and partsupp will produce a single table. This decision is consistent with our conceptual schema, which relates these three facts (therefore, our conceptual schema allows us to "drill-across" [6] between them, obtaining the same results as if they were merged in a single table).

To summarize, the MDBE method can derive the same MD schema as SSB but, in contrast to the SSB, it does so in an automated way.

5.3. Computational complexity and performance

We present an in-depth analysis of the 22 business queries in the TPC-H benchmark and use the findings as the basis for discussing the complexity and performance of the algorithm. Table 4 summarizes some of the relevant statistics for each query. In brackets, Statistics for their subqueries, if any, are shown in brackets (briefly, subqueries must be validated by their own, as they can be considered a materialized factual table and must therefore make MD sense as well). The first column represents the query id and the other columns should be read as follows: the second column shows the number of implicit nodes we have for the query (i.e., nodes that remain unlabeled up to Step 6 or which are relabeled at that point). According to the number of implicit nodes, we can produce $2^{\#implicit\ nodes}$ label combinations (note that Step 6 only tries two label alternatives for unlabeled nodes). However, as discussed previously, many of these combinations are not even generated, since they raise contradictions with knowledge already depicted in the graph and, therefore, do not satisfy the MD constraints. Ungenerated combinations are shown in the third column,

Table 4
MDBE statistics for the TPC-H case study.

Id	Implicit nodes	Edges contradict.	Alternative graphs	Validation process	Siblings	#Results	Factless facts	New dim. attrs.
Q1	0	0	1	0	0	1	0	3
Q2	5(3)	23(4)	9(4)	1(1)	7(2)	1(1)	0	0(1)
Q3	2	1	3	0	2	1	0	(1)
Q4	1(1)	0	2(2)	1(1)	0	1(1)	0	2(1)
Q5	5	24	8	6	0	2	2	0
Q6	0	0	1	0	0	1	0	3
Q7	5(6)	20(51)	12(13)	0(1)	11(11)	1(1)	0	1(1)
Q8	7(8)	98(225)	30(31)	0(1)	29(29)	1(1)	0	0
Q9	4(4)	4(4)	12(12)	0	11(11)	1(1) †	1(1)	0
Q10	3	4	4	0	3	1	0	1
Q11	2(2)	1(1)	3(3)	0	2(2)	1(1)	0	2(0)
Q12	2(2)	1(1)	3(3)	1(1)	1(1)	1(1) †	5(5)	0
Q13	2	1	3	1	1	1	1	0
Q14	1(2)	0(1)	2(3)	0(1)	1(1)	1(1)	0(1)	1(0)
Q15	1(2)	0(1)	2(3)	0(1)	1(1)	1(1)	0(1)	1(0)
Q16	2(1)	1(0)	3(2)	2(1)	0	1(1)	1(1)	0
Q17	1(1)	0	2(2)	0	1(0)	1(1)	0	1(1)
Q18	2(1)	1(0)	3(2)	0(1)	2(0)	1(1)	0	0(1)
Q19	1(1)(1)	0	2(2)(2)	0	1(1)(1)	1(1)(1)	0	3(3)(3)
Q20	2(1)(0)	1(0)(0)	3(2)(1)	(1)(1)(0)	1(0)(0)	1(1)(1)	1(1)(0)	0(0)(3)
Q21	4(1)(1)	9(0)(0)	7(2)(2)	1(1)(1)	5(0)(0)	1(1)(1)	0(1)(1)	0
Q22	0(0)(1)	0	0(0)(2)	0(0)(1)	0	1(1)(1)	0(0)(1)	2(2)(0)

and the fourth column shows the number of many alternative graphs (to be validated) generated for each query. The fifth column shows the number of MD graphs that are discarded in the MDBE validation stage, and the sixth column shows the number of graphs that are collapsed, according to the sibling rule introduced in Section 4.3.

The MDBE tool execution time for the TPC-H benchmark is negligible (~1 s[5]). Our approach only has a potential combinatorial explosion in Step 6 (note that the conciliation process carried out — see Section 4.4 — is linear regarding the number of schemas obtained and, therefore, it does not raise the computational complexity of the MDBE process). However, most combinations of labels generated by Step 6 are discarded on the basis of edge semantics (see the third column), which produces a tractable algorithm. In all queries, the final set of graphs to be validated is considerably smaller than $2^{\#implicit\ nodes}$ (see the fourth column). This statement is based on the empirical results provided, but we can intuitively identify why Step 6 will never generate an exponential number of combinations: the whole MD graph must be semantically valid, which means that several nodes and edge labelings will not be allowed. For example, Table 2 shows 25 forbidden combinations (we count those allowed by relaxing [C5] and [C6], as they will only be considered if no result is generated. Thus, if considered, we obtain just one result at most). Consequently, many combinations of labels will fail to make MD sense. Furthermore, the first five steps of the MDBE process always label most of the nodes/edges for MD requirements. Only *implicit* nodes (see the discussion prior to Step 6 in Section 4.1 for further details) can produce unlabeled nodes. Consequently, the exponent value in the $2^{\#implicit\ nodes}$ expression will be typically a small number. For example, in the statistics shown, only two queries (Q7 and Q8) have more than 6 implicit nodes. However, Q7 invalidates 51 of 64 alternative graphs (and is computed in ~0.1 s) according to edge semantics, and 225 of 256 in Q8 (computed in ~0.12 s). Let us consider a query with a large number of implicit nodes. In this case, we will only generate all possible combinations of labels (i.e. exponential computational complexity) if these tables are related by one-to-one relationships with a double FK pointing between each pair (see Tables 1 and 2). However, this would be an unlikely real-world scenario and, in any case, an SQL query is unlikely to have a large number of tables in its FROM clause. The MDBE validation process takes an average of 0.007 s, so in the worst-case scenario discussed above a query with 10 unlabeled tables in the FROM clause would generate 1024 label combinations, which would be processed in 7.168 s.

5.4. Additional output inferred

In this section we measure the impact of the main contributions of MDBE on the output. The first six columns of Table 4 show statistics about the MDBE process, whereas the last three show statistics about the results for each query. The seventh column shows the number of final star-schemas retrieved by MDBE for the corresponding query (i.e., the number of alternative graphs that have been completely validated). The next column shows the number of factless facts identified for the query, and the final column shows the number of new dimensional attributes identified in the process (if any). These attributes are those identified as dimensional data in a hybrid node (i.e., dimensional attributes in *CD* and *CDM* nodes). The † symbol denotes that [C5] and [C6] have been relaxed to produce the output result for that query. These results highlight some interesting features of our method:

- First, the MDBE validation process. In this stage, 14 of 22 queries invalidate alternative graphs that may initially appear to be correct. Consequently, simply labeling nodes is not sufficient and it is necessary to consider the semantics of the results proposed as a whole.

[5] The computer used in these test was equipped with an Intel Core 2 Duo 2.16 GHz processor, 3 GB of RAM.

- Our process for obtaining new knowledge from implicit nodes is carried out in most of the queries. In the TPC-H case study, the sixth step of our method labels (or relabels) nodes in 20 of 22 queries (see the second column), which reveals the importance of this step in retrieving additional information to that explicitly requested by the user.
- Denormalization is very important in our approach. Although the TPC-H logical schema is normalized and well-formed, from an MD perspective some of its tables contain degenerated dimensions, as a result of which many nodes have been labeled as *CD* or *CDM* during the labeling process. For example, in one of the solutions proposed for Q5, orders is labeled as a factless fact *CD* with o_orderdate as a degenerated dimension [18]. This result is sound, since time and date are typical analysis **dimensions** in any data warehouse and, in fact, some current methods always complement their results with these two **dimensions** (e.g., [28]). Therefore, in our final result these concepts are explicitly stated according to their MD role. In our example, 15 of 22 queries identify at least one new dimensional attribute for the corresponding query (see the final column); for example, shipdate, returnflag and shipmode from lineitem. In addition, we may also identify *dual* attributes (see Section 3.2.1); for example, ps_supplycost from partsupp. Consequently, the resulting conceptual schema contains a **dimensional attribute** and a **measure** derived from this relational attribute.
- Our process can identify interesting additional information that is traditionally overlooked by other methods. Our method supports factless facts (see column 8) and can also identify new derived measures (see †) that are not captured in the relational sources but which can be derived from them.

6. Conclusions and further work

In this paper we presented a novel approach for supporting the data warehouse design process. The MDBE method is a hybrid approach for automatically generating MD schemas from end-user requirements and relational data sources. This method differs from previous approaches by combining the best features of each design paradigm: (i) it considers requirements as first-class citizens within a largely automated approach; (ii) it improves the quality of the final output by improving communication between the supply-driven and demand-driven stages — in fact, the two stages are merged in MDBE and depend on each other to produce the output schema; (iii) it constitutes a novel approach that helps users to discover the analytical potential of the data sources; and (iv) it can identify new concepts such as specializations or new measures derived from the data sources. Importantly, the conceptual schemas produced by MDBE are derived from a validation process of the input requirements, which ensures that they are sound and meaningful. Consequently, this process can be used to validate MD requirements and to determine whether an MD system is required. For example, we can validate our information needs, and if they are MD it would be sound to use an OLAP tool in the organization in question.

We also demonstrated the practical application of our method, using the TPC Benchmark H case study to illustrate the potential of the approach and to provide a detailed example of how the method is executed. The MDBE method opens up a range of new research opportunities. For example, our approach is incremental and provides a solid foundation for the maintainability and evolution of the conceptual schema, which is a topic that has gained importance in recent years [31].

Acknowledgments

This work has been partly supported by the Ministerio de Ciencia e Innovación under project TIN2008-03863.

References

[1] TPC-H Version 2.8.0, http://www.tpc.org/tpch/default.asp, last access 30-09-2008.
[2] A. Abelló, J. Samos, F. Saltor, YAM^2 (Yet Another Multidimensional Model): an extension of UML, Information Systems 31 (6) (2006) 541–567.
[3] R. Agrawal, A. Gupta, S. Sarawagi, Modeling multidimensional databases, Proc. of the 13th Int. Conf. on Data Engineering, IEEE, 1997, pp. 232–243.
[4] O. Romero, A. Abelló, Multidimensional Design by Examples. In Proc. of 8th Int. Conf. on Data Warehousing and Knowledge Discovery, volume 4081 of LNCS, page 85–94. Springer, 2006.
[5] O. Romero, A. Abelló, Automating Multidimensional Design from Ontologies. In Proc. of ACM 10th Int. Workshop on Data Warehousing and OLAP, page 1–8. ACM, 2007.
[6] O. Romero, A. Abelló, On the Need of a Reference Algebra for OLAP. In Proc. of 9th Int. Conf. on Data Warehousing and Knowledge Discovery, volume 4654 of LNCS, page 99–110. Springer, 2007.
[7] O. Romero, D. Calvanese, A. Abelló, M.Rodríguez-Muro, Discovering Functional Dependencies from Ontologies. In Proc. of ACM 12th Int. Workshop on Data Warehousing and OLAP (to appear), LNCS, page 1–8. Springer, 2009.
[8] O. Romero, A. Abelló, A Survey of Multidimensional Modeling Methodologies, International Journal of Data Warehousing and Mining (IJDWM) 5 (2) (2009) 1–23.
[9] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, S. Paraboschi, Designing data marts for data warehouses, ACM Transactions on Software Engineering and Methodology 10 (4) (2001) 452–483.
[10] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Closest pair queries in spatial databases, Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, ACM, 2000, pp. 189–200.
[11] J. Demetrovics, G.O. Katona, D. Miklós, Functional dependencies distorted by errors, Discrete Applied Mathematics 156 (6) (2008) 862–869.
[12] D. Dori, R. Feldman, A. Sturm, Transforming an operational system model to a data warehouse model: a survey of techniques, IEEE Int. Conf. on Software — Science, Technology and Engineering (SwSTE 2005), IEEE Computer Society, 2005, pp. 47–56.
[13] P. Giorgini, S. Rizzi, M. Garzetti, Goal-oriented requirement analysis for data warehouse design, Proc. of 8th Int. Workshop on Data Warehousing and OLAP, ACM Press, 2005, pp. 47–56.
[14] M. Golfarelli, D. Maio, S. Rizzi, The dimensional fact model: a conceptual model for data warehouses, International Journal of Cooperative Information Systems 7 (2–3) (1998) 215–247.

[15] J.-L. Hainaut, M. Chandelon, C. Tonneau, M. Joris, Contribution to a theory of database reverse engineering, Proc. of the 1st Working Conf. on Reverse Engineering, IEEE, 1993, pp. 161–170.
[16] B. Hüsemann, J. Lechtenbörger, G. Vossen, Conceptual data warehouse modeling, Proc. of 2nd Int. Workshop on Design and Management of Data Warehouses, CEUR-WS.org, 2000, p. 6.
[17] M.R. Jensen, T. Holmgren, T.B. Pedersen, Discovering multidimensional structure in relational data, 6th Int. Conf. on Data Warehousing and Knowledge Discovery, LNCS, vol. 3181, Springer, 2004, pp. 138–148.
[18] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross, The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses, John Wiley & Sons, Inc., 1998
[19] J. Lechtenbörger, G. Vossen, Multidimensional normal forms for data warehouse design, Information Systems 28 (5) (2003) 415–434.
[20] W. Lehner, J. Albrecht, H. Wedekind, Normal forms for multidimensional databases, Proc. of 10th Int. Conf. on Statistical and Scientific Database Management, IEEE, 1998, pp. 63–72.
[21] H. Lenz, A. Shoshani, Summarizability in OLAP and statistical data bases, Proc. of 9th Int. Conf. on Scientific and Statistical Database Management, IEEE, 1997, pp. 132–143.
[22] H. Mannila, K.-J. Räihä, On the complexity of inferring functional dependencies, Discrete Applied Mathematics 40 (2) (1992) 237–243.
[23] J.-N. Mazón, J. Trujillo, An mda approach for the development of data warehouses, Decision Support Systems 45 (1) (2008) 41–58.
[24] J.-N. Mazon, J. Trujillo, J. Lechtenborger, Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms, Data & Knowledge Engineering 23 (3) (2007) 725–751.
[25] Microsoft. MDX Specification. http://msdn.microsoft.com/en-us/library/aa216767.aspx. Last access: 8/08/2008.
[26] D. Moody, M. Kortink, From enterprise models to dimensional models: a methodology for data warehouse and data mart design, Proc. of 2nd Int. Workshop on Design and Management of Data Warehouses, CEUR-WS.org, 2000.
[27] Patrick O'Neil, Elizabeth O'Neil and Xuedong Chen. The Star Schema Benchmark, http://www.cs.umb.edu/~poneil/StarSchemaB.PDF, last access 30-09-2008.
[28] C. Phipps, K.C. Davis, Automating data warehouse conceptual schema design and evaluation, Proc. of 4th Int. Workshop on Design and Management of Data Warehouses, vol. 58, CEUR-WS.org, 2002, pp. 23–32.
[29] N. Prakash, Y. Singh, A. Gosain, Informational scenarios for data warehouse requirements elicitation, 23rd International Conference on Conceptual Modeling, ER 2004, Lecture Notes in Computer Science, vol. 3288, Springer, 2004, pp. 205–216.
[30] N. Prat, J. Akoka, I. Comyn-Wattiau, A UML-based data warehouse design method, Decision Support Systems 42 (3) (2006) 1449–1473.
[31] S. Rizzi, A. Abelló, J. Lechtenbörger, J. Trujillo, Research in data warehouse modeling and design: dead or alive? Proc. of ACM 9th International Workshop on Data Warehousing and OLAP, ACM, 2006, pp. 3–10.
[32] J. Schiefer, B. List, R.M. Bruckner, A holistic approach for managing requirements of data warehouse systems, 8th Americas Conference on Information Systems (AMCIS 2002), 2002, pp. 77–87.
[33] Y. Sismanis, P. Brown, P.J. Haas, B. Reinwald, Gordian: efficient and scalable discovery of composite keys, Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006), ACM, 2006, pp. 691–702.
[34] I.-Y. Song, R. Khare, B. Dai, SAMSTAR: a semi-automated lexical method for generating STAR schemas from an ER diagram, Proc. of the 10th Int Workshop on Data Warehousing and OLAP, ACM, 2007, pp. 9–16.
[35] H.B.K. Tan, Y. Zhao, Automated elicitation of functional dependencies from source codes of database transactions, Information & Software Technology 46 (2) (2004) 109–117.
[36] R. Winter, B. Strauch, A method for demand-driven information requirements analysis in DW projects, Proc. of 36th Annual Hawaii Int. Conf. on System Sciences, IEEE, 2003, pp. 231–239.

Alberto Abelló has an MSc and a PhD in computer science from the Universitat Politècnica de Catalunya (Polytechnical University of Catalonia). He an associate professor at the Facultat d'Informàtica de Barcelona (Computer Science School of Barcelona). He also a member of the GESSI research group (Grup de recerca en Enginyeria del Software per als Sistemes d'Informació) at the same university, specializing in software engineering, databases and information systems. His research interests are database design, data warehousing, OLAP tools, ontologies and reasoning. He is the author of articles and papers presented and published in national and international conferences and journals on these subjects

Oscar Romero has an MSc and a PhD in computer science from the Universitat Politècnica de Catalunya (Polytechnical University of Catalonia). Currently, he is an assistant professor at the Escola Tècnica Superior d'Enginyeria Industrial i Aeronàutica de Terrassa (Industrial and Aeronautical Engineering School of Terrassa). He also a member of the GESSI research group (Grup de recerca en Enginyeria del Software per als Sistemes d'Informació) at the same university, specializing in software engineering, databases and information systems. His research interests are database design, data warehousing, OLAP tools, ontologies and reasoning. He is the author of articles and papers presented and published in national and international and journals conferences and journals on these subjects