Automatic Multidimensional Design of Data Warehouses from Requirements

Oscar Romero and Alberto Abelló
Departament de Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya, Spain
{oromero|aabello}@lsi.upc.edu

Abstract

The ideal scenario to derive the multidimensional conceptual schema of a data warehouse would entail a hybrid approach (i.e. a combined data-driven and requirement-driven approach). Thus, the resulting multidimensional schema would satisfy the end-user requirements and it would have been conciliated with the data sources. Currently, most methodologies follow either a data-driven or requirement-driven paradigm and only a few of them follow a hybrid approach. Furthermore, current hybrid methodologies are unbalanced and they do not benefit from all the advantages brought by each paradigm.

In this paper we present a novel methodology that derives conceptual multidimensional schemas from relational sources bearing in mind the end-user requirements. The most relevant step within our methodology is the MDBE method that introduces three main benefits with regard to previous approaches: (i) the MDBE method is a fully automatic approach and therefore, it also handles requirements in an automatic way. (ii) Unlike data-driven methods, we focus on data of interest for the end-user. However, the user may not know all the potential analysis contained in the data sources and, unlike requirement-driven approaches, MDBE is able to propose new interesting multidimensional knowledge related to concepts already queried by the user. (iii) Finally, MDBE proposes meaningful multidimensional schemas derived from a validation process. Therefore, schemas proposed are sound and meaningful.

1 Introduction

Data warehousing systems were conceived to support decision making within organizations. These systems homogenize and integrate the data of organizations in a huge repository (i.e. the data warehouse) in order to exploit this single and detailed representation of the organization and extract relevant knowledge for the organization’s decision making process.

Nowadays, it is widely accepted that the conceptual schema of a data warehouse must be structured according to the multidimensional model. The multidimensional conceptual view of data is distinguished by the fact / dimension dichotomy, and it is characterized by representing data as if placed in a n-dimensional space, which allows to easily understand and analyze data in terms of facts (the subjects of analysis) and dimensions showing the different points of view where a subject can be analyzed from.

Since a data warehouse is the result of homogenizing and integrating relevant data of the organization in a single and detailed view, it is assumed that the multidimensional conceptual schema of a data warehouse must be derived from the organization’s data source schemas. Traditionally, this process has been performed manually, but automating it is essential to not depend on the expert’s ability to properly apply the methodology chosen, and to avoid the tedious and time-consuming task (even unfeasible when working over large databases) of analyzing the data sources. In the last years, a few approaches have been proposed to automate this process. These approaches mainly follow a data-driven process focusing on a thorough analysis of the data sources to derive the data warehouse schema in a reengineering process, overlooking the end-user multidimensional requirements. However, as discussed in [28], a requirement analysis phase is crucial to meet the user needs and expectations. Otherwise, the user may find himself frustrated since s/he would not
be able to analyze data of his / her interest, entailing the failure of the whole system. In the literature we may find several requirement-driven methods but all of them must be carried out manually. Automating requirement-driven approaches would require to formalize the end-user requirements (i.e. translate them to a language understandable by computers) and current methodologies handle requirements mostly stated in languages (such as natural language) lacking the required degree of formalization.

As already discussed in the literature [16, 28], the ideal scenario to derive the data warehouse conceptual schema would entail a hybrid approach (i.e. a combined data-driven and requirement-driven approach). Therefore, the resulting multidimensional schema would satisfy the end-user requirements and it would have been conciliated with the data sources at once (i.e. capturing the analysis potential depicted in the data sources and being able to be populated with data within the organization). However, current automatable methodologies follow a full data-driven approach whereas current requirement-driven approaches are not automatable since they tend to work with requirements at a high level of abstraction.

In this paper we present a novel methodology that has as its main contribution the MDBE (Multi-dimensional Design By Examples) method: an automatic approach conciliating both 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 carry out both phases at once. Thus, both paradigms benefit from feedback returned by each other and eventually, MDBE is able to derive more valuable information than carrying out both phases sequentially (a detailed list of the MDBE main contributions with regard to previous approaches is presented in section 2).

Our methodology automatically derives multidimensional conceptual schemas from relational sources bearing in mind the end-user requirements. It consists of three steps: requirements elicitation, requirements formalization and the MDBE method (see Fig. 1). First step gathers the end-user information requirements that will be used to guide the whole process. However, requirements are typically expressed in a high abstraction level and if we want to automate their analysis we need to formalize them. In our approach requirements are expressed into SQL queries over the relational data sources (i.e. at a logical level over the data sources). SQL queries provide a well-known structure that will help us to fully automate the MDBE method and they also represent the first step towards the conciliation of requirements and data sources.

End-user information requirements (expressed as SQL queries) and the logical model of the data sources would be the inputs of the MDBE method. As output, MDBE presents a constellation schema [13] derived from the data sources, which allows to retrieve data demanded in the input requirements. In short, along this step, MDBE validates whether each input SQL query represents a valid multidimensional query (i.e. if the query retrieves data able to be analyzed from a multidimensional perspective). We will say so if we are able to derive, at least, one multidimensional schema able to retrieve data demanded in the SQL query (the reader may find further information in section 3.1). The conciliation of these schemas will give rise to the output constellation schema.

To show a practical application of our methodology, consider the following example based on the TPC benchmark H (TPC-H) [1]. TPC-H is a decision support benchmark that introduces a database logical schema (see Fig. 2) as well as a suite of 22 business questions that we will consider as our information requirements such as "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). According to our methodology, first, we need to translate these requirements into SQL queries over the data sources logical schema. The TPC-H benchmark provides the SQL translation for each business query and, for instance, the business query #1 (Q1) is formalized as:

```
SELECT l_returnflag, l_linestatus,
sum(l_quantity) as sum_qty,
sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
avg(l_quantity) as avg_qty,
avg(l_extendedprice) as avg_price,
avg(l_discount) as avg_disc,
count(*) as count_order
FROM lineitem WHERE l_shipdate <= date '1998-12-01' - interval '[DELTA]' day
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

Once it is done for each input requirement, the 22 SQL queries, altogether with the database logical schema would be the inputs of our MDBE method, that will automatically derive a set of multidimensional schemas from the data sources, which meet the end-user requirements (see Fig. 3).

To sum up, we would like to notice three main benefits of our approach: (i) the MDBE method is a fully automatic approach and therefore, it also handles requirements in an automatic way. (ii) Unlike data-driven methods, we focus on data of interest for the end-user. However, the user may not know all the potential analysis contained in the data sources [9, 16, 28] and, unlike requirement-driven approaches, MDBE is able to propose new interesting multidimensional knowledge related to concepts already queried by the user. Finally, (iii) MDBE proposes multidimensional schemas derived from a validation process. Therefore, schemas proposed are sound and meaningful. Indeed, MDBE might be used as a validation tool for multidimensional requirements besides its design purposes.

Along this paper we introduce our methodology in detail. Section 2 discusses the related work we may
find in the literature as well as we stress the MDBE main contributions with regard to them. Section 3 presents our methodology. There, the foundations and internals of our approach are discussed in depth. Section 4 focuses on the the core of our methodology, the MDBE method. Finally, section 5 takes advantage of the MDBE tool to present a practical case based on the TPC-H benchmark, in order to show the MDBE potential.

2 Related Work and Main Contributions

In the literature, some methodologies to derive the conceptual multidimensional schema from the data sources have been presented. However, most of them must be carried out manually (by a step-by-step guide) and just a few of them automate the process. According to Winter et al. [28] they can be mainly classified within a supply-driven or demand-driven framework:

- Supply-driven: These approaches start from a detailed analysis of the data sources to determine the multidimensional concepts in a reengineering process. Many methodologies presented in the literature follow this paradigm. For instance, [10, 12, 18] among others.

- Demand-driven: These approaches focus on determining the user multidimensional requirements (as typically performed in other information systems) to later map them onto data sources as, for instance, [9, 11, 22, 28].

- Hybrid approaches: Some works have also proposed to combine both approaches in order to design the data warehouse from the data sources but bearing in mind the end-user requirements. Some examples of methodologies combining somehow both approaches are [4, 9, 16, 20].

Most of these approaches do not automate the process and just present a set of steps (i.e. guidelines) to be followed by an expert in order to derive the multidimensional conceptual schema. Mainly, these methodologies introduce different patterns or heuristics to discover concepts likely to play a multidimensional
role and therefore, to carry out these approaches manually it is compulsory to have well-documented data sources (for instance, with up-to-date conceptual schemas) at the expert’s disposal. However, in a real organization, the data sources documentation may be incomplete, incorrect or may not even exist [10] and furthermore, it would be rather difficult for a non-expert designer to follow these guidelines. In order to solve these problems, automatable methods [9, 10, 12, 16, 20] work directly over relational database logical schemas (i.e. getting up-to-date data). These methods always rely on a thorough analysis of the relational sources, and they mainly share three limitations:

**User requirements not considered** Despite user requirements are essential to fulfill the expectations of end-users [28], these methods mainly do not consider them. In general, they introduce a set of design patterns to identify which multidimensional role may play each relational concept and the multidimensional schema is eventually derived in a reengineering process from the relational schemas by applying these patterns. Among the automatable approaches, only [9, 16] consider requirements but their demand-driven stages must be performed manually (like in any other demand-driven approach) and they just automate their supply-driven stages.

**Design patterns to identify facts** Identifying facts automatically is a hard task [20], and some of these methods rely on heuristics such as table cardinalities or numerical attributes that may identify false facts or overlook real ones [12, 20]. The rest of approaches demand to identify facts manually [9, 10, 16].

**Dependency on normalization** Design patterns used to identify dimensional data are mainly based on “foreign” (FK) and “candidate key” (CK) constraints. In multidimensional design, it is well-known that facts and dimensions must be related by means of 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 got depends on the degree of normalization of the logical schema, since some FK’s and CK’s are lost if we do not consider a schema up to third normal form.

Moreover, in general, supply-driven approaches risk to waste resources by handling many unneeded information [28]. Since they overlook the multidimensional requirements, they must apply their patterns all over the data sources. Specifically, we present below a detailed discussion about each automatable method presented in the literature:

- **Golfarelli et al.** [10] introduced the first semi-automatable method to derive the multidimensional conceptual schema. However, it demands to manually identify facts. Once it is done, for each fact proposed, a conceptual schema is derived following its many-to-one relationships; i.e. dimensions are identified following FK constraints. Thus, they are not able to cope with denormalized schemas nor requirements.

- **Phipps and Davis** [20] propose a supply-driven method to be validated, a posteriori, by means of a demand-driven stage. In this approach, they propose some potential multidimensional schemas that are validated by end-user requirements expressed in terms of MDX queries [17]. This approach fully automates its supply-driven stage but its demand-driven stage must be performed manually. Moreover, they do not use requirements to guide the process but to filter results got and therefore they are not able to identify new knowledge from requirements but just prune results got in the supply-driven stage. Furthermore, this approach relies on a weak heuristic to identify facts: any relational table containing numeric fields is identified as a potential fact and dimensional data is identified following FK chains from those tables identified as facts. Despite this method mainly relies on FKS constraints to identify dimensional data, to our knowledge, this is the only one partially supporting denormalized input relational schemas as each remaining attribute in a table identified as a fact that is non-numeric and non-key is considered as an interesting analysis dimension for that fact. However, this approach introduces too much noise in the final result and that is a reason to use a demand-driven process to filter results got.

- **Jensen et al.** [12] present a method that analyzes the data sources by means of data mining techniques. 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, in order to point out potential snowflake schemas [13]. To infer this metadata, they access the instances and perform 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 like previous methods discussed. Finally, this approach may present problems in terms of complexity due to the high number of permutations computed when looking for inclusion dependencies since requirements are overlooked and patterns introduced must be computed for all the instances.

- Giorgini et al. [9] present a demand-driven approach to derive the conceptual multidimensional schema. However, the authors argue that their approach may also be used as a hybrid approach. To do so, they propose to gather multidimensional requirements as described in their demand-driven stage and later map them onto the data sources in a conciliation process but the automation degree achieved is rather low. Facts, dimensions and measures identified during the requirements analysis must be manually mapped over the data sources. Once it is done, the aggregation hierarchies are complemented with an automatic algorithm similar to the one presented in [10]. Finally, they propose a refinement step to rearrange the multidimensional schema in order to better fit the user’s needs. The authors propose to use this information to reorder dimensions or try to find new directions of analysis but this process must be performed manually.

- Mazón et al. [16] present a method to conciliate both requirements and data-sources. This method is, to our knowledge, the first one introducing a balanced hybrid approach and, in that sense, it would be closer to our general idea. From a multidimensional conceptual schema derived from the end-user requirements, they apply a set of Query/View/Transformation (QVT) relations to guarantee that the conceptual schema got is sound with the data sources. However, the conceptual schema derived from requirements (i.e. their demand-driven stage) is obtained manually. Then, in a second step, from this schema derived from requirements, the authors propose to automatically derive another one sound with the data sources. In our case, we carry out both phases automatically at once, being able to improve the final result as it is discussed in next subsection. Moreover, unlike our approach, they assume that data sources are normalized up to third normal form.

2.1 MDBE Main Contributions

The MDBE method was conceived to overcome those limitations shared by current automatable methods. To our knowledge, (i) MDBE is the first method automating its demand-driven stage. MDBE demands to formalize end-user requirements into SQL queries and later, each SQL query is validated to infer whether it makes multidimensional sense (see section 3.1 for further information). Main contribution in this aspect is that (ii) MDBE validates the query with regard to the explicit and implicit multidimensional knowledge that it contains. For instance, relationships between concepts depict the potential multidimensional role that each concept may play, and joins stated in the WHERE clause identify relationships (i.e. concept associations) explicitly stated by the user that, in some cases, could not be in the data sources logical schema. Moreover, we also take advantage of the knowledge contained in the data sources as supply-driven approaches do, such as foreign key and candidate key constraints if present. Another contribution in this issue is that (iii) MDBE works at an attribute level (since SQL queries handle attributes) whereas other automatable methods work at a table level. Therefore, we may label attributes as dimensional or factual data and accordingly, tables are identified as dimensional data, factual data or tables containing factual data along with denormalized dimensional data [13]. Consequently, we may identify the role played by each attribute within each relation and split it up into different concepts in the resulting multidimensional schema. The main consequence of these two contributions, is that, to our knowledge, (iv) MDBE is the first method able to cope with denormalized relational schemas and get equivalent results as if the logical schema was up to third normal form.

Furthermore, MDBE also benefits from carrying out their demand-driven and supply-driven stages at once in many aspects. In short, we are able to produce more and better outputs than carrying out both
stages sequentially. For instance, (v) MDBE is able to derive implicit knowledge according to the input query and the data sources. It may happen that some attributes in the query do not play a relevant role in the output produced and therefore, they could be overlooked. However, we analyze the potential alternatives we have, as well as metadata contained in the logical schema, and how these alternatives would affect to the output schema, deriving, in some cases, interesting alternatives overlooked by the user. This contribution is quite relevant since in data warehouse modeling it is assumed that the user may not be able to know all the data sources analysis potential and therefore, s/he may overlook interesting analysis alternatives. However, analyzing the whole data sources may be expensive and produce too much noise in the final result [28]. In this paper we present an intermediate solution, where concepts are analyzed to discover its analysis potential if they are implicitly related to concepts already stated by the user in his / her requirements (see step 6 in section 4.1 for further details). Moreover, (vi) MDBE is able to derive new concepts not stated in the logical schemas. Since we handle requirements automatically, we are able to analyze them in depth, identifying information such as concept specializations or new derived measures.

Finally, we would like to remark that the method presented in this paper is a natural evolution of the one presented in [25]. We have improved our previous work in many aspects: now, MDBE is able to handle denormalized logical schemas and we have improved the conciliation between the demand-driven and supply-driven stages (for instance, see vi). Moreover, we have relaxed some of the theoretical patterns introduced in the preliminar version to cope with practical issues. Finally, this paper also presents a detailed case study and introduces the MDBE tool (the implementation of our method).

3 Our Methodology

Our methodology main objective is to support the data warehouse design process. It consists of three steps: requirements elicitation, requirements formalization and the MDBE method. As shown in Fig. 1, first step starts gathering the end-user information requirements. Data warehousing systems differ in various aspects from conventional operational systems (since they are oriented to support decision making) and they demand specialized requirement elicitation processes [16, 28]. Nevertheless, this issue has been thoroughly studied, and nowadays we may find several methodologies that may be used along this first step (for instance, [9, 16, 21, 27, 28]). Furthermore, notice that we gather information requirements in this step. Information requirements [28] aim to fulfill end-user information necessities, which is the objective of a data warehouse [16]. Unlike in other systems, end-users do not have problems to state their information necessities since they represent data that would be of his / her interest for decision making. Thus, information requirements may be easily stated in end-users own words and close to their reality. For instance, "examine stocks provided by suppliers" or "analyze customer purchases with regard to region, product and time" would be typical information requirements.

Next step in our methodology formalizes the requirements gathered. As discussed previously, we aim to automate the requirements manipulation (i.e. integrate them in a fully-automated method) and therefore, they must be translated into a language understandable by computers. In our approach, end-user requirements are expressed into SQL queries over the relational data sources (i.e. at a logical level over the data sources). This step must be carried out by a database expert (for instance, the database administrator of the organization) able to lower the level of abstraction of the input requirements at a logical level. Each actor involved in these two steps is asked to carry out what s/he does best: end-users express requirements in a high abstraction level whereas a database expert is asked to answer these requirements by means of SQL queries over the relational sources.

As depicted in Fig. 1, next step in our methodology corresponds to the MDBE method, that has two inputs: the end-user information requirements (expressed as SQL queries) and the logical model of the data sources. As output, MDBE presents a multidimensional schema derived from the data sources, which allows to retrieve data demanded in the input requirements. Along this step, MDBE validates whether each input SQL query represents a valid multidimensional query; i.e. if the query retrieves data able to be analyzed from a multidimensional perspective. We may say so if the input SQL query represents a valid set of multidimensional operators over a multidimensional schema (i.e. if the query represents data retrieved
from a multidimensional schema after performing valid data manipulations according to the multidimensional 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 multidimensional operators (see section 3.1 for further information). These constraints may be summarized as follows: data retrieve should be (1) free of data summarizability anomalies, and (2) able to be placed in a multidimensional space. If these constraints are guaranteed then, we may find a set of multidimensional operators which would retrieve that data from the multidimensional schema proposed. Finally, notice that each query (i.e. each multidimensional requirement) gives rise to a potential multidimensional schema. The last step within the MDBE method would embrace to conciliate those results in a minimum set of conceptual schemas meeting all the requirements (i.e. obtaining a constellation of multidimensional schemas).

3.1 Foundations

In this section we present the criteria our work is based on. That is, those used to validate the input SQL query (i.e. the information requirement) as a valid multidimensional requirement. We say a query makes multidimensional sense, if it retrieves data able to be analyzed from a multidimensional point of view. In other words, if data retrieved conform a data cube [13]. With this purpose, we carried out a study [26] to identify which constraints a SQL query must satisfy to make multidimensional sense.

Data manipulation in the multidimensional model should be restricted to the multidimensional operators. Unfortunately, nowadays we do not benefit yet from a standard multidimensional algebra and several multidimensional operators have been introduced in the literature. To overcome this problem, we surveyed all these multidimensional operators and we analyzed how they should be translated into SQL queries in a relational implementation of the data warehouse. This study revealed that multidimensional data manipulation (i.e. multidimensionality) pays attention to two aspects: (i) placement of data in a multidimensional space and (ii) correct summarizability of data. If data retrieved preserves both constraints we will be able to depict it as a data cube (i.e. orthogonal dimensions fully functionally determining the fact) free of summarizability problems. Said in other words, this query would represent the translation to SQL of a set of multidimensional operators.

The following criteria are the basis of our method. These constraints are addressed to identify the multidimensional role played by each relational concept as well as to guarantee that schemas proposed by our method would be able to retrieve (by means of multidimensional operators) data demanded in the requirements:

[C1] Multidimensional modeling: Multidimensionality is based on the fact/dimension dichotomy. Dimensional concepts give rise to the multidimensional space where the fact is placed. By dimensional concepts we refer to any concept likely to be used as a new perspective of analysis. Traditionally, they have been classified as dimensions, levels and descriptors. Thus, we consider a dimension to contain a hierarchy of levels representing different granularities (or levels of detail) to study data, and a level to contain descriptors. On the other hand, a fact contains Cells which contain measures. Like in [18], we consider a fact may contain not just one but several different materialized levels of granularity of data. Therefore, one Cell represents those individual cells of the same granularity that show data regarding 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 dimension of analysis. Finally, one fact and several dimensions to analyze it give rise to a star schema.

[C2] The cube-query template: The standard SQL’92 query template to retrieve a Cell of data from the RDBMS was first presented in [13]:

```
SELECT l1.ID, ..., ln.ID, [ F( c.Measure ) ], ... FROM Cell c, Level l1, ... , Level ln WHERE c.key1=l1.ID AND ... AND c.keyn=ln.ID [ AND l1.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 means of “joins” that represent concept associations. The WHERE clause also contains logical clauses restricting an specific level attribute (i.e. a descriptor) to a constant \( K \) by means of 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 in order to identify the values in the result. Finally, the ORDER BY clause is intended to sort the output of the query.

To our purpose, a SQL query will make multidimensional sense if it fits this pattern and fulfills the following semantic constraints.

[C3] The multidimensional space arrangement constraint: Dimensions arrange the multidimensional space where the fact of study is depicted. Each instance of data is identified (i.e. placed in the multidimensional space) by a point in each of its analysis dimensions. Conceptually, it entails that a fact must be related to each analysis dimension by a to-one relationship. That is, every instance of the fact is related to, at least and at most, one instance of an analysis dimension, and every dimension instance may be related to many instances of the fact.

[C4] The Base integrity constraint: We denote by base a minimal set of dimensions functionally determining a fact. Therefore, it guarantees that two different instances of data cannot be placed in the same point of the multidimensional space. Said in other words, given a point in each of these dimensions they only determine one, and just one, instance of data. Moreover, dimensions giving rise to a base must be orthogonal (i.e. functionally independent) [2]. Otherwise, we would use more dimensions than strictly needed to represent data and it would generate empty meaningless zones in the space. In a relational implementation of the data warehouse, the base concept would be implemented as a primary key for the fact table.

[C5] The summarization integrity constraint: Data summarization performed must be correct, and we warrant this by means of three necessary conditions (intuitively also sufficient) [15]: (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, kind of measure being aggregated and the aggregation function. Compatibility must be satisfied since certain functions are incompatible with some dimensions and kind of measures. For instance, we cannot aggregate Stock over Time dimension by means of sum, as some repeated values would be counted. However, compatibility will not be automatically checked in our method unless additional metadata was provided (for instance, a list of compatibilities could be asked to the user for each measure identified).

3.1.1 Additional Considerations

As previously introduced, these constraints are used to validate the final output. If they are not preserved in a given query, we may end the process and inform the user that the current requirement does not make multidimensional sense. Otherwise, the final result would conform a data cube and therefore we would say that the input query is a valid multidimensional requirement. However, it may happen that [C5] is not preserved and yet, retrieve a valid data cube of interest for the end-user.

Our method is able to identify when disjointness and completeness are not preserved with regard to the data sources logical schema. However, end-users may state new concepts in their requirements that have not been captured in the data sources but still derivable from them. Specifically, (i) if completeness is not preserved, the user may be asking for a concept specialization whereas (ii) not preserving disjointness s/he may be asking for derived measures. In the first case, it is easy to illustrate it with an example: if we do not want to query all the countries but only those related to the shops table. In the second case, it may happen that two values not preserving disjointness give rise to a meaningful derived measure. For instance, if the measure is properly weighted. In general, our method produces results which preserve [C5]. However, along the MDBE method we apply the following rule:

\( R1: \) If we are not able to produce any output by preserving [C5], our method tries to find a result by relaxing this constraint.

Along this paper we will clearly remark those steps where this assumption stands. Hence, steps affected by this assumption will try to guarantee [C5] but if no result is produced then, these steps would be relaunched
3.2 Internals

This section presents how the criteria introduced in section 3.1 are used along our method to validate the input SQL queries and produce meaningful multidimensional schemas.

MDBE uses a graph to store information elicited from the whole process. From here on, we will refer to it as the multidimensional graph. Such graph is composed of nodes, representing tables involved in the query and edges, relating nodes (i.e. tables) joined in the query. Each node contains information about the table attributes involved in the query and their potential multidimensional role. On the other hand, edges keep track of joins in the WHERE clause of the query (i.e. keep track of concept associations). Our aim along our method is to label each graph node and their attributes in such a way that the whole graph preserves the MDBE constraints introduced in section 3.1.

3.2.1 Attribute Labeling:

A given relational attribute of multidimensional interest may play a dimensional or a factual role. If it represents an interesting analysis value it will be labeled as a measure (i.e. factual data) and if it represents an interesting perspective of analysis for the multidimensional 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 as a descriptor (see section 4.1 for further information).

Nevertheless, a given attribute may be labeled both as a dimensional concept and as a measure. Agrawal et al. [3] already proposed in their multidimensional model to handle measures and dimensions uniformly (they presented two multidimensional operators to transform measures into dimensions and viceversa), that were also considered by other multidimensional models a posteriori (see [26]). MDBE allows this multiple labeling of a relational attribute, and the final multidimensional schema generated will consequently contain a measure and an analysis dimension derived from the same attribute.

3.2.2 Node Labeling:

Nodes represent relational tables and according to the kind of attributes they contain they may be labeled as dimensional data or factual data:

- **Dimensional data (L):** If that node contains attributes representing an interesting perspective of analysis for the multidimensional data it will be labeled as a level (i.e. as L).

- **Factual data (CM or C):** If that node contains factual data we label it as a Cell. However, we distinguish between two different kinds of Cells:
  - **Cell With Measures (CM):** These nodes represent Cells that contain measures. According to [C4], these nodes will also contain dimensional concepts giving rise to the multidimensional
space where to place data (i.e. the multidimensional base). As discussed in section 3.1, the base fully functionally determines the factual data and in the relational model, it means that these dimensions must conform a candidate key for that node.

- **Cell (C):** These nodes represent “factless fact tables” [13]. This definition is equivalent to the previous one but this kind of Cell does not contain measures. However, these tables are very useful to describe events and coverage and a lot of interesting questions may be asked from them [13].

To distinguish the factual label a node identified as a Cell must take, we follow the decision diagram depicted in figure 4. There, some questions with regard to the query and the table metadata are posed. These questions are directly derived from constraints introduced in section 3.1. For instance, if the input SQL query performs data grouping in the node to be labeled by means of an aggregate attribute in the SELECT clause we label that node as a CM (i.e. we have explicit knowledge -a measure- identifying that node as factual data). However, if the input query performs data grouping (i.e. it contains a GROUP BY clause) but the node to be labeled does not contain any aggregate attribute in the SELECT clause then, it is labeled as a factless fact (C). Similarly, if no data grouping is performed but we are able to give rise to a multidimensional space (i.e. a table candidate key is selected) this node will be labeled as a Cell: as CM if some other attributes than the key are selected (i.e. if it contains measures) or as C otherwise.

When checking if any measure is selected besides a table key, we do not only consider numerical attributes. Traditionally, numerical attributes have given rise to measures since they are perfectly additive, but as discussed in [13] it may happen that semi-additive or nonadditive values could be of interest for the end-user. Moreover, there are some areas where non-numerical values are additive indeed. As example, in the spatial databases area we may find algorithms to perform aggregation of text values representing geographical information (for instance, see [5]).

Finally, notice the semantics involving each alternative in the decision diagram discussed above. Cells identified without grouping will represent “atomic factual data” [2] (i.e. the lowest granularity of data in the data warehouse) whereas those Cells identified by means of data aggregation will represent “aggregated factual data” (i.e. other data granularities of interest).

### 3.2.3 Coping with Denormalization:

As discussed in section 2, our method is able to cope with denormalized input schemas. Thus, it may happen that a given node may play a factual and dimensional role at the same time. Consequently, we introduce two new labels to identify hybrid nodes containing mixed data (i.e. nodes containing factual and denormalized dimensional data). Notice however that as previously discussed, a factual table does always contain dimensional data conforming the multidimensional base. However, hybrid tables contain denormalized dimensional data within that table:

- **Cell With Measures and Denormalized Dimensional Data (CDM):** This label is equivalent to the CM label (therefore, it contains a dimensional base as well as measures) with additional dimensional data. This additional dimensional data represent other analysis levels and descriptors within the same analysis dimension.

- **Cell with Denormalized Dimensional Data (CD):** Similarly, factless facts with denormalized dimensional data are labeled as CD.

Once we know how to label attributes and nodes, the state diagram of a node labeling is shown in Fig. 5. There, transitions between possible labels are shown. Every node remains unlabeled at its initial state (i.e. at the beginning of the labeling process) and according to the explicit knowledge extracted from the query, we properly update its label. For instance, from the initial state, we may label each node either as CM (if one of its attributes is identified as a measure) or as L (if one of its attributes is identified as a dimensional concept). From the CM state, we keep that label if any other measure is identified or we
may update it to $CDM$ if an attribute playing a dimensional role and not being part of the multidimensional base is identified (i.e. if that attribute represents denormalized dimensional data); and so on.

The reader will notice that some transitions depicted in the state diagram are labeled with the NKD (New Knowledge Discovery) tag. Along MDBE, an state transition may take place due to the explicit knowledge extracted from the query or because of implicit knowledge derived both from the input query and the data sources metadata. The latter case represents a scenario either where the user has not explicitly stated a node role or where we do have an alternative labeling according to the implicit knowledge available. In these cases, we analyze every labeling alternative we have for that nodes. As discussed in section 2.1 and presented in detail in section 4.1, this process is also used to derive new multidimensional knowledge not depicted in the requirements.

3.2.4 Edge Labeling:

Edges relate nodes and they keep track of joins in the WHERE clause of the query. A given edge is labeled according to the multidimensional conceptual relationship it may represent. We consider four potential labels: Cell - Cell, Cell - Level, Level - Cell and Level - Level. For instance, a Cell - Level edge label would mean that this relationship could relate factual data (i.e. a node playing a Cell role) to dimensional data (i.e. a level) and similarly for the rest of labels. The reader should notice that edge labels only depict the conceptual role that each involved node may play within the context of 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 eventually labeled as $CM$, $C$, $CDM$ or $CD$ whereas a node playing a dimensional role can only be labeled as $L$.

At this point it is important to remark that if a node is required to play a dimensional role by an edge label it could only be labeled as $L$ and it cannot be labeled as $CDM$ or $CD$. Despite these two labels represent hybrid nodes (i.e. nodes containing factual and denormalized dimensional data), their semantics are rather different to the $L$ label. A node labeled as $L$ guarantees that we have a key identifying the dimensional data within that table but this is not the case of the other two labels since denormalization introduces data redundancy. When an edge relates a given node $n$ to a hybrid node $h$, we are indeed relating $n$ to the factual data within $h$ since the factual data contains the multidimensional base (i.e. a candidate key of the relation). In fact, hybrid nodes could be represented as factual data (i.e. a node labeled as $CM$ or $C$) related by means of a many-to-one relationship to dimensional data (i.e. a node labeled as $L$). Said in other words, we might normalize them. Therefore, if we split hybrid nodes up, $n$ would be just related to the factual node derived from $h$, and not to the denormalized dimensional data.

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

![Figure 5: A state diagram showing the transition between node labels](image)
Table 1: Summarization of rules used to infer relationships multiplicities

<table>
<thead>
<tr>
<th>Cell</th>
<th>Relationship</th>
<th>Multiplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK1</td>
<td>Attr → Attr</td>
<td>N = M</td>
</tr>
<tr>
<td>CK2</td>
<td>CK → FK + NN</td>
<td>1-o-N</td>
</tr>
<tr>
<td>CK2</td>
<td>CK → FK</td>
<td>1-o-1</td>
</tr>
<tr>
<td>CK2</td>
<td>CK + FK → CK</td>
<td>1-o-1</td>
</tr>
<tr>
<td>CK2</td>
<td>CK → CK</td>
<td>1-o-1</td>
</tr>
<tr>
<td>CK2</td>
<td>CK + FK → CK</td>
<td>1-o-1</td>
</tr>
<tr>
<td>CK2</td>
<td>CK → FK</td>
<td>1-1</td>
</tr>
<tr>
<td>CK2</td>
<td>CK + FK → CK</td>
<td>1-1</td>
</tr>
<tr>
<td>CK2</td>
<td>CK → FK</td>
<td>1-1</td>
</tr>
<tr>
<td>CK2</td>
<td>CK → FK</td>
<td>1-1</td>
</tr>
</tbody>
</table>

Table 2: Valid multidimensional relationships in a relational schema.
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 presents a constellation schema from the data sources which allows to retrieve data demanded in the input requirements. In this scenario, each query is analyzed to derive a multidimensional schema meeting its information requirements. This automatic process is depicted in figure 6, and may be divided into three different stages:

- For each input query, first stage extracts the multidimensional knowledge contained in the query (i.e. the multidimensional role played by each concept in the query as well as the conceptual relationships between concepts), that is properly stored in the multidimensional graph (see section 3.2). Along this stage, the role played by the data sources will be crucial to infer the conceptual relationships between concepts.

- Second stage validates the multidimensional graph deployed in the first stage according to the constraints introduced in section 3.1. Our objective is to check if those concepts and relationships stated in the graph give rise as a whole to a data cube. From the graph building point of view, MDEB first stage aims to derive a multidimensional labeling (i.e. label attributes, nodes and edges) that would be validated along the second stage (i.e. check the whole graph soundness). Therefore, this stage checks if we would be able to retrieve data demanded in the input query from the multidimensional schema represented by the multidimensional graph by means of a set of multidimensional operators (see section 3.1 for further information). If the validation process fails, our method ends since data demanded could not be analyzed from a multidimensional point of view (i.e. we would not be able to retrieve data demanded just by means of multidimensional operators). Otherwise, the resulting multidimensional schema is directly derived form the multidimensional graph.

- Third stage aims to find the most representative results among those got. Our step to discover new multidimensional concepts may introduce new potential results of interest and in this stage we introduce a rule to decide which results present to the end-user.

4.1 First Stage: Concepts Labeling

First stage is aimed to build the multidimensional graph along 6 steps. In this section, we introduce a detailed algorithm in pseudo code (namely the MDEB algorithm) to implement the first stage of MDEB, followed by a brief explanation of each one of its steps. For the sake of readability, comprehension of the algorithm took priority over its performance:

declare MDEB 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(set_of_relations(fitting))) then return notify_fail("Tables relationship not allowed");
      (f) Create an edge(get_join_attributes(set_of_joins)); Label edge to relationships fitting;
      (g) if (unequivalent_knowledge_inferred(relationships fitting)) then propagate knowledge;

6. for each g in New_Knowledge_Discovery(graph) do
   (a) output += validation_process(g);
   return output;

The algorithm starts analyzing each query clause according to [C2]:

**Step 1**: Each table in the FROM clause is represented as a node in the multidimensional graph. As presented in section section 3.2, along the whole process we aim to label every node, attribute and edge depicted in the query.

**Step 2**: This step looks for explicit dimensional data used to arrange the multidimensional space. According to [C2] and [C4] the GROUP BY clause must fully functionally determine data. Thus, fields on it represent interesting points of view where analyze data from. Moreover, fields joined to these attributes in the WHERE clause will also be labeled as dimensional data (since joins represent concept associations stated by the user in the requirements [C2]). Up to now, current methods rely on foreign keys to identify dimensional data and therefore results got 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 made in the data source logical schemas and we are able to identify them from the requirements. For instance, it may happen that the end-user states conceptual relationships not depicted in the data source logical schemas. Consequently, every attribute identified along this step is labeled in the multidimensional graph as an interesting level of analysis. Along these steps, every time an attribute is labeled the label of the node where it belongs to will be properly updated according to figure 5.

**Step 3**: This step looks for explicit factual data. Aggregated attributes in the SELECT clause surely play a measure role. However, if the input query does not contain a GROUP BY clause we are not forced to aggregate measures in the SELECT clause, and this step would not be able to point them out (this kind of Cells as well as those not containing measures will be identified along step 6).
**Step 4:** This step looks for explicit dimensional data used to restrict the multidimensional space. Since a multidimensional Selection [26] (i.e. a comparison between an attribute and a constant value) must be carried out over dimensional data [13], this step labels attributes as *dimensional concepts* looking for comparisons in the WHERE clause and following the same criteria regarding concept associations presented in step 2. Attributes identified in this step are labeled as *descriptors* unless they were also used to arrange the multidimensional space (and therefore, they would have been previously labeled as *levels* in step 2).

**Step 5:** Previous steps are aimed to create and label nodes and their attributes whereas this step creates and labels edges (i.e. concept associations). Conceptual relationships are depicted in a SQL query by means of joins in the WHERE clause. In the multidimensional graph joins are represented as edges and along this step we aim to label them according to the process described in section 3.2.

According to the multiplicity inferred for a conceptual association in the WHERE clause a list of potential edge labels is inferred (see table 2). These alternatives are checked prior to label the edge and a given label is overlooked if it contradicts current knowledge depicted in the graph. For instance, it may happen if a node has already been labeled and the edge label demands to label it in an incompatible way (see section section 3.2).

Once every alternative has been validated we have two potential scenarios: We have been able to label that edge with at least, one alternative, or we do not. In the first case the algorithm goes on and if we have been able to infer unequivocal knowledge for a given edge (i.e. if a unique edge label stands) then this knowledge is propagated to the rest of the graph in cascade. Oppositely, in the second case, the algorithm stops since we have identified a conceptual relationship that does not make multidimensional sense.

After these steps the multidimensional 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 demanded by the user (and nodes where they belong to) will be labeled after step 5. When writing the SQL query of a given requirement we may need to introduce *intermediate* concepts to relate explicit concepts stated by the end-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). Moreover, some nodes already labeled after step 5 may have potential alternatives of interest, which may happen if the structure of the query does not clearly identify *measures* (see step 3) or if we are looking for interesting factless facts. Along this paper, we will refer to intermediate nodes and nodes with interesting alternative labels as *implicit* nodes.

As discussed in section 2.1, we propose an intermediate solution to automatically derive new multidimensional knowledge not considered by the user. In our approach, we focus on the implicit concepts of the query, and we analyze the labeling alternatives we have for them. The objective of this step is to know how these alternatives would affect to the output schema, deriving in some cases, interesting analysis alternatives overlooked by the user.

**Step 6:** This step derives new multidimensional knowledge from unlabeled nodes or, according to the NKD transitions in figure 4, testing alternative labels for nodes already labeled. Each unlabeled node may be considered to play a dimensional role (i.e. to be labeled as $L$) or a factual role (according to figure 4, to be labeled as $C$ or $CM$). On the other hand, nodes with potential alternatives of interest will introduce an alternative label. For each combination with regard to these new labels, an *alternative graph* is created if the labels do not contradict knowledge already depicted in the graph. Later, each one of these graphs will be validated as explained in section 4.2 and only those that make multidimensional sense will be finally taken into consideration. Therefore, it is important to remark that a query could give rise to several valid multidimensional graphs. In that case, MDBE would be able to derive several resulting multidimensional schemas for one query.
In short, this step guarantees that according to the input requirement all the possible multidimensional labelings (each one represented as an alternative multidimensional graph) will be generated. For this reason, it may happen that all the nodes of a given graph would have been labeled as dimensional data. However, this kind of graph is directly disregarded by our method since a multidimensional graph must contain, at least, one Cell [C1].

4.2 Second Stage: The Multidimensional Graph Validation

In this stage we validate each one of the multidimensional graphs generated in the previous stage. This validation process also guarantees the multidimensional normal forms presented in [14] to validate the output multidimensional schema. Again, we introduce a detailed algorithm in pseudo code (the validation_process algorithm) to implement our method, followed by a brief explanation of each one of its steps. The reader will notice that this algorithm is called once for each alternative graph generated along step 6 (see the MDBE algorithm):

```plaintext
declare validation_process

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 can not be used to select data.");
       iii. else ask user for semantical validation;
   (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 can not be used to select data.");
    (b) else ask user for semantical validation; Create context edge(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 Measure.");

12. If exists_two_1_to_N_alternative_branches(Cells path) then return notify_fail("Aggregation problems between Cells.");
```

**Step 7:** The multidimensional graph must be connected to avoid the “Cartesian Product” ([C3]). Furthermore, the multidimensional graph should be composed of valid edges giving rise to a path among Cells (factual data) and connected subgraphs of levels (dimensional data) surrounding it, but these constraints will be properly checked along the next steps.

**Step 8:** This step validates levels subgraphs with regard to Cells placement: According to [C3], two different levels in a subgraph cannot be related to the same Cell (step 8b); to preserve [C5], level - level edges raising aggregation problems on Cells with measures selected must be forbidden (step 8c), and finally, every subgraph must represent a valid dimension hierarchy (i.e. not being used to select data) [C1]. Thus, we must be able to point out two nodes in the subgraph representing the top and bottom levels of the hierarchy, and if there are more than one alternative path between those nodes, they must be semantically equivalent (8a). As discussed in section 3.1.1, this step may eventually relax [C5] (i.e. disjointness) if needed along step 8c.
Step 9: Cells determine multidimensional data and they must be related somehow in the graph giving rise to a single Cell path. Otherwise, they would not retrieve a single cube of data [C2]. For every pair of Cells in the graph, we aim to validate those 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 relationships involved in that correspondence, by a one-to-one context edge between both Cells (i.e. a context edge replaces that subgraph representing the one-to-one correspondence). As depicted in figure 7.1, it means that there are a set of relationships linking, as a whole, a Cell CK, also linked by one-to-one paths to a whole CK of the other Cell. (ii) Otherwise, if both CKs are related by means of one-to-many paths or the first CK matches the second one partially, we replace involved relationships by a one-to-many directed context edge. (iii) From the data sources point of view, many-to-many relationships between Cells should be invalidated since they do not preserve disjointness. Nevertheless, this step may eventually relax disjointness as discussed in section 3.1.1.

Steps 10, 11 and 12: Previous step has validated the correspondences between Cells whereas these steps validate the Cells path (multidimensional data retrieved) as a whole: According to [C4], step 10 validates cycles in the path of Cells to assure they are not used to select data, similar to the levels cycles validation. Once the cycle has been validated, Cells involved are clustered in a context node labeled with the cycle multiplicity, as showed in figure 7.2. Steps 11 and 12, according to [C5], look for potential aggregation problems. First one looks for Cells with measures selected at the left side of a one-to-many context edge or node whereas second one looks for alternative branches with one-to-many context edges or nodes each, raising a forbidden many-to-many relationship between Cells involved (as depicted in figure 7.2). Finally, as in any step involving [C5], this step may eventually relax disjointness as discussed in section 3.1.1.

In the end, the MDBE second stage would have validated each graph to be a data cube and only those guaranteeing every step discussed above will be presented to the user.

4.3 Third Stage: Finding Representative Results

Step 6 of the MDBE first stage may produce several alternative graphs for the same query. Along that step, unlabeled nodes (and those with interesting alternatives according to figure 5) are proved as factual and dimensional data in alternative graphs that are validated in the MDBE second stage and eventually, those graphs that make multidimensional sense will be presented to the user. Consequently, we may produce more than one multidimensional schema for a given query.

However, it may happen that an alternative graph makes multidimensional sense and does not represent a new real interesting perspective of analysis. Specifically, dimensional data could always be considered as an alternative factless fact since, as discussed in section section 3.2, our approach considers factual and dimensional data uniformly. Thus, this step aims to find the representativeness of new alternatives produced by step 6 according to the following rule:

R2: If for a given query we got two sibling graphs that suggest to analyze a given dimensional node also as factual data, we disregard the factual role for that node.

Two sibling graphs only differ on the labeling of a given node. Therefore, they exactly have the same labels except for one node considered to play a factless fact role on one graph and a strict dimensional role.
on the other. For instance, consider the following table that depicts the alternative graphs got after the validation step for a given query:

<table>
<thead>
<tr>
<th>Id</th>
<th>Node A</th>
<th>Node B</th>
<th>Node C</th>
<th>Node D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CM</td>
<td>CD</td>
<td>C</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>CM</td>
<td>L</td>
<td>C</td>
<td>L</td>
</tr>
<tr>
<td>3</td>
<td>CM</td>
<td>L</td>
<td>C</td>
<td>CD</td>
</tr>
</tbody>
</table>

According to the previous definition of sibling graphs, alternative graphs number 1 and 2, and graphs number 2 and 3 are siblings. In this case and according to $R2$, for the first sibling relationship we disregard the first graph and choose graph number 2 as the most representative and for the second pair we disregard graph number 3 and choose graph number 2 again. Eventually, this query will produce just one multidimensional schema.

In short, by means of sibling graphs, MDBE is proposing to also analyze the potential factual data that a dimension may contain. However, in most cases, the end-user would not be interested on this kind of analysis. In our method, only two slightly different situations would be considered interesting: (i) a dimension that would also be proposed as factual data if we have been able to find implicit interesting measures (therefore that node would have not been labeled as a factless fact) or (ii) a factless fact that cannot be identified as dimensional data (i.e. results got do not contain the sibling graph for this labeling).

5 The TPC-H Case Study

Along this section we present a representative case study to show the MDBE potential. The TPC benchmark $H$ [1] is a decision support benchmark that introduces a relational database logical schema and a suite of business oriented queries. This benchmark was conceived to represent a real world information system and therefore, the database schema and queries of the TPC-H benchmark have been chosen to have broad industry-wide relevance. As said, TPC-H is a decision support benchmark, and it is sound to propose a multidimensional schema to analyze this data. In fact, in the literature we may find a Star-Schema benchmark (SSB) [19] derived from the TPC-H benchmark to evaluate database system performance of star schema data warehouse queries.

In short, our objective in this section is twofold: (i) we aim to present results got after applying the MDBE method over a representative case study (the TPC-H benchmark), (ii) and compare results got with the multidimensional schema presented in the SSB Star-Schema benchmark.

5.1 The Benchmark

The TPC-H schema provides a relational database logical schema (see Fig. 2) and 22 business queries. The database schema presented portrays the activity of a wholesale supplier. TPC-H does not represent the activity of any particular business segment, but rather any industry which must manage, sell, or distribute a product worldwide (e.g., car rental, food distribution, parts, suppliers, etc.). Queries presented in the benchmark have been given a realistic context and they were chosen to be representative and answer to real-world questions. These queries are defined by the following components:

- The business question, which illustrates the business context in which the query could be used,
- and the functional query definition, which defines, using the SQL-92 language, the function to be performed by the query.

Therefore, the TPC-H benchmark provides high-level descriptions (i.e. the information requirements the user would provide) and their translation to SQL (that the database administrator should carry out). Consequently, the TPC-H benchmark provides all the inputs needed to launch the MDBE method.
5.2 The MDBE Tool

The TPC-H case study was carried out taking advantage of the MDBE Tool [24]. The MDBE tool is fully automatic and once the user provides the method inputs (i.e. the SQL queries and the data sources logical model), the output is directly provided.

About the implementation, the MDBE tool is a web-application running on Apache Tomcat [8]. The tool core was developed in Java (using the Eclipse IDE [6]) following the Java Struts framework [7], although the SQL module to parse SQL queries and database scripts is an external C# web-service.

The reader may find a detailed view of the MDBE tool in appendix A.

5.3 Application of MDBE on the TPC-H Benchmark

Along this section we present a detailed application of the MDBE method to one of the queries of the TPC-H benchmark. We have chosen the business question #5 (Q5) that gives rise to many labeling alternatives that will help us to present our method in detail. Specifically, this query "lists the revenue volume done through local suppliers" is translated into SQL as:

```sql
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_nationkey = n_nationkey and
  s_nationkey = n_nationkey and n_regionkey = r_regionkey and
  r_name = '[REGION]' and o_orderdate >= '[DATE]' and
  o_orderdate < '[DATE]' + '1' year
GROUP BY n_name
ORDER BY revenue desc;
```

Hence, the SQL query above presented along with the data sources logical schema of TPC-H will be the input of the MDBE method execution discussed in this section. Our objective is to validate this requirement as a valid multidimensional requirement and, from this validation process, to derive a meaningful and sound multidimensional schema (potentially more than one, according to step 6 of section 4.1).

5.3.1 Giving Rise to the Multidimensional Graph

First MDBE stage aims to build the multidimensional graph along 6 steps as follows (a detailed explanation of each step may be found in section 4):

- Step 1: Every table in the FROM clause of the query is represented as a node in the multidimensional graph. Therefore, the graph will have six nodes: `customer`, `orders`, `lineitem`, `supplier`, `nation` and `region`.

- Step 2: Each attribute in the GROUP BY clause is identified as a level (i.e. as a dimensional concept). Thus, attribute `n_name` from node `nation` is labeled as a level and accordingly (see figure 5), `nation` is labeled as a node containing dimensional data (i.e. L). Furthermore, with the purpose of propagating that knowledge, we check any concept association in the WHERE clause where `n_name` is involved. However, there is not any join involving that attribute. If `c_nationkey` would have 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 them (i.e. `c_nationkey = s_nationkey` and `s_nationkey = n_nationkey`).

- Step 3: Those aggregated attributes depicted in the SELECT clause (i.e. `l_extendedprice` and `l_discount`) are identified as measures, and accordingly table `lineitem` is labeled as a Cell with measures (CM).

- Step 4: There are three comparison clauses between attributes and constants in the WHERE clause (r_name = '[REGION]', o_orderdate >= '[DATE]' and o_orderdate < '[DATE]' + '1' year). Selections identify dimensional data and r_name and o_orderdate would be labeled as descriptors.
Figure 8: The multidimensional graph for Q5 after step 5

(i.e. dimensional data). Accordingly, orders and region are labeled as dimensional data ($L$). In this step, we check again joins in the WHERE clause involving any of these attributes to propagate the multidimensional knowledge through concept associations. However, none of them are involved in a join.

- **Step 5**: This step depicts the semantic relationships between concepts in the graph.
  
  (i) For each conceptual relationship in the WHERE clause we infer the relationship multiplicity. In this example, each conceptual relationship is defined by means of a single-attribute join despite they might be depicted by means of multi-attribute joins. For instance, $l\_orderkey = o\_orderkey$ represents a relationship between lineitem and orders. According to table 1 (second row), this join gives rise to 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 demands to label it as dimensional data. Oppositely, the Cell - Cell relationship is sound with current knowledge depicted in the graph. Despite orders has already been labeled as dimensional data in step 4, according to figure 5 it could also be considered as a hybrid node (see the NKD transition) and therefore, it could also be labeled either as CDM or CD. In this case, according to figure 4 it should be labeled as CD (since the query performs data grouping but there is not any orders attribute aggregated in the SELECT clause). Finally, the level - Cell relationship is allowed, so current edge is labeled with both possibilities (level - Cell and Cell - Cell). The reader may find a graphical representation of the multidimensional graph after step 5 in Fig. 8.

- **Step 6**: This step aims to discover new multidimensional knowledge with regard to concepts involved in the query. We have two different ways to do it: studying potential alternatives for each unlabeled node and discovering new labeling alternatives following the NKD edges in figure 5. In our example, we have two unlabeled nodes (i.e. customer and supplier, labeled as '?' in figure 8) and three nodes that, according to figure 5, may play a factual role besides their current dimensional role (i.e. orders, nation and region, marked with a '*' in figure 8). For each combination with regard to these new labels that do not contradict knowledge depicted in the current graph an alternative graph is generated.
Table 3: Graph labelings generated after the first stage of MDBE.

<table>
<thead>
<tr>
<th>Id</th>
<th>Lineitem</th>
<th>Customer</th>
<th>Orders</th>
<th>Supplier</th>
<th>Nation</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CM</td>
<td>C</td>
<td>CD</td>
<td>C</td>
<td>CM</td>
<td>CD</td>
</tr>
<tr>
<td>2</td>
<td>CM</td>
<td>C</td>
<td>CD</td>
<td>C</td>
<td>L</td>
<td>CD</td>
</tr>
<tr>
<td>3</td>
<td>CM</td>
<td>C</td>
<td>CD</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>4</td>
<td>CM</td>
<td>L</td>
<td>CD</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>5</td>
<td>CM</td>
<td>L</td>
<td>CD</td>
<td>C</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>6</td>
<td>CM</td>
<td>L</td>
<td>CD</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>7</td>
<td>CM</td>
<td>L</td>
<td>L</td>
<td>C</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>8</td>
<td>CM</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

After step 6, we have 5 nodes with two potential labeling alternatives that give rise to 8 different multidimensional graphs. Notice that we do not generate 32 graphs (i.e. \(2^5\) combinations) since many of them are meaningless in the multidimensional model. According to step 5 of section 4.1, a given labeling is overlooked if it contradicts knowledge depicted in the graph. For instance, consider the following labeling alternative where 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 be labeled as C as well. Otherwise, it would not make multidimensional sense (see table 2). This kind of contradictions prunes 24 out of 32 possible combinations. The other 8 combinations (shown in table 3) would be validated along the MDBE second stage.

5.3.2 Validating the Multidimensional Graphs Generated

Each one of the 8 graphs generated in the previous stage must be validated in this stage. Along this section we present the highlights about the validation process of these graphs:

- **Step 7:** This step checks if the multidimensional graphs are connected and in this case all of them are.

- **Step 8:** This step validates subgraphs of levels along three substeps. Step 8a checks the semantics of cycles of Levels (if any) for each level subgraph. In our example, none of the graphs contain a cycle within a level subgraph so that all of them satisfy this step. Step 8b checks if two different Levels of the same subgraph are related to the same Cell of data. This is the case of the alternative graph depicted in row 8 of table 3. There, all the nodes except for lineitem are labeled as levels. Thus, 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 disregarded since it does not make multidimensional sense. Eventually, step 8c looks for level-level edges raising aggregation problems on Cells with measures. 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:** This step validates Cells as a whole by means of the context graph. In our example 3 out of the 7 remaining labeling alternatives give rise to incoherent context graphs that do not preserve the multidimensional constraints. For instance, the labeling alternative depicted in the third row of table 3 would give rise to a forbidden many-to-many relationship between customer and supplier in the context graph. Only four alternatives may fulfill 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 give rise to an invalid context graph.

- **Step 10:** This step validates cycles of Cells. In our example, this would be the case if every node in the graph cycle has been considered to play a factual role (rows 1) or all of them except for region (row 2). In both cases, cycles spotted would not make multidimensional sense since they do not preserve disjointness of lineitem (that contains measures) and hence, both alternatives are disregarded.

- **Step 11 and 12:** These steps look for potential aggregation anomalies in Cells with measures. At this moment, we only have two valid alternatives (rows 6 and 7) but none of them give rise to aggregation problems with regard to lineitem.
At the end of the validation process we have two labeling alternatives (see rows 6 and 7 of table 3) out of the 8 initial ones which are sound and meaningful with regard to multidimensionality. Therefore, MDBE would give rise to two different multidimensional schemas that would fulfill the input requirement.

5.4 Discussion

Along this section we have presented an in-depth analysis of one (Q5) of the 22 business queries of the TPC-H benchmark. Table 4 presents some statistics about the process carried out for the other 21 queries (between brackets, statistics for their subqueries, if any). First column represents the query id and the rest of the columns must be read as follows: second column depicts how many implicit nodes do we have for that query (i.e. nodes that remain unlabeled up to step 6 or relabeled there). According to the number of implicit nodes, we may give rise to $2^{\text{implicit nodes}}$ label combinations. 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, not preserving the multidimensional constraints. Combinations not generated are summarized in the third column whereas the forth column shows how many alternative graphs (to be validated) are generated for each query. Fifth column summarizes how many multidimensional graphs are disregarded in the validation stage of MDBE and the sixth column shows how many graphs are collapsed according to the rule introduced in section 4.3.

These six columns show statistics about the MDBE process whereas next three show statistics about the results got for each query: seventh column depicts how many final star schemas have been retrieved by MDBE for that query (i.e. how many alternative graphs have been completely validated). Next column summarizes the number of factless facts identified for this query and the last column depicts the number of new dimensional attributes spotted along the process (if any). These attributes are those identified as dimensional data within a hybrid node (i.e. dimensional attributes within CD and CDM nodes). Finally, symbols † and ‡ depict if [C5] has been relaxed to produce an output result. The first one is used if new derived measures have been proposed by relaxing disjointness and the second one if new concept specializations have been identified by relaxing completeness. In both cases, these results should only be taken into account if these new measures and concept specializations are of interest for the end-user. In this example, according to the specification of the TPC-H benchmark, the Q9 output (the only result produced by relaxing [C5]) should be taken into account. All in all, these results place emphasis on some interesting features of our method:

- Firstly, we would like to place stress on the MDBE validation process. Along this stage, 14 out of 22 queries invalidate some alternative graphs that, at first sight, may seem correct. Thus, labeling nodes is not enough and we must take into account the semantics of the results proposed as a whole.
- Our process to discover new knowledge from implicit nodes is carried out in most of the queries. For the TPC-H case study, the sixth step of our method labels (or relabels) some nodes in 20 out of 22 queries (see the second column), revealing the importance of this step that proposes complementary information to those explicitly stated by the user.
- We would like to remark the importance of handling denormalization. Although the TPC-H logical schema is normalized and well-formed, from a multidimensional point of view it is not normalized and therefore, it means that many nodes along the labeling process have been labeled as CD or CDM. For instance, in one of the solutions proposed for Q5, orders is labeled as a factless fact CD with o_orderdate as a denormalized (from a multidimensional point of view) dimensional attribute. This result is sound since time and date are typical dimensions of analysis in any data warehouse and in fact, some current methodologies always complement their results with these two dimensions (for instance, [20]). Thus, in our final result, these concepts are explicitly stated according to their multidimensional role. In our example, 15 out of 22 queries identify, at least, one new dimensional attribute for that query (see column 9). For instance, shippedate, returnflag and shipmode from lineitem. Moreover, we may also find uniform attributes (see section 3.2.1). For instance, ps_supplycost from
partsupp. Therefore, the resulting conceptual schema contains a dimensional attribute and a measure derived from this relational attribute.

- Our process is able to identify interesting additional information traditionally overlooked in the rest of methodologies. Our method supports factless facts (see column 8) and it is also able to identify new derived measures (see †) and concept specializations (see ‡) not captured in the relational sources but derivable from them.

About the MDBE algorithm complexity, notice that most combinations of labels generated by step 6 are disregarded according to edges semantics and hence, giving rise to a tractable algorithm. In all the queries, the final set of graphs to be validated is considerably smaller than \( 2^{#\text{implicit nodes}} \) (see the fourth column).

Once the MDBE tool has been launched for each query we must carry out a final step to conciliate results got for all the queries into a single conceptual schema. This conciliation process gives rise a to a minimal multidimensional schema (normally, a constellation schema if there is more than one output schema of each query). For example, results got for each of the 22 input queries of the TPC-H benchmark would give rise to the multidimensional schema represented in Fig. 3. Nowadays, this last step must be carried out manually.

Figure 3 only shows measures identified along the process since dimensional concepts have been overlooked to not fuss the final result. Nevertheless, we would like to underline that MDBE works at an attribute level and keeps track of the role assigned to each attribute when deriving partial schemas from each query. Thanks to it, we are able to split some tables up (for instance, orders has given rise to two different concepts in the multidimensional schema since the dimensional attributes contained in the relational orders table have been explicitly represented in orders_dim). Furthermore, the resulting schema only contains those attributes of multidimensional interest (we do not select the whole relational tables with all their attributes) since a multidimensional schema may be considered as a strategic view of the organization data. For instance, in our schema, region would contain r_regionkey and r_name but the r_comment attribute in the TPC-H field would have not been selected since it is not relevant for any of the 22 queries carried out.

### 5.5 Comparison to the Star Schema Benchmark

One of our objectives along this section was to compare our multidimensional schema produced with that presented in the Star-Schema Benchmark. The Star Schema Benchmark (or SSB) [19] presents a multidimensional logical schema manually derived from the TPC-H schema. This schema was devised to improve the querying performance of the data warehouse by denormalization. Data denormalization is quite common

<table>
<thead>
<tr>
<th>Id</th>
<th>Implicit Nodes</th>
<th>Edges Contradicted</th>
<th>Alternative Graphs</th>
<th>Validation Process</th>
<th>Siblings</th>
<th># Results</th>
<th>Factless Facts</th>
<th>New Dim Attr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q2</td>
<td>5(3)</td>
<td>24(4)</td>
<td>9(4)</td>
<td>1(1)</td>
<td>5(2)</td>
<td>1(1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q3</td>
<td>1</td>
<td>2</td>
<td>2(2)</td>
<td>1(1)</td>
<td>0</td>
<td>1(1)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Q4</td>
<td>1(1)</td>
<td>0</td>
<td>2(2)</td>
<td>1(1)</td>
<td>0</td>
<td>1(1)</td>
<td>0</td>
<td>2(3)</td>
</tr>
<tr>
<td>Q5</td>
<td>5(0)</td>
<td>2(2)</td>
<td>9(3)</td>
<td>1(1)</td>
<td>1(1)</td>
<td>9(1)</td>
<td>0</td>
<td>1(1)</td>
</tr>
<tr>
<td>Q6</td>
<td>0</td>
<td>3</td>
<td>2(2)</td>
<td>1(1)</td>
<td>0</td>
<td>1(1)</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Q7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1(1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q8</td>
<td>5(0)</td>
<td>4(4)</td>
<td>9(3)</td>
<td>1(1)</td>
<td>1(1)</td>
<td>1(1)</td>
<td>1(1)</td>
<td>1(1)</td>
</tr>
<tr>
<td>Q9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1(1)</td>
<td>0</td>
<td>1(1)</td>
</tr>
<tr>
<td>Q10</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3(3)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Q11</td>
<td>2(2)</td>
<td>1(1)</td>
<td>3(3)</td>
<td>3</td>
<td>3(3)</td>
<td>0(1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q12</td>
<td>2(2)</td>
<td>1(1)</td>
<td>3(3)</td>
<td>3</td>
<td>3(3)</td>
<td>0(1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q13</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3(3)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Q14</td>
<td>1(2)</td>
<td>0(1)</td>
<td>2(2)</td>
<td>0(1)</td>
<td>1(1)</td>
<td>0(1)</td>
<td>1(1)</td>
<td>0(1)</td>
</tr>
<tr>
<td>Q15</td>
<td>1(2)</td>
<td>0(1)</td>
<td>2(2)</td>
<td>0(1)</td>
<td>1(1)</td>
<td>0(1)</td>
<td>1(1)</td>
<td>0(1)</td>
</tr>
<tr>
<td>Q16</td>
<td>2(1)</td>
<td>1(1)</td>
<td>3(3)</td>
<td>3</td>
<td>3(3)</td>
<td>0(1)</td>
<td>0</td>
<td>1(1)</td>
</tr>
<tr>
<td>Q17</td>
<td>2(1)</td>
<td>1(1)</td>
<td>3(3)</td>
<td>3</td>
<td>3(3)</td>
<td>0(1)</td>
<td>0</td>
<td>1(1)</td>
</tr>
<tr>
<td>Q18</td>
<td>3(3)</td>
<td>3(3)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3(3)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Q19</td>
<td>3(3)</td>
<td>3(3)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3(3)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Q20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0(1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0(1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0(1)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: MDBE statistics over the TPC-H case study.
in data warehouse systems to speed up certain queries [13], and it is achieved by implementing a logical star-schema [13]. Unlike SSB, the MDBE method produces a conceptual schema but we may get the SSB logical schema by means of the same design decision took by the SSB authors: implementing our output schema as a logical star-schema. Star schemas denormalize dimensional data as much as possible to speed up queries. Therefore, we may get the SSB schema as follows:

- Denormalizing dimensions of analysis as much as possible. Therefore, nation and region data will be denormalized within supplier, partsupp and customer. Furthermore, dimensional attributes that gave rise to new concepts (i.e. orders_dim, lineitem_dim and partsupp_dim) must be denormalized as well.

- Merge the three schemas that conform our constellation (since we have three facts of interest) into a single schema. Therefore, lineitem, orders and partsupp will give rise to a single table. This decision is sound with our conceptual schema which relate these three facts (therefore, our conceptual schema allows to “drill-across” [26] between them, getting the same results as merging them down in one single table).

Summing up, the MDBE method is able to derive the same multidimensional schema as SSB.

6 Conclusions and Further Work

In this paper we have presented a novel methodology to support the design process of data warehouses. The MDBE method is a hybrid approach to automatically generate multidimensional schemas from end-user requirements and relational data sources. This method differs from previous approaches in that it joins together the best features of each design paradigm: (i) it considers requirements as first-class citizens within a fully automated approach. (ii) It improves the quality of the final output by improving the communication between its supply-driven and demand-driven stages. In fact, both stages have been merged within MDBE and they depend on each other to produce the output schema. (iii) MDBE also proposes a novel approach to support the user in discovering the analysis potential of the data sources. Moreover, (iv) our method is able to identify new concepts such as specializations or new derived measures from the data sources and finally, (v) we would like to remark that the conceptual schemas produced by MDBE are derived from a validation process and therefore, these schemas are sound and meaningful.

Furthermore, our method has been implemented in a case tool and we have presented the TPC Benchmark H case study to show the potential of our approach as well as to introduce a detailed example of execution of our method.

About our future work, the MDBE method opens new research perspectives. For instance, our approach provides a good basis for the maintainability and evolution of the conceptual schema, which is a topic that has gained relevance in the last years [23]. Finally, we also aim to automate the conciliation process carried out at the end of the method to derive the final constellation schema from the results got for each query.

References


APPENDIX A: The MDBE Tool

As depicted in Fig. 9, the tool main menu has three options: new schema (to upload data sources logical schemas in the tool), modify schema (for maintenance purposes) and new query (to upload SQL queries representing the end-user information requirements).

Using the MDBE tool is quite easy. First, we need to upload the data sources logical schema in the tool. To do so, we must choose the new schema option in the main menu. There, the user is supposed to upload the SQL script (i.e. the logical schema) of the data sources. This script would be checked to see if it is syntactically correct. If it is not correct, an error is prompted and the user will be asked again to introduce a valid SQL script. Otherwise, the schema is stored within the tool and it is presented to the user in a friendly way (the user will be able to modify / delete that schema from the modify schema option of the main menu).

Next, we need to upload the SQL queries one by one by means of the New Query option in the main menu. Every time a query is uploaded the tool asks to choose a schema (among those uploaded in the tool) to validate that query against it. The user may directly upload the query as shown in figure 9 (in this case we are uploading Q5 of the TPC-H benchmark) or use the MDBE wizard developed to assist the user in the query formulation process. Once we have introduced the query and an identifier, the check button checks if the query is syntactically correct. If any problem is found, a message is shown, otherwise, we will launch the MDBE method. If we do so, MDBE presents a multidimensional schema (up to now, in text mode) derived from the TPC-H one, that may retrieve data asked in the information requirement. Figure 10 shows results retrieved by the MDBE tool for query Q5 of the TPC-H benchmark.

Nowadays, the MDBE tool does not support the whole SQL syntax (e.g. some key words such as “case” or “extract”) neither it is able to detect all semantics of the model (e.g. transitivity of foreign keys). However, any of these SQL queries can be rewritten into semantically equivalent queries supported by MDBE (i.e. using the SQL subset supported by our tool) and obtain the same final result. For this reason, some queries of the TPC-H benchmark may need to be manually rewritten into equivalent ones prior to be handled by the MDBE tool.
Figure 10: Results retrieved by the MDBE tool for the TPC-H Q5 query