Automating the Multidimensional Design of Data Warehouses

PhD. Thesis

PhD. Student: Oscar Romero Moral
Advisor: Alberto Abelló Gamazo

Programa de Doctorat en Software
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Barcelona

December 18, 2009

A thesis presented by Oscar Romero Moral
in partial fulfillment of the requirements for the degree of
Doctor en Informàtica per la Universitat Politècnica de Catalunya

To those always there.
This work is, somehow, also yours.

Acknowledgements

"Every thesis begins with a single keystroke."

PhD Comics, 2009

My foremost thank goes to Alberto Abelló. Without him, this dissertation would have not been possible. I thank him for his endless patience and encouragement, for his exigency and rigor, and for his insights and suggestions that helped to shape my research skills. Likewise, I really appreciate that he always had time for me, despite it was not easy with two little children around. Thanks Alberto!

I would also like to thank Ernest Teniente and my colleagues in the *Grup Facing On towards Logic database Rule Enforcement*, and people in the *Secció de Sistemes d'Informació*, for their support.

Thanks to Dr. Diego Calvanese, for the opportunity to stay in his group for five months, and people in the *KRDB Centre for Knowledge and Data*, who made me feel like at home.

I also thank Joan Marc Montesó, for implementing the AMDO tool, and Antonio Montero, for encapsulating the MDBE implementation in a web service. Also thanks to Josep Berbegal, Anna Queralt, Leonor Frías, Gemma Grau and Jordi Conesa for their help on various details of this thesis.

I thank my thesis committee members: Dr. Antoni Olivé, Dr. Ernest Teniente, Dr. Diego Calvanese, Dr. Alkis Simitsis, and Dr. Juan Trujillo. It is a pleasure for me that they accepted to be part of this committee.

I am greatly indebted to my friends, for all the good moments they provided but, specially, for listening to me when I needed to talk, and for making me laugh when I most needed it... such good memories I cannot capture in words!

Last but not least, I owe my deepest gratitude to my mother, who always encouraged me to work hard and always believed in me, and also to my father and brothers, for always being there when I needed them most, and for supporting me through all these years.

This work has been partly supported by the Ministerio de Educación y Ciencia and FEDER, under projects TIN 2005-05406 and TIN2008-03863.

Abstract

Previous experiences in the data warehouse field have shown that the data warehouse multidimensional conceptual schema must be derived from a hybrid approach: i.e., by considering both the end-user requirements and the data sources, as first-class citizens. Like in any other system, requirements guarantee that the system devised meets the end-user necessities. In addition, since the data warehouse design task is a reengineering process, it must consider the underlying data sources of the organization: (i) to guarantee that the data warehouse must be populated from data available within the organization, and (ii) to allow the end-user discover unknown additional analysis capabilities.

Currently, several methods for supporting the data warehouse modeling task have been provided. However, they suffer from some significant drawbacks. In short, requirement-driven approaches assume that requirements are exhaustive (and therefore, do not consider the data sources to contain alternative interesting evidences of analysis), whereas data-driven approaches (i.e., those leading the design task from a thorough analysis of the data sources) rely on discovering as much multidimensional knowledge as possible from the data sources. As a consequence, data-driven approaches generate too many results, which misleads the user. Furthermore, the design task automation is essential in this scenario, as it removes the dependency on an expert's ability to properly apply the method chosen, and the need to analyze the data sources, which is a tedious and time-consuming task (which can be unfeasible when working with large databases). In this sense, current automatable methods follow a data-driven approach, whereas current requirement-driven approaches overlook the process automation, since they tend to work with requirements at a high level of abstraction. Indeed, this scenario is repeated regarding data-driven and requirement-driven stages within current hybrid approaches, which suffer from the same drawbacks than pure data-driven or requirement-driven approaches.

In this thesis we introduce two different approaches for automating the multidimensional design of the data warehouse: MDBE (*Multidimensional Design Based on Examples*) and AMDO (*Automating the Multidimensional Design from Ontologies*). Both approaches were devised to overcome the current limitations previously discussed. On the one hand, we rely on the end-user requirements, but we do not decline that the data sources may also contain hidden analysis capabilities that, eventually, may be of interest. Nevertheless, in any case, we do not generate endless chunks of results from the sources. On the contrary, we aim at filtering by means of objective evidences the results obtained by analyzing the sources. Importantly, our approaches consider opposite initial assumptions, but both consider the end-user requirements and the data sources as first-class citizens. Furthermore, we also focus on the automation of the process, to facilitate the

designer task as much as possible.

- MDBE follows a classical approach, in which the end-user requirements are well-known beforehand. This approach benefits from the knowledge captured in the data sources, but guides the design task according to requirements and consequently, it is able to work and handle semantically poorer data sources. In other words, providing high-quality end-user requirements, we can guide the process from the knowledge they contain, and overcome the fact of disposing of bad quality (from a semantical point of view) data sources.

- AMDO, as counterpart, assumes a scenario in which the data sources available are semantically richer. Thus, the approach proposed is guided by a thorough analysis of the data sources, which is properly adapted to shape the output result according to the end-user requirements. In this context, disposing of high-quality data sources, we can overcome the fact of lacking of expressive end-user requirements.

Importantly, our methods establish a combined and comprehensive framework that can be used to decide, according to the inputs provided in each scenario, which is the best approach to follow. For example, we cannot follow the same approach in a scenario in which the end-user requirements are clear and well-known, and in a scenario in which the end-user requirements are not evident or cannot be easily elicited (e.g., this may happen when the users are not aware of the analysis capabilities of their own sources). Interestingly, the need to dispose of requirements beforehand is smoothed by the fact of having semantically rich data sources. In lack of that, requirements gain relevance to extract the multidimensional knowledge from the sources. So that, we claim to provide two approaches whose combination turns up to be exhaustive with regard to the scenarios discussed in the literature.

Key Words: Data Warehouse, Data Warehousing, OLAP, Multidimensional Design, Automatic Reasoning, Description Logics

Contents

List of Figures

List of Tables

Chapter 1

Introduction

" 'Input! Input!', Need input!' "

Number 5, "Short Circuit", 1986

Nowadays, the *free market economy* is the basis of *capitalism* (the current global economic system) in which the production and distribution of goods are decided by market businesses and consumers; giving rise to the *supply and demand* concept. In this scenario, being more competitive than the rest of organizations becomes essential, and *decision making* raises as a key factor for the organization success.

Decision making is based on *information*. The more accurate information I get, the better decisions I can make to get competitive advantages. That is the main reason why information (understood as the result of processing, manipulating and organizing data in a way that adds new knowledge to the person or organization receiving it) has become a key piece in any organization. In the past, managers' ability for foreseeing upcoming trends was crucial, but this largely subjective scenario changed when the world *became digital*. Actually, any event can be recorded and stored for later analysis, which provides new and objective business perspectives to help managers in the decision making process. Hence, (digital) information is a valuable asset to organizations, and it has given rise to many well-known concepts such as *Information Society*, *Information Technologies* and *Information Systems* among others.

For this reason, today, decision making is a research hot topic. In the literature, those applications and technologies for gathering, providing access to, and analyzing data for the purpose of helping organization managers make better business decisions are globally known as *Business Intelligence*. This term implies having a comprehensive knowledge of any of the factors that affect an organization business with one main objective: the better decisions you make, the more competitive you are.

Under this concept we embrace many different disciplines such as *Marketing*, *Geographic Information Systems* (GIS), *Knowledge Discovery* or *Data Warehousing*.

1.1 Data Warehousing Systems

Data warehousing systems are aimed at exploiting the organization data, previously integrated in a huge repository of data (the *data warehouse*), to extract relevant knowledge of the organization.

A formal definition can be found in [GR09]: *Data Warehousing is a collection of methods, techniques and tools used to support* knowledge workers *-senior managers, directors, managers and analysts- to conduct data analyses that help with performing decision-making processes and improving information resources*. This definition gives a clear idea of these systems final aim: give support to decision making without regard of technical questions like data heterogeneity or data sources implementation. This is a key factor in data warehousing. Nowadays, any event can be recorded within organizations. However, the way each event is stored differs in every organization, and it depends on several factors such as relevant attributes for the organization (i.e., their daily needs), technology used (i.e., implementation), analysis task performed (i.e., data relevant for decision making), etc. Thus, these systems must gather and assemble all (relevant) business data available from various (and possibly heterogeneous) sources in order to gain a single and detailed view of the organization that later will be properly managed and exploited to give support to decision making.

The role of data warehousing can be better understood with five claims introduced by Kimball [KRTR98]:

- *We have heaps of data, but we cannot access it.* Loads of data are available. However, we need the appropriate tools to effectively exploit (in the sense of query and analyze) it.

- *How can people playing the same role achieve substantially different results?* Any organization may have several databases available (devoted to specific business areas) but they are not conceptually integrated. Providing a single and detailed view of the business process is a must.

- *We want to select, group and manipulate data in every possible way.* This claim underlines the relevance of providing powerful and flexible analysis methods to be carried out in real time.

- *Show me just what matters.* Too much information may be, indeed, too much. The end-user must be able to focus on relevant information for his / her current decision making processes.

- *Everyone knows that some data is wrong.* A sensitive amount of transactional data is not correct and it has to be properly cleaned (transformed, erased, filtered, etc.) in order to avoid misleading results.

Data warehousing systems have three main components: the data warehouse, the ETL *(Extraction, Transformation and Load*) tools and the exploitation tools. The data warehouse is a huge repository of data; i.e., a database. It is the data warehousing system core and that is why these systems are also called *data warehouse systems*. However, it is not just another traditional database: it depicts a single and detailed view of the organization business. By means of the ETL

tools data from a variety of sources is loaded (i.e., homogenized, cleaned and filtered) into the data warehouse. Once loaded, it is ready to be exploited by means of the exploitation tools.

The reader is addressed to [GR09] for further details on data warehousing systems. In next sections we will focus on the data warehouse and the exploitation tools, as they are tightly related to this thesis.

1.1.1 The Data Warehouse

The data warehouse term was coined by B. Inmon in 1992 [Inm92] that he defined as: "*a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process*". Where *subject oriented* means that data stored gives information about a particular subject instead of the daily operations of an organization; *integrated* means that data have been gathered into the data warehouse from a variety of sources and merged into a coherent whole; *time-variant* means that all data in the data warehouse is identified with a particular time period and finally, *non-volatile* means that data is stable in the data warehouse. Thus, more data is added but data is never removed. This enables management to gain a consistent picture of the business.

Despite this definition was introduced almost 20 years ago, it still remains reasonably accurate. However, a single-subject data warehouse is currently referred to as a *data mart* (i.e., a local or departmental data warehouse), while data warehouses are more global, giving a general enterprise view. In the literature, we can find other definitions like the one presented in [KRTR98], where a data warehouse is defined as "*a copy of transaction data specifically structured for query and analysis*"; this definition, despite being simpler, is not less compelling, since it underlines the relevance of querying in a data warehousing system. The data warehouse design is focused on improving queries performance instead of improving update statements (i.e., insert, update and delete) like transactional databases do. Moreover, the data warehousing system end-users are high-ranked people involved in decision making rather than those low/medium-ranked people maintaining and developing the organization information systems. Next table summarizes main differences between an operational database and a data warehouse:

Criterion	**Operational DB**	**Data Warehouse**
Objective	Operational (daily operations)	Analysis and decision making
Process	Transactional, repetitive and well-known	Massive querying, specific and not-known
Main Activity	Update statements	Querying
Performance	Relevance of transactions time response	Relevance of the massive querying time response
Users	Medium/Low profile	High profile
Data Model	Relational	Multidimensional

1.1.2 Exploitation Tools

The final aim of every data warehousing system is to exploit the data warehouse. The data warehouse is a huge repository of data that does not tell much by itself; like in the operational

databases field, we need auxiliary tools to query and analyze data stored. In this field, those tools aimed at extracting relevant information from the repository of data are known as the exploitation tools. Without the appropriate exploitation tools, we will not be able to extract valuable knowledge of the organization from the data warehouse, and the whole system will fail in its aim of providing information for giving support to decision making. Most used exploitation tools can be classified in three categories:

- Query & Reporting: This category embraces the evolution and optimization of the traditional query & reporting techniques. This concept refers to an exploitation technique consisting of querying data and generating detailed pre-defined reports to be interpreted by the end-user.
- Data Mining: *Data mining is the exploration and analysis of large quantities of data in order to discover meaningful patterns and rules* [BL04]. The data mining field is a research area per se, but as the reader may note, this kind of techniques and tools suit perfectly to the final goal of the data warehousing systems.
- OLAP Tools: OLAP stands for *On-Line Analytical Processing*, which was accurately chosen to confront the OLTP acronym (*On-Line Transactional Processing*). Its main objective is to analyze business data from its dimensional or components perspective; unlike traditional operational systems such as OLTP systems.

In this thesis we will focus on OLAP tools, but the reader is addressed to [GR09] for further details on accessing the data warehouse.

1.2 OLAP Tools

OLAP tools are intended to ease information analysis and navigation all through the data warehouse, for extracting relevant knowledge of the organization. This term was first introduced by E.F. Codd in 1993 [CCS93], but it was more precisely defined by means of the FASMI (*Fast Analysis of Shared Multidimensional Information*) test [Pen08]. According to it, an OLAP tool must provide *Fast* query answering to not frustrate the end-user reasoning; offer *Analysis* tools, implement security and concurrent mechanisms to *Share* the business *Information* from a *Multidimensional* point of view. This last feature is the most important one since OLAP tools are conceived to exploit the data warehouse for analysis tasks based on *multidimensionality*.

1.2.1 Multidimensionality

Multidimensionality, as it is known today, was first introduced by Kimball in [Kim96], where the author argued about the necessity of an ad hoc modeling technique for data warehouses. Multidimensional modeling optimizes the system query performance in contrast to conventional *Entity-Relationship* (ER) models [Che76] (widely used for modeling relational databases) that are constituted to remove redundancy in the data model and optimize OLTP performance (see discussion in section 1.1.1 for further details on data warehouses vs. OLTP databases).

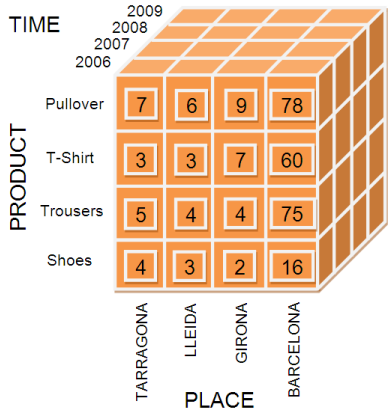


Figure 1.1: Multidimensional view of data

Specifically, the multidimensional conceptual view of data is distinguished by the *fact / dimension* dichotomy, and it is characterized by representing data as if placed in an n-dimensional space, allowing us to easily understand and analyze data in terms of *facts* (the subjects of analysis) and *dimensions* showing the different points of view from where a subject can be analyzed. For instance, Fig. 1.1 depicts `sales` (subject of analysis) of an organization from three different dimensions or perspectives of view (`time`, `product` and `place`). One fact and several dimensions to analyze it give rise to what is known as the *data cube*[1]

This paradigm provides a friendly, easy-to-understand and intuitive visualization of data for non-expert end-users. Importantly, most events recorded are likely to be analyzed from a multidimensional point of view. An event (a potential fact) is recorded altogether with a set of relevant attributes (potential analysis dimension). For example, consider a `sales` event. We may record the `shop`, `city` and `country` where it was purchased, the `item`, `color`, `size` and `add-ons` selected, the `time` (`hour`, `minute`, `second` and even `millisecond`) and `date` (`day`, `month`, `year`), `payment method`, `price`, `discount applied`, the `customer`, etc. Interestingly, every attribute opens a new perspective of analysis for the `sales` event.

More precisely, OLAP functionality is characterized by dynamic multidimensional analysis of consolidated data supporting end-user analytical and navigational activities. Thus, OLAP users are able to navigate (i.e., query and analyze) data in real-time. The user provides a *navigation path* in which each node (resulting in a data cube) is derived from the previous node in the path (and thus we say that the user *navigates* the data). Each node is transformed into the next one in the path by applying specific multidimensional operators. Most popular multidimensional

[1]The data cube refers to the placement of factual data in the multidimensional space. And thus, it can be thought as a mathematical function. Nevertheless, nowadays it is rather common also refer to the multidimensional space as the data cube. However, note that, in both cases, it is a language abuse, since the multidimensional space (or the placement of data in the multidimensional space) only gives rise to a cube if three analysis dimensions are considered. We address the reader to [ASS06] for further details.

operators are "roll-up" (increase the aggregation level), "drill-down" (decrease the aggregation level), "screening and scoping" (select by means of a criterion evaluated against the data of a dimension), "slicing" (specify a single value for one or more members of a dimension) and "pivot" (reorient the multidimensional view). Some works, like [PJ01] and [ASS06], add "drill-across" (combine data from cubes sharing one or more dimensions) to these basic operations.

As a result, multidimensionality enables analysts, managers, executives and in general those people involved in decision making, to gain insight into data through fast queries and analytical tasks, allowing them to make better decisions.

1.3 Multidimensional Design

Developing a data warehousing system is never an easy job, and raises up some interesting challenges. One of these challenges focus on modeling multidimensionality. OLAP tools are conceived to exploit the data warehouse for analysis tasks based on the multidimensional paradigm and therefore, the data warehouse must be structured according to the multidimensional model. Note that we are talking about the multidimensional model and not just about a paradigm. Hence, it must properly define a *data structure*, a *set of operations to handle data* and a *set of integrity constraints*. Unfortunately, we still lack of a standard multidimensional model like the relational model is for operational databases. Nevertheless, lots of efforts have been devoted to multidimensional modeling, and several models and design methods have been developed and presented in the literature. Consequently, we can nowadays design a multidimensional conceptual schema, create it physically and later, exploit it through the model algebra or calculus (implemented in the exploitation tools).

1.3.1 Logical Design: ROLAP vs. MOLAP

When implementing our conceptual schema, and in general the OLAP tool, there are two main trends: using the relational technology or an ad hoc one, giving rise, respectively, to what are known as ROLAP (*Relational On-line Analytical Processing*) and MOLAP (*Multidimensional On-line Analytical Processing*) architectures. ROLAP maps the multidimensional model over the relational one (a multidimensional middleware on the top of the relational database makes this fact transparent for the users), allowing them to take advantage of a well-known and established technology. As consequence, ROLAP tools deal with larger volumes of data than MOLAP tools (i.e., ad hoc multidimensional solutions), but their performance for query answering and cube browsing is not as good. Thus, new HOLAP (*Hybrid On-line Analytical Processing*) tools were proposed. HOLAP architecture combines both ROLAP and MOLAP ones trying to obtain the strengths of both approaches, and they usually allow to change from ROLAP to MOLAP and viceversa.

Although ROLAP tools have failed to dominate the OLAP market due to its severe limitations (mainly slow query answering) [Pen05], at the beginning, they were the reference architecture. Indeed, Kimball's reference book [KRTR98] presents how a data warehouse should be implemented over a RDBMS (*Relational Database Management System*) and how to retrieve data from it. To do so, he introduced two logical (i.e., relational) patterns: the *star schema* and the

snowflake schema. The star schema consists of one table for the fact and one denormalized table for every dimension, with the latter being pointed by *foreign keys* (FK) from the *fact table*, which compose its *primary key* (PK). The normalized version of a star schema is a snowflake schema; getting a table for each level with a FK pointing to each of its parents in the dimension hierarchy. Nevertheless, both approaches can be conceptually generalized into a more generic one consisting in partially normalizing the dimension tables according to our needs: completely normalizing each dimension we get a snowflake schema, and not normalizing them at all results in a star schema.

Currently, pure ROLAP tools have lost their reference position, but the star schema had, and yet has, great impact on multidimensional conceptual modeling, as we discuss in next section.

1.4 Multidimensional Modeling Methods

As discussed in section 1.2.1, multidimensional modeling was first introduced by Kimball in [Kim96]. Kimball's approach was well received by the industry and a deeper and advanced view of multidimensional modeling was presented in [KRTR98]. In these books Kimball also introduced the first method to derive the data warehouse *logical* schema. Similar to traditional information systems modeling, Kimball's method is *requirement-driven*: it starts eliciting business requirements of an organization and through a step-by-step guide we are able to derive the multidimensional schema. Only at the end of the process data sources are considered to map data from sources to target.

In short, Kimball's approach follows a traditional modeling approach (i.e., from requirements), but it set down the principles of multidimensional modeling. Multidimensional modeling is radically opposite to OLTP systems modeling: the data warehouse conceptual schema is directly derived from the organization operational sources and provides a single, detailed, integrated and homogenized view of the business domain. Consequently, the data warehouse can be thought as a strategic view of the organization data and for this reason, and unlike most information systems that are designed from *scratch*, the organization data sources must be considered as first-class citizens in the data warehouse design process. This major additional requirement has such interesting consequences so much so that it gave rise to a new research topic and up to now, several multidimensional modeling methods have been introduced in the literature. With the perspective of time, we may now highlight those features that drew the attention of the community. The evolution of the modeling methods introduced in the literature focuses on two main aspects: (i) the dichotomy requirements versus data sources (and how to deal with it) and (ii) the level of abstraction of the method's output.

1.4.1 A Piece of History

In this section we introduce the background of multidimensional modeling. Our objective here is to provide an insightful view of how this area evolved with time. Note, however, that this dissertation comes up from the discussion of the state of the art carried out in Section 2.1.

Shortly after Kimball introduced his ad hoc modeling method for data warehouses, some other methods were presented in the literature. Like Kimball's method, these methods are step-

by-step guides to be followed by a data warehouse expert that start gathering the end-user requirements. However, unlike Kimball's work, they give more and more relevance to the data sources. Involving the data sources in these approaches means that it is compulsory to have well-documented data sources (for instance, with up-to-date conceptual schemas) at the expert's disposal but it also entailed two main benefits: on the one hand, the user may not know all the potential analysis contained in the data sources and analyzing them we may find unexpected potential analysis of interest for the user; on the other hand, we should assure that the data warehouse can be populated with data available within the organization.

As said, to carry out these approaches manually it is compulsory to have well-documented data sources, but in a real organization, the data sources documentation may be incomplete, incorrect or may not even exist and, in any case, it would be rather difficult for a non-expert designer to follow these guidelines. Indeed, when automating this process is essential not to depend on the expert's ability to properly apply the method chosen and to avoid the tedious and time-consuming task (even unfeasible when working over large databases) of analyzing the data sources.

In order to solve these problems, several new methods automating the design task were introduced in the literature. These approaches work directly over relational database logical schemas. Thus, despite they are restricted to relational data sources, they get up-to-date data that can be queried and managed by computers. They also argue that restricting to relational technology makes sense since nowadays it is the most widely used technology for operational databases. About the process carried out, these methods follow a *data-driven* process focusing on a thorough analysis of the data sources to derive the data warehouse schema in a reengineering process. This process consists of techniques and design patterns that must be applied over the data sources schema to identify data likely to be analyzed from a multidimensional perspective.

Nevertheless, a requirement analysis phase is crucial to meet the user needs and expectations. Otherwise, the user may find himself frustrated since he / she would not be able to analyze data of his / her interest, entailing the failure of the whole system. Today, it is assumed that the ideal scenario to derive the data warehouse conceptual schema embraces a hybrid approach (i.e., a combined data-driven and requirement-driven approach). Then, the resulting multidimensional schema will satisfy the end-user requirements and it will have been conciliated with the data sources simultaneously. However, these later approaches overlook the automatization of the process as automating their requirement-driven stages requires to formalize the end-user requirements (i.e., translate them to a language understandable by computers). Unfortunately, current methods handle requirements mostly stated in languages (such as natural language) lacking the required degree of formalization.

Another interesting trend worth to remark, is the level of abstraction used for the methods output. The first methods introduced (such as Kimball's method) produced multidimensional star schemas (i.e., logical schemas), but soon the community realized it was as important as in any other system to differentiate the conceptual and logical layer in the data warehouse design task (for example, it was obvious when MOLAP tools gained relevance regarding ROLAP ones). As result, newest approaches generate conceptual schemas and it is up to the user to implement them with any of the logical design (either relational or ad hoc multidimensional) alternatives. The fact that logical design was first addressed in data warehousing gave rise to a spread language abuse when referring to multidimensional conceptual schemas, which are also denoted as *star schemas*

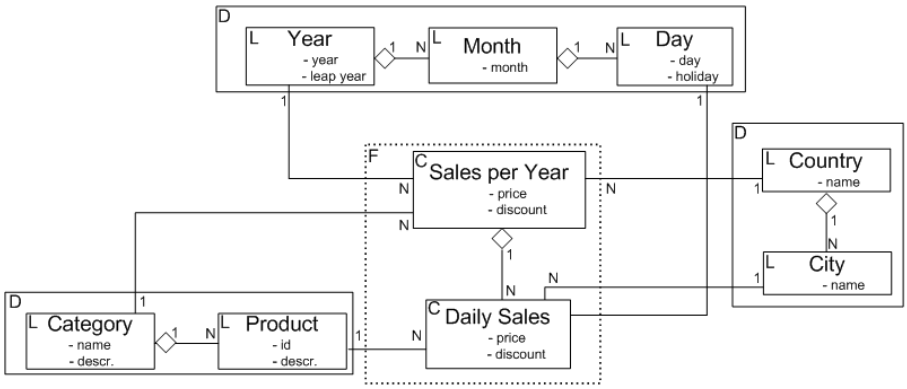


Figure 1.2: The multidimensional concepts

(originally, star schemas were logical design models introduced by Kimball, see section 1.3.1 for further details). The reason is that multidimensional conceptual schemas also are *star-shaped* (with the fact in the center and the dimensions around it), and the star schema nomenclature was reused for conceptual design.

1.5 Terminology and Notation

Lots of efforts have been devoted to multidimensional modeling, and several models and approaches have been developed and presented in the literature to support the design of the data warehouse. However, since we lack a standard multidimensional terminology, terms used among different methods may vary. To avoid misunderstandings, in this section we detail a specific terminology (based on **YAM**^{2} [ASS06]) to be used in this document. Relevant concepts will be **bolded** for the sake of understandability:

Multidimensionality is based on the fact / dimension dichotomy. **Dimensional concepts** produce the multidimensional space in which the **fact** is placed. **Dimensional concepts** are those concepts likely to be used as a new analytical perspective, which have traditionally been classified as **dimensions**, **levels** and **descriptors**. Thus, we consider that a **dimension** consists of a hierarchy of **levels** representing different granularities (or levels of detail) for studying data, and a **level** containing **descriptors** (i.e., **level** attributes). We denote by **atomic level** the **level** at the bottom of the **dimension** hierarchy (i.e., that of the finest level of detail) and by **All level** the **level** at the top of the hierarchy containing just one instance representing the whole set of instances in the **dimension**. In contrast, a **fact** contains **Cells** which, in turn, contain **measures**. As in [MK00], we consider that a **fact** may contain not just one but several different levels of data granularity. Therefore, one **Cell** represents individual cells of the same granularity that contain data corresponding to the same **fact** (i.e., a **Cell** is a "Class" and cells are its instances). Specifically, a **Cell** of data is related to one **level** for each of its associated **dimensions** of analysis.

Finally, one **fact** and several **dimensions** for its analysis produce a star schema.

For example, consider Figure 1.2. There, one **fact** (`sales`) containing two **measures** (`price` and `discount`) is depicted. This fact has three different **dimensions** of analysis (`place`, `time` and `product` sold). Each **dimension** has its own aggregation hierarchy. For instance, the `time` **dimension** has three **levels** of detail (i.e., `year`, `month` and `day`) that, in turn, contain some **descriptors** (for example, the `holiday` or `leap year` attributes). Furthermore, the `sales` **fact** contains several **Cells**. Each **level** of data granularity available for a given **fact** gives rise to a **Cell** within that **fact**. Thus, we may analyze this **fact** from a wide range of data granularities: from its finest data granularity (i.e, {`product` × `day` × `city`}, which is the one shown in the figure as `daily sales`) up to its coarsest data granularity[2] (i.e., {`category` × `year` × `country`}, which is shown as `sales per year`). Any of the factual instances within `daily sales` or `sales per year` is called a cell. Finally, note that we consider {`product` × `day` × `city`} to be the multidimensional **base** of the **Cell** with the finest granularity level (i.e., that related to the **atomic levels** of each **dimension**, which is also known as the **atomic Cell**) and {`category` × `year` × `country`} to be the **base** of the **Cell** with the coarsest data granularity. Thus it means that one value of each one of these **levels** determines one cell (i.e., a `sale` with its `price` and `discount`).

1.6 Motivation

In this section we discuss the advantages and disadvantages of current multidimensional design methods. Specifically, we will motivate our work focusing on the weaknesses of current approaches. To do so we carried out a detailed analysis of the current state of the art (see chapter 2 for further details). In this section we present the conclusions of this work. As result, we were able to identify the major drawbacks still not addressed in the literature. For presenting our results we take advantage of the method classification introduced by Winter & Strauch [WS03]. According to it, multidimensional modeling methods may be classified within a *demand-driven*, a *supply-driven* or a *hybrid* framework. They properly define each framework as follows:

- Supply-driven approaches: Also known as *data-driven*, start from a detailed analysis of the data sources to determine the multidimensional concepts in a reengineering process.

- Demand-driven approaches: Also known as *requirement-driven* or *goal-driven*, focus on determining the user multidimensional requirements (as typically performed in other information systems) to later map them onto data sources.

- Hybrid approaches: Propose to combine both paradigms in order to design the data warehouse from the data sources but bearing in mind the end-user requirements. In this document, we distinguish as well between *interleaved hybrid* approaches and *sequential hybrid* approaches. The main difference is that sequential approaches perform the demand-driven and supply-driven stages independently and later on conciliate results got in a final step,

[2]Note that we do not consider the **All levels** in this example.

whereas interleaved approaches perform both stages simultaneously benefiting from feedback retrieved by each stage all over the whole process and obtaining better results at the end.

Each paradigm has its own advantages and disadvantages, which we remark in the following discussion:

Carrying out an exhaustive search of dimensional concepts among all the concepts of the domain (like supply-driven approaches do) has a main benefit with regard to those approaches that derive the schema from requirements and later conciliate them with the data sources (i.e., demand-driven approaches): in many real scenarios, the user may not be aware of all the potential analysis contained in the data sources and, therefore, overlook relevant knowledge. Demand-driven and hybrid approaches do not consider this and assume that requirements are exhaustive. Thus, knowledge derived from the sources not depicted in the requirements is not considered and discarded. As a counterpart, supply-driven approaches tend to generate too many results (since they overlook the multidimensional requirements, they must apply their design patterns all over the data sources) and mislead the user with non relevant information. Furthermore, demand-driven approaches (or demand-driven stages within a hybrid approach) are not automated whereas supply-driven stages tend to facilitate their automation. The main reason is that demand-driven stages would require to formalize the end-user requirements (i.e., translate them to a language understandable by computers). Unfortunately, current methods handle requirements mostly stated in languages (such as natural language) lacking the required degree of formalization. Thus, matching requirements over the data sources must be performed manually. However, the time-consuming nature of this task can render it unfeasible when large databases are used.

In general, most approaches do not automate the process and just present a set of steps (i.e., a guideline) to be followed by an expert in order to derive the multidimensional schema. Mainly, these methods introduce different patterns or heuristics to discover concepts likely to play a multidimensional role and to carry out these approaches manually it is compulsory to have well-documented data sources at the expert's disposal. This prerequisite is not easy to fulfill in many real organizations and in order to solve this problem, current automatable methods directly work over relational databases (i.e., getting up-to-date data). These methods (or stages within hybrid approaches) follow a supply-driven paradigm and thus, rely on a thorough analysis of the relational sources. Relevantly, they mainly share three limitations:

User requirements not considered: Despite user requirements are essential to fulfill the expectations of end-users, 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. The multidimensional schema is eventually derived in a reengineering process from the relational schemas by applying these patterns.

Design patterns to identify facts: Identifying facts automatically is a hard task and these approaches either demand to identify facts manually or they rely on heuristics such as table cardinalities or numerical attributes that may identify false facts or overlook real ones. Certainly, these heuristics may help to identify potential facts but their isolate use without considering other inputs (for instance, requirements or additional semantics or metadata)

generates too much results that will surely contain facts of no interest for the end-user (since requirements are not considered) or they may even discard interesting ones (for instance, *factless facts* [KRTR98]).

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) [KRTR98]. Starting from a logical schema, however, may present some inconveniences when looking for many-to-one relationships. A relational schema is tied to design decisions made when devising the system and these decisions either made to fulfill the system requirements (for instance, improve query answering, avoid insertion / deletion anomalies, preserve features inherited from legacy systems, etc.) or naively made by non-expert users, have a big impact on the quality of the multidimensional schemas got by current automatable approaches. In fact, these approaches require a certain degree of normalization in the input logical schema to guarantee that it captures as much as possible the to-one relationships existing in the domain. Discovering this kind of relationships is crucial in the design of the data warehouse, and the most common way to represent them at the logical level is by means of foreign and candidate key constraints. Therefore, the accuracy of results depends on the degree of normalization of the logical schema, since some FKs and CKs are lost if we do not use a schema up to third normal form.

Summing up, *demand-driven approaches assume that requirements are exhaustive*, whereas *supply-driven approaches rely on discovering as much multidimensional knowledge as possible*. As a consequence, *supply-driven approaches generate too many results*. Furthermore, *current automatable methods follow a supply-driven approach*, whereas *current demand-driven approaches overlook the process automation*, since they tend to work with requirements at a high level of abstraction. Finally, all *current hybrid approaches follow a sequential approach with two well-differentiated steps*: the supply-driven and the demand-driven stages. Each one of these stages, however, suffers the same drawbacks as pure supply-driven or demand-driven approaches.

1.7 Contributions

Previous experiences in the data warehouse field have shown that the data warehouse multidimensional conceptual schema must be derived from a hybrid approach: i.e., by considering both the end-user requirements and the data sources, as first-class citizens. Like in any other system, requirements guarantee that the system devised meets the end-user necessities. In addition, since the data warehouse design task is a reengineering process, it must consider the underlying data sources of the organization: (i) to guarantee that the data warehouse must be populated from data available within the organization, and (ii) to allow the end-user discover unknown additional analysis capabilities.

As discussed in previous section, the latter issue leaded to the development of several approaches relying on a thorough analysis of the data sources. As previously discussed in this

chapter (see Section 1.2.1), most current digital events recorded are suitable for being analyzed from a multidimensional point of view and, consequently, these methods generate two many results. Filtering the results provided by these approaches is a must, but currently, they leave to the user the burden to (manually) filter them. Similarly, demand-driven approaches consider the requirements in depth and thus, they generate proper-sized solutions. However, these methods make the user responsible for mapping the requirements over the data sources. Indeed, these tasks are not just hard and time-consuming but can even be unfeasible for large data sources.

In this thesis we introduce two different approaches for automating the multidimensional design of the data warehouse: MDBE (*Multidimensional Design Based on Examples*) and AMDO (*Automating Multidimensional Design from Ontologies*). Both approaches were devised to overcome the current limitations discussed in previous section. On the one hand, we rely on the end-user requirements, but we do not decline that the data sources may also contain hidden analysis capabilities that, eventually, may be of interest. Nevertheless, in any case, we do not generate endless chunks of results from the sources. On the contrary, we aim at filtering by means of objective evidences the results obtained by analyzing the sources. Importantly, our approaches consider opposite initial assumptions, but both consider the end-user requirements and the data sources as first-class citizens. Furthermore, we also focus on the automation of the process, to facilitate the designer task as much as possible.

Nowadays, we may find several methods for supporting the data warehouse conceptual design but, all of them, start from very different assumptions that make them hardly comparable. For example, some approaches claim to fully automate the design task, but they do so by overlooking the end-user requirements in a fully supply-driven approach (and thus, making the user responsible for manually filtering the results obtained according to his / her needs). Similarly, exhaustive demand-driven approaches claim to derive high-quality outputs, but they completely overlook the task automation. For this reason, every approach fits to a narrow-ranged set of scenarios and do not provide an integrated solution for every real-world case. Importantly, our methods establish a combined and comprehensive framework that can be used to decide, according to the inputs provided in each scenario, which is the best approach to follow. For example, we cannot follow the same approach in a scenario where the end-user requirements are clear and well-known, and in a scenario in which the end-user requirements are not evident or cannot be easily elicited (e.g., this may happen when the users are not aware of the analysis capabilities of their own sources). Interestingly, the need to dispose of requirements beforehand is smoothed by the fact of having semantically rich data sources. In lack of that, requirements gain relevance to extract the multidimensional knowledge from the sources. So that, we claim to provide two approaches whose combination turns up to be exhaustive with regard to the scenarios discussed in the literature.

1.7.1 A Comprehensive Framework: What Are Indeed Current Approaches Providing?

To overcome the situation above discussed, we aim to establish a clear framework in which place the most relevant design methods introduced in the literature. This comprehensive picture of the state of the art will help to identify the major drawbacks of current approaches and, regarding

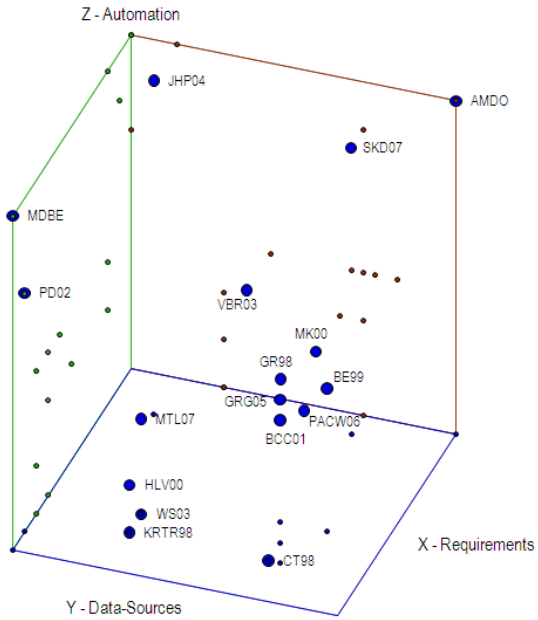


Figure 1.3: A comprehensive framework for introducing current multidimensional design methods (I)

them, this thesis main contributions. As earlier introduced in this chapter and discussed in depth in Chapter 2, the data warehouse design task must consider (i) the end-user requirements and (ii) the data sources. Furthermore, we also aim to analyze the (iii) automation degree achieved and (iv) the quality of the output produced. In the following, we rate the most relevant methods introduced in the literature with regard to these four criteria. Note that, in this way, we are able to identify, at first sight, the assumptions made by each approach and moreover, analyze the goodness of the process proposed: i.e., the automation degree achieved and the quality of the outputs produced.

Consider Figures 1.3 and 1.4. Axes x and y represent the assumptions of each method; i.e., the use of the requirements and the data sources in each approach. The x axis measures how important requirements are in the approach, and if the method proposes to formalize them somehow, to facilitate their analysis. The y axis assesses how important the analysis of the data sources is for the approach, and if detailed patterns are provided to exploit them. Finally, axes z measure either the automation degree achieved (see Figure 1.3) and the quality of the output produced (see Figure 1.4) regarding the assumptions made by the method (i.e., axes x and y). In the 3D-space formed, every approach is identified as a rhombus labeled with the bibliographical item in the reference list of this thesis. Furthermore, for the sake of understandability, we provide the projection of each rhombus in the three planes (green points for the XZ plane projections;

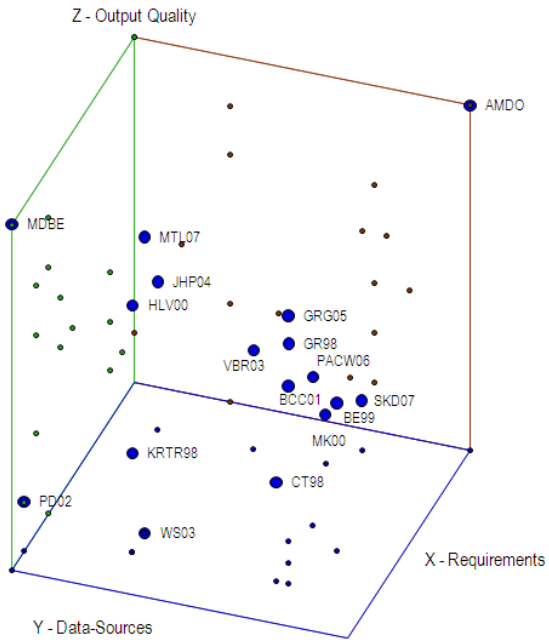


Figure 1.4: A comprehensive framework for introducing current multidimensional design methods (II)

blue points for the XY plane and red points for the XZ plane). The methods depicted in both figures are the MDBE (see Chapter 3) and AMDO (see Chapter 4) approaches proposed in this thesis, and the most relevant multidimensional design methods introduced in the literature. Each approach is placed in the 3D-space according to the conclusions extracted from Section 2.1.4.

On the one hand, the first figure shows that the automation degree achieved, in the general case, is medium or low. Only 6 approaches automate the design task up to a fair degree. Importantly, this figure shows our first main contribution: both the MDBE and AMDO approaches automate the design task as much as possible. However, note that they follow two different paradigms. Despite considering both, requirements and the data sources as first-class citizens, MDBE leads the process by exploiting the knowledge of the requirements, whereas AMDO leads the process from a thorough analysis of the data sources (and thus, they are placed in opposite vertexes). On the other hand, the quality of the output produced by most approaches is medium / high, but MDBE and AMDO produce yet semantically richer outputs. In other words, they are able to extract more valuable knowledge from the requirements / data sources. All these assertions will be properly justified in this document, but our approaches provide detailed algorithms for discovering multidimensional concepts traditionally overlooked such as *factless facts*, *bases*, *aggregate measures* and *semantic relationships* between the multidimensional concepts

identified.

As said, both methods are complementary, as each one starts from a different set of assumptions. In this thesis, our objective is to provide the best method for each potential real-world scenario we may find. In this sense, MDBE follows a classical approach. It assumes that the data warehouse designer has been able to gather the organization multidimensional requirements (it means, then, that the end-user has been able to clearly state which are his / her informational needs). By analyzing the data sources and requirements, MDBE generates schemas fulfilling the requirements and conciliating them with the data sources. In contrast with MDBE, AMDO focuses on those scenarios in which the multidimensional requirements are not clear, and a thorough analysis of the organization data is required. Nowadays, many organizations are not aware of their own data and therefore, of their own potential analysis perspectives. In these cases, it is interesting to use AMDO to discover them and help the end-users to decide what kind of analysis could be of their interest. However, different from traditional supply-driven approaches, AMDO filters results obtained in each stage (by means of quality evidences) and performs the upcoming stages with knowledge known to be of interest to the user.

1.7.2 Integrating Requirements in a Largely Automated Design Approach

Our first relevant contribution is a largely automated approach for supporting multidimensional design based principally on *Multidimensional Design By Examples* (MDBE); an automated method following an interleaved hybrid approach. Unlike other hybrid approaches, MDBE does not carry out two well-differentiated phases (i.e., data-driven and requirement-driven) that need to be conciliated a posteriori but instead performs both phases simultaneously. Consequently, each paradigm benefits from feedback obtained by the other, and eventually MDBE is able to derive more valuable information than approaches in which the two phases are carried out sequentially. To our knowledge, this is the first interleaved hybrid method introduced in the literature.

In our approach we derive MD *conceptual* schemas from *relational sources* according to end-user *requirements*. There are two steps: requirement formalization and the MDBE method. As in previous requirement-driven methods (or requirement-driven stages within hybrid methods), a prior requirements elicitation step is required. However, our approach is not based on a step-by-step manual process in which the requirements and data sources that will eventually derive the MD schema are analyzed in details, but rather on a largely automatable approach. Requirements are typically expressed at a high level of abstraction and need to be formalized prior to automation of the analysis step. In our framework, requirements are expressed as SQL queries over the relational data sources (i.e., at the logical level over the data sources). SQL queries provide a clearly defined structure that will facilitate full automation of the MDBE method (the second step in our approach). Although requirement formalization must be performed manually, translating requirements into SQL queries requires considerably less effort than carrying out any of the step-by-step requirement-driven approaches in current use. In our approach, we have reduced the amount of manual operations as much as possible (i.e., removing ambiguous semantics by formalizing the requirements) and delegated most of the design workload to the MDBE method, which will use the semantics captured in the requirements and the data sources to automate the rest of the process.

The inputs of the MDBE method are the end-user information requirements (expressed as

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

1.7.2.1 Correspondence Between SQL and the Multidimensional Algebra

The validation step of the MDBE method gives rise to another relevant contribution. The MDBE method generates sound and meaningful multidimensional schemas by validating if each input SQL query represents a valid multidimensional query. In other words, if the input SQL query represents a valid set of multidimensional operators.

Unfortunately, we do not dispose of a standard set of multidimensional operators (see section 1.3). To overcome this major drawback we carried out a thorough study to identify a set of multidimensional operators subsuming all the operators introduced in the literature. Then, we analyzed how these operators should be translated into SQL queries (i.e., when mapping from the multidimensional to the relational model, like a ROLAP tool would internally do). As consequence, we were able to identify the subset of SQL corresponding to the multidimensional algebra and viceversa; as well as three problematic scenarios that must be considered (and fixed if needed) during the translation process to avoid changing the semantics of the multidimensional query performed.

This study conforms the foundations of the MDBE validation process. An input SQL query is said to represent a valid set of multidimensional operators if we are able to find a correct mapping from the SQL query to the multidimensional algebra. If so, we will be able to find a multidimensional schema meeting the requirements stated in the query. Note that, beyond our purpose, this work can be reused for validating the multidimensional queries generated in a ROLAP tools. Importantly, the criteria identified in this study correspond to the integrity constraints of the multidimensional model considering multidimensionality as the data structure, and the whole set of multidimensional operators surveyed as the set of operators allowed to handle data.

1.7.3 Automatic Multidimensional Design from Ontologies

AMDO assumes a scenario in which the data sources available are semantically richer and therefore, it does not ask for the multidimensional requirements beforehand. Oppositely to MDBE, it is a sequential hybrid approach (a fully supply-driven followed by a demand-driven stage) aimed at discovering relevant knowledge for analysis purposes. Furthermore, it does not work over relational sources but from a conceptual view of the organization business. Indeed, we start from a conceptual formalization of the domain to avoid being tied to logical design decisions that

would directly impact on the quality of the output schemas. The role of a conceptual layer on top of information systems has been discussed in depth in the literature. In case of reengineering processes like the data warehouse conceptual design, the benefits are clear: the conceptual layer provides more and better knowledge about the domain to carry out this task. Note that MDBE overcomes the limitations of the logical schemas by means of the multidimensional requirements.

In our approach we choose *Description Logics* (DL) ([BCM+03]) ontologies as our method input. Thus, we benefit from the reasoning services provided by ontology languages, that will facilitate the automation of our task. Although other works already proposed to work at the conceptual level, AMDO is the first method presented in the literature automating the whole design task: i.e., identifying facts, measures and dimension hierarchies. As a novel contribution, AMDO also identifies potential bases (see section 1.5) of interest for each fact discovered. AMDO considers all the multidimensional concepts in depth by analyzing their semantics and how they should be identified from the sources. As result we propose new and original design patterns. Previous (conceptual) approaches mainly rely on their requirement elicitation (i.e., demand-driven) stages to discover the multidimensional concepts rather than on an accurate analysis of the data sources. In this sense, AMDO follows a completely different framework based on a thorough and fully automatic analysis of the sources and then, carrying out a guided requirement elicitation stage a posteriori. Therefore, unlike previous approaches, the automatic analysis of the sources leads the process and we allow the designer to restrict and control the process by stating his / her requirements on the fly. Importantly, AMDO does not generate endless amounts of multidimensional results, but, prior to show them to the user, filters them according to quality indicators.

To our knowledge, our approach is the first one considering the data warehouse design from ontologies. Hence, we do believe that this work opens new interesting perspectives. For example, we can extend the data warehouse and OLAP concepts to other areas like the Semantic Web, where ontologies play a key role providing a common vocabulary. One consequence would be that despite the data warehouse design has been typically guided by data available within the organization, we would be able to integrate external data from the web into our data warehouse to provide additional up-to-date information about our business domain (this novel concept of data warehousing is known in the literature as *Web-Warehousing* [RALT06]).

1.8 Organization of the Thesis

This thesis has been organized in five chapters (including this one). A brief overview of each one is shown below.

1.8.1 Second Chapter: Related Work

This chapter presents the state of the art of two different topics: multidimensional design methods and multidimensional algebras. While the first one aims to contextualize the work carried out in this thesis, the second one is needed to identify the multidimensional model constraints, which must be enforced when automating the multidimensional design task.

All in all, we present a detailed picture of the current state of both fields, as well as a comprehensive comparison between current methods / algebras presented in the literature.

1.8.2 Third Chapter: Integrating Requirements in a Largely Automated Design Approach

This chapter presents the first of the two multidimensional design methods introduced in this thesis: a largely automated approach based on MDBE.

We start this chapter explaining the overall idea behind this approach. This method takes a classical approach. We assume that the end-user knows his / her informational needs and therefore, the system designer will be able to gather the multidimensional requirements. After a detailed explanation of how MDBE works, we discuss its set of contributions regarding the related work. As discussed in this chapter, MDBE is the first method formalizing the end-user requirements (as SQL queries over the OLTP systems), and automating their use at the same time as analyzing the data sources. This approach gives rise to a new and original framework that we call interleaved hybrid approach.

Next, we discuss how we are able to derive correct multidimensional schemas from SQL queries representing the multidimensional requirements. Our work focuses on a deep analysis of the multidimensional semantics regarding SQL. In short, we identify the mapping between the multidimensional algebra and SQL, which relevantly, it is the same that a ROLAP tool would exploit. If the SQL query represents a correct multidimensional navigation path (i.e., a set of multidimensional operators), we map it onto the data sources and accordingly, we identify which multidimensional role each relational concept must play. Eventually, this mapping will produce a multidimensional schema.

Once the foundations of the MDBE method have been set, we introduce a detailed step-by-step view of our method. Furthermore, we also introduce a practical case of study (the TPCH-H) to show the feasibility of the method and the quality of results obtained. Prior to present our conclusions, we also introduce the MDBE tool that implements our method.

1.8.3 Fourth Chapter: Multidimensional Design from Ontologies

This chapter presents our second approach for automating the multidimensional design task: the AMDO method. This chapter emulates as much as possible the structure of the previous one.

First, we introduce the main idea behind our approach. Although classical approaches for information systems design start with a requirement elicitation stage, sometimes, it may be difficult to gather them. The data warehousing system design is a reengineering task and therefore, it must consider the underlying operational sources of the organization. This approach, despite considering the end-user requirements as first-class citizens, it starts by thoroughly analyzing the data sources, as any supply-driven approach would do. However, we provide a novel framework by completely automating the process from the conceptual point of view. After discussing the general idea and the main contributions of AMDO, we present our method and we set and formalize its working context. First, we introduce the multidimensional patterns it applies to analyze the data sources. Then, we discuss how to efficiently compute them. Relevantly, AMDO relies on a novel algorithm conceived to discover bases (i.e., multidimensional keys), which have been traditionally overlooked in the design process.

In this chapter, we provide two different case studies upon which we show AMDO's feasibility: the EU-Car Rental and the TPC-H. Finally, we conclude by introducing the the AMDO tool,

which implements our approach.

1.8.4 Fifth Chapter: Conclusions and Further Work

Conclusions about the work presented in this thesis and future work to be carried out are discussed in this section.

Chapter 2

Related Work

" In the middle of difficulty lies opportunity. "

Albert Einstein

The related work discussed in this chapter refers to two different topics: multidimensional design methods and multidimensional algebras.

The necessity of the first study is clear. This thesis introduces two novel multidimensional design methods and therefore, it is a must to clearly depict the current situation of the area. On the contrary, the reason why a study about multidimensional algebras is relevant to this thesis is, at first sight, subtler. As discussed in Section 1.7, one of this thesis main objectives is to automate, as much as possible, our design approaches and consequently, automatically produce *correct* multidimensional schemas. Automating the design task, however, entails that outputs produced must have been *validated* during the process. A multidimensional schema is correct if (i) it is *aligned* with the multidimensional data structure and (ii) preserves the multidimensional integrity constraints (see Section 1.3). Only then, we can guarantee that data manipulation (by means of the multidimensional operators) will always be correct.

Despite we do not yet benefit from a standard multidimensional model, there is a general consensus on the model data structure (indeed, the multidimensionality introduced by Kimball -see Section 1.2.1- is assumed to be a *de facto* data structure standard), but this is not the case of the model integrity constraints and set of operators. For this reason, we surveyed all the multidimensional algebras that, to our knowledge, were introduced in the literature. After this analysis, we were able to find an implicit agreement on how multidimensional data should be manipulated and eventually, identify the multidimensional integrity constraints preserving a correct data manipulation. Thus, the schemas produced by our approaches guarantee the multidimensional constraints identified in our study and consequently, they fully make multidimensional sense.

For both topics, we first present a detailed state of the art and later, a comprehensive framework that will facilitate the comparison and further discussion of the methods / algebras we may

find in the literature.

2.1 Multidimensional Design Methods

2.1.1 Terminology

For the sake of understandability, in this document we take advantage of the method classification introduced by Winter & Strauch [WS03]. For this reason, we first introduce the notation proposed. According to it, multidimensional modeling methods may be classified within a *demand-driven*, a *supply-driven* or a *hybrid* framework. They properly define each framework as follows:

- Supply-driven approaches: Also known as *data-driven*, start from a detailed analysis of the data sources to determine the multidimensional concepts in a reengineering process.
- Demand-driven approaches: Also known as *requirement-driven* or *goal-driven*, focus on determining the user multidimensional requirements (as typically performed in other information systems) to later map them onto data sources.
- Hybrid approaches: Propose to combine both paradigms in order to design the data warehouse from the data sources but bearing in mind the end-user requirements. In this document, we distinguish as well between *interleaved hybrid* approaches and *sequential hybrid* approaches. The main difference is that sequential approaches perform the demand-driven and supply-driven stages independently and later on conciliate results got in a final step, whereas interleaved approaches perform both stages simultaneously benefiting from feedback retrieved by each stage all over the whole process and obtaining better results at the end.

2.1.2 A Comprehensive Survey

This section presents an insight into current multidimensional design methods. These methods were selected according to three factors: reference papers with a high number of citations (according to Google Scholar [Goo] and Publish or Perish [Har]), papers with novelty contributions and in case of papers of the same authors, we have included the latest version of their works. As general rule, each method is described and classified according to the terminology presented in Section 1.5. Finally, we follow a chronological order when introducing the design methods surveyed. Thus, we provide a comprehensive framework of the evolution of multidimensional design methods (note that Section 1.4.1 sketches this survey):

Kimball et al. [KRTR98] introduced multidimensional modeling as known today. In addition, they also introduced the first method to derive the multidimensional schema. Being the first approach, it does not introduce a formal design procedure, but a detailed guide of tips to identify the multidimensional concepts and then, give rise to the multidimensional schema. The presentation is quite informal and it relies on examples rather than on formal

rules. Kimball's approach follows a demand-driven framework to derive a data warehouse relational schema (i.e., logical).

First, the designer must identify all the data marts we could possibly build. Data marts are essentially defined as pragmatic collections of related facts. Although data sources are not considered, they already suggested to take a look to the data sources to find which data marts may be of our interest.

Next step aims to list all conceivable dimensions for each data mart. At this point it is suggested to build an ad hoc matrix to capture our multidimensional requirements. Rows represent the data marts, whereas columns represent the dimensions. A given cell is marked whether that dimension must be considered for a data mart. This matrix is also used to show the associations between data marts by looking at dimensions shared. This process is supposed to be incremental. First, it is suggested to focus on single-source data marts, since it will facilitate our work and later, in a second iteration, look for multiple-sources data marts combining the single-source designs.

The method's third step designs the fact tables of each data mart:

- First, we must declare the *grain* of detail (i.e., the data granularity of interest). It is suggested to be declared by the design team at the beginning, although it can be reconsidered during the process. Normally, it must be determined by primary dimensions.
- Next, we choose the analysis dimensions for each fact table. Dimensions selected must be tested against the grain selected. This must be a creative step. We need to look for the dimension *pieces* (i.e. levels and descriptors) in different (and potentially heterogeneous) models and through different documents which, in the end, results in a time-consuming task. At this point, it is also suggested to choose a large number of descriptors to populate dimensions.
- Finally, the last stage adds as many measures as possible within the context of the declared grain.

Cabibbo and Torlone [CT98a] present one of the most cited multidimensional design methods. This approach generates a logical schema from *Entity-Relationship* [Che76] (ER) diagrams, and it may produce multidimensional schemas in terms of relational databases or multidimensional arrays. At first sight, this method may be thought to follow a supply-driven paradigm, as it performs an in-depth analysis of the data sources. However, no formal rules to identify the multidimensional concepts from the data sources are given. In fact, multidimensional concepts must be manually identified by the user (i.e., from requirements). For this reason, we consider it to follow a hybrid framework. In general, like Kimball's approach, this approach is rather informal but they set up the foundations that were later used by the rest of methods.

This method consists of four steps. First and second steps aim to identify facts and dimensions and restructure the ER diagram. Both steps may be performed simultaneously and benefit from the feedback retrieved by each step. Indeed the authors suggest to perform them in an iterative way to refine results obtained. However, no clue about how to identify

facts, measures and dimensions are given and they must be identified from the end-user requirements. Once they identified, each fact is represented as an entity. Next, we add dimensions of interest that may be missing in the schema but could be derived from external sources or metadata associated to our data sources. At this point, it is also compulsory to refine the levels of each dimension by means of the following transformations: (i) replacing many-to-many relationships, (ii) adding new concepts to represent new levels of interest, (iii) selecting a simple identifier for each level entity and (iv) removing irrelevant concepts. Finally, two last steps aim to derive the multidimensional schema. Some clues are given to derive a multidimensional graph that will be directly mapped into the multidimensional schema.

Golfarelli and Rizzi [GMR98] [1] present one of the reference methods in this area. This work presents a generic overview of the multidimensional design process that embraces their previous works such as [GR98], and recently revisited in [GR09]. This approach presents a formal and structured method (partially automatable) that consists of six well-defined steps. However, the fourth step aims to estimate the data warehouse workload which goes beyond the scope of this study:

- First step analyzes the underlying information system and produces a conceptual schema (i.e., a ER diagram) or a logical schema (i.e., a relational schema).
- Second step collects and filters requirements. In this step it is important to identify facts. The authors give some tips to identify them from ER diagrams (entities or n-ary relationships) or relational schemas (tables frequently updated are good candidates).
- Next step derives the multidimensional conceptual schema from requirements and facts identified in previous steps. This step may be carried out semi-automatically as follows:
  - Building the attribute tree: From the primary key of the fact we create a tree by means of functional dependencies. Thus, a given node (i.e., an attribute) of the tree functionally determines its descendants.
  - Pruning and grafting the attribute tree: The tree attribute must be pruned and grafted in order to eliminate unnecessary levels of detail.
  - Defining dimensions: Dimensions must be chosen in the attribute tree among the root vertices.
  - Defining measures: Measures are defined by applying aggregation functions, at root level, to numerical attributes of the tree.
  - Defining hierarchies: The attribute tree shows a plausible organization for hierarchies. Hierarchies must be derived from to-one relationships that hold between each node and its descendants.
- Finally, the last two steps derive the logical (by translating each fact and dimension into one relational table) and physical schemas (the authors give some tips regarding indexes to implement the logical schema in a ROLAP tool).

[1] A revisited version of this design method has been published in [GR09]. However, to preserve the chronological order, we will refer to [GMR98] in this section.

The fourth step of this method aims to estimate the workload of the data warehouse. The authors argue that this process may be used to validate the conceptual schema produced in the third step, as queries could only be expressed if measures and hierarchies have been properly defined. However, no more information is provided.

Boehnlein and Ulbrich-vom Ende [BvE99] present a hybrid approach to derive logical schemas from SER (Structured Entity Relationship) diagrams. SER is an extension of ER that visualizes existency dependencies between objects. For this reason, the authors argue that SER is a better alternative to identify multidimensional structures. This approach has three main stages:

- Pre-process: First, we must transform the ER diagram into a SER diagram. A detailed explanation is provided.
- Step 1: Business measures must be identified from goals. For instance, the authors suggest to look for business events to discover interesting measures. Once business measures have been identified, they are mapped to one or more objects in the SER diagram. Eventually, these measures will give rise to facts.
- Step 2: The hierarchical structure of the SER diagrams is helpful to identify potential aggregation hierarchies. Dimensions and aggregation hierarchies are identified by means of direct and transitive functional dependencies. The authors argue that discovering dimensions is a creative task that must be complemented with a good knowledge of the application domain.
- Step 3: Finally, a star or snowflake schema is derived as follows: each fact table is created by using the set of primary keys of their analysis dimensions as its compound primary key, and denormalizing or normalizing the aggregation hierarchies accordingly.

Hsemann et al. [HLV00] present a requirement-driven method to derive multidimensional schemas in *multidimensional normal form* (MNF). This work introduces a set of restrictions that any multidimensional schema produced by this method will satisfy. Furthermore, although this approach produces conceptual schemas, they also argue that the design process must comprise four sequential phases (requirements elicitation and conceptual, logical and physical design) like any classical database design process:

- Requirement analysis and specification: Despite it is argued that the operational ER schema should deliver basic information to determine the multidimensional analysis potential, no clue about how to identify the multidimensional concepts from the the data sources is given. Business domain experts must select strategically relevant operational database attributes and specify the purpose to use them as dimensions or measures. The resulting requirements specification contains a tabular list of attributes together with their multidimensional purpose, similar to Kimball's proposal. Supplementary informal information may be added such as standard multidimensional queries that the user would like to pose.

- Conceptual design: This step transforms the semi-formal business requirements into a formalized conceptual schema. This process is divided in three sequential stages:
  - Context definition of measures: This approach requires to determine a base for each measure (i.e., a minimal set of dimension levels functionally determining the measure values). Furthermore, measures sharing bases are grouped into the same fact, as they share the same dimensional context.
  - Dimensional hierarchy design: From each atomic level identified, this step gradually develops the dimension hierarchies by means of functional dependencies. Descriptors and levels are distinguished from requirements. In this approach, the authors distinguish between simple and multiple (containing, at least, two different aggregation path) hierarchies. Moreover, specialization of dimensions must be considered to avoid structural *NULL* values when aggregating data.
  - Definition of summarizability constraints: The authors argue that some aggregations of measures over certain dimensions do not make sense. Therefore, they propose to distinguish meaningful aggregations from meaningless ones and include this information in an appendix of the conceptual schema.

Finally, the authors argue that a multidimensional schema derived by means of this method is in dimensional normal form (MNF) [LAW98] and therefore it fully makes multidimensional sense. Consequently, we can form a data cube (i.e., a multidimensional space) free of summarizability problems. In short, it is achieved by means of five constraints: measures must be fully functionally identified by the multidimensional base, each dimension hierarchy must have an atomic level, each dimension level must be represented by identifier attribute(s), every descriptor must be associated to a dimension level and dimensions generated must be orthogonal. By following their method, all these constraints are guaranteed.

Moody and Kortink [MK00] present a method to develop multidimensional schemas from ER models. It was one of the first supply-driven approaches introduced in the literature, and one of the most cited papers in this area. Although it is not the first approach working over ER schemas, they present a structured and formal method to derive multidimensional logical schemas. Their method is divided into four steps:

- Pre-process: This step develops the enterprise data model if it doesn't exist yet.
- First step: This step classifies the ER entities in three main groups:
  - Transactional entities: These entities record details about particular events that occur in the business (orders, sales, etc). They argue that these are the most important entities in a data warehouse and form the basis of fact tables in star schemas, as these are the events that decision makers want to analyze. Although the authors do not consider requirements, they underline the relevance of requirements to identify facts, because not all the transactional entities will be of interest for the user. Moreover, they provide the key features to look for this kind of entities: the entity must describe an event that happens at a point in time, and it must contain measures or quantities summarizable.

  - Component entities: These entities are directly related to a transaction entity via a one-to-many relationship and they define details or components of each business event. These entities will give rise to dimension tables in star schemas.
  - Classification entities: These entities are related to component entities by a chain of one-to-many relationships. Roughly speaking, they are functionally dependent on a component entity directly or by transitivity. They will represent dimension hierarchies in the multidimensional schema.

  The authors assume that a given entity may fit into multiple categories. Therefore, they define a precedence hierarchy for resolving ambiguities: Transaction $>$ Classification $>$ Component. Thus, if an entity may play a transaction entity role, it is not considered neither as a classification nor a component entity. The rest of entities in the ER schema will not be included in the multidimensional schema.

- Second step: Next step aims to shape dimension hierarchies. The authors provide some formal rules to identify them. Specifically, a dimension hierarchy is defined as a sequence of entities joined together by one-to-many relationships all aligned in the same direction. Moreover, they introduce the concept of minimal entity (i.e., atomic level) and maximal entity (i.e., that with a coarser granularity data). Some formal rules to identify minimal and maximal entities are given. For instance, minimal entities are those without one-to-many relationships, and maximal are those without many-to-one relationships.
- Third step: Transactional entities will give rise to facts, whereas dimension hierarchies will give rise to their analysis perspectives. The authors introduce two different operators to produce logical schemas:
  - Collapse hierarchy: Higher levels in hierarchies can be collapsed into lower levels. Indeed, the authors propose to denormalize the hierarchies according to our needs, as typically performed in data warehousing to improve query performance.
  - Aggregation: Can be applied to a transaction entity to create a new entity containing summarized data. To do so, some attributes are chosen to be aggregated (i.e., measures) and others to aggregate by (i.e., dimensional concepts).

  By these operators, this approach introduces five different dimensional design alternatives. According to the resulting schema level of denormalization and the granularity of data, they introduce rules to derive *flat schemas, terraced schemas, star schemas, snowflake schemas* or *star cluster schemas*. They also introduce the notion of constellation schema that denotes a set of star schemas with hierarchically linked fact tables.

Bonifati et al. [BCC$^+$01] present a hybrid semi-automatic approach consisting of three basic steps: a demand-driven, a supply-driven and a third stage to conciliate the two first steps (i.e., it introduces a sequential hybrid approach). The final step aims to integrate and conciliate both paradigms and generate a feasible solution that best reflects the user's necessities. This method generates a logical multidimensional schema and it was the first to

introduce a formal hybrid approach with a final step conciliating both paradigms. Moreover, this method has been applied and validated in a real case study:

- We start collecting the end-user requirements through interviews and expressing user expectations through the *Goal / Question / Metrics* (GQM) paradigm. GQM is composed of a set of forms and guidelines developed in four stages: (i) a first vague approach to formulate the goals in abstract terms, (ii) a second approach using forms and a detailed guide to identify goals by means of interviews, (iii) a stage to integrate and reduce the number of goals identified by collapsing those with similarities and finally, (iv) a deeper analysis and a detailed description of each goal. Next, the authors present an informal guideline to derive a logical multidimensional schema from requirements. Some clues and tips to identify facts dimensions and measures from the forms and sheets used in this process are given.
- Second step aims to carry out a supply-driven approach from ER diagrams capturing the operational sources. This step may be automated, and it performs an exhaustive analysis of the data-sources. From the ER diagram, a set of graphs that will give rise to star schemas are created as follows:
  - They label potential fact entities according to the number of additive attributes they have. Each identified fact is taken as the center node of a graph.
  - Dimensions are identified by means of many-to-one and one-to-one relationships from the center node. Moreover, many-to-many relationships are transformed into one-to-many relationships. Finally, each generalization / specialization taxonomy is also included in the graphs.

  Next, they introduce an algorithm to derive snowflake schemas from each graph. This transformation is immediate and once done, they transform the snowflake schemas into star schemas by flattening the dimension hierarchies (i.e., denormalizing dimensions).
- Third step aims to integrate star schemas derived from the first step with those identified from the second step. In short, they try to map demand-driven schemas into supply-driven schemas by means of three steps:
  - Terminology analysis: Before integration, demand-driven and supply-driven schemas must be converted to a common terminological idiom. A mapping between GQM and ER concepts must be provided.
  - Schema matching: Supply-driven schemas are compared, one-by-one, to demand-driven schemas. A match occurs if both have the same fac. Some metrics, with regard to the number of measures and dimensions, are calculated.
  - Ranking and selection: Supply-driven schemas are ranked according to the metrics calculated in the previous step and presented to the user.

As final remark, this method does not introduce the concept of descriptor in any moment. However, since they map relational entities into levels, we may consider attributes contained in the entities as the multidimensional descriptors.

Phipps and Davis [PD02] introduced one of the first methods automating part of the design process. This approach proposes a supply-driven stage to be validated, a posteriori, by a demand-driven stage. It is assumed to work over relational schemas (i.e., at a logical level) and a conceptual multidimensional schema is produced. In this approach, their main objective is the automation of the supply-driven process with two basics premises: numerical fields represent measures and the more numerical fields a relational table has, the more likely to play a fact role. Furthermore, any table related with a to-many relationship is likely to play a relevant dimensional role. In general, they go one step beyond in the formalization of their approach since a detailed pseudo-algorithm is presented in this paper (and therefore, automation is immediate). However, this approach generates too many results and a demand-driven stage is needed to filter results according to the end-user requirements. Thus, the demand-driven stage in this approach is rather different from the rest of demand-driven approaches, because they do not derive the multidimensional schema from requirements but they use requirements to filter results. This method consists of five steps:

- First step finds tables with numerical fields and create a fact node for each table identified. Tables with numerical fields are sorted in descending order of number of numeric fields. Tables will be processed in this order.
- Second step creates measures based on numerical fields within fact tables.
- Third step creates date and / or time dimension levels with any date / time fields per fact node.
- Fourth step creates dimensions (consisting of just one level) for each remaining table attribute that is non-numerical, non-key and non date field. Although this may be considered as a controversial decision (any other attribute would give rise to a dimension of analysis), it was the first method handling partially denormalized data sources.
- Fifth step recursively examines the relationships of the tables to add additional levels in a hierarchical manner. To do so, it looks for many-to-one relationships (according to foreign keys and candidate keys) all over the schema.

The heuristics used to find facts and determine dimensional concepts within a fact table are rather generic, and they generate results containing too much noise. Consequently, the authors propose a final requirement-driven step to filter results obtained. This step present a step-by-step guide to analyze the end-user requirements expressed as MDX queries and guide the selection of candidate schemas most likely to meet user needs. This last step must be manually performed.

Winter and Strauch [WS03] present a detailed demand-driven approach. This is a reference paper because it presents a detailed discussion between different multidimensional design paradigms. Furthermore, they present a method developed from the analysis of several data warehouse projects in participating companies. However, their approach is rather different from the rest of methods. They do not assume the multidimensional modeling

introduced by Kimball like the rest of methods do, and they present a high-level step-by-step guideline.

In short, they identify the best practices that a data warehouse design project must consider, according to their analysis task. The design process must be iterative and it is divided into four stages:

- First step embraces the analysis of the information supply (i.e., from the sources) and the analysis of the information needed.
- Next, we must match requirements demanded with current information supply and order requirements accordingly.
- In a third step, information supply and information demand must be synchronized on a full level of detail (i.e., considering data granularity selected).
- Finally, we must develop the multidimensional schema. This schema must be evaluated and if needed, reformulate the process from the first step to develop the multidimensional schema in an iterative way.

Despite this approach gives relevance to the data sources and demands to synchronize data demanded with the sources, we consider it to be a demand-driven approach since no clue about how to analyze the data sources is given.

Vrdoljak et al. [VBR03] present a semi-automatic supply-driven approach to derive logical schemas from XML schemas. This approach considers XML schemas as data sources. Therefore, the authors propose to integrate XML data in the data warehouse, as XML is now a *de facto* standard for the exchange of semi-structured data. Their approach works as follows:

- Preprocessing the XML schema: The schema is simplified to avoid complex and redundant specifications of relationships.
- Creating and transforming the schema graph: Every XML schema can be represented as a graph. Two transformations are carried out at this point; functional dependencies are explicitly stated (by means of key attributes) and nodes not storing any value are discarded.
- Choosing facts: Facts must be chosen among all *vertexes* (i.e., nodes) and *arcs* (i.e., edges) of the graph. An arc can be chosen only if it represents a many-to-many relationship.
- Building the dependency graph: For each fact, a dependency graph is built. The graphical representation of the XML schema facilitates finding the functional dependencies. The graph must be examined in the direction expressed by arcs and according to cardinalities included in the dependency graph. It may happen that no cardinality is provided. In this case, XML documents are queried by means of XQueries to look for to-one relationships. The authors also consider many-to-many relationships to be of interest in some cases. However, these cases must be manually identified by the user. Finally, the dependency graph will give rise to aggregation hierarchies.

- Creating the logical schema: Facts and measures are directly depicted from vertexes and arcs chosen whereas dimensions are derived from the aggregation hierarchies identified.

Jensen et al. [JHP04] present a supply-driven method from relational databases. They present data-mining techniques to be applied over the intensional data to discover functional and inclusion dependencies and, eventually, derive snowflake schemas.

Their method starts collecting metadata such as table and attribute names, cardinality of attributes, frequency, etc. Later, data is divided into three groups according to its potential multidimensional role: measure, keys and descriptive data. Next, integrity constraints such as functional and inclusion dependencies are identified between attributes and finally, the snowflake schema is produced.

First two steps are performed consulting the database catalog. The role of each attribute is derived with a bayesian network that takes as input metadata collected for each attribute. Third step discovers the database structure by identifying functional and inclusion dependencies that represent many-to-one relationships that will give rise to dimensions. Candidate keys and foreign keys are identified assuming that there are no composite keys in the database. Furthermore, inclusion dependencies among foreign keys and candidate keys are identified in this step. These dependencies will be mainly used to identify dimensions. This step is critical, since all permutations of candidate keys and foreign keys are constructed with the consequent computational cost. To pair two keys, both must have the same attribute type and the candidate key must have, at least, as many distinct values for the attribute as the table containing the foreign key. If these constraints hold, a SQL statement is issued to check if the join of both tables (by means of these attributes) has the same cardinality as the table containing the candidate foreign key. If so, an inclusion dependency is identified between both keys. Next, they propose an algorithm to derive snowflake schema from this metadata:

- Fact tables are identified in a semi-automatic process involving the user. First, facts are proposed by means of the table cardinality and the number of measures identified by the bayesian network. Then, the user chooses those of his / her interest.
- Inclusion dependencies discovered form different connected graphs. A connected graph is considered to be a dimension if exists a inclusion dependency between a fact table and a graph node. In this case, that node will play the atomic level role of the dimension. The authors propose an algorithm to break potential cycles and give rise to the aggregation hierarchy from the graph. When shaping the aggregation hierarchy, two consecutive levels are analyzed to avoid aggregation problems (i.e., duplicated or lost values).

Giorgini et al. [GRG05] present a hybrid approach to derive the conceptual multidimensional schema. They propose to gather multidimensional requirements and later map them onto the data sources in a conciliation process. However, they also suggest that their approach could be considered a pure demand-driven if the user do not want to consider the data sources.

The authors introduce an agent-oriented method based on the *i** framework [Yu97]. They argue that it is important to model the organization setting in which the data warehouse will operate (organization modeling) and capture the functional and non-functional requirements of the data warehouse (what authors call the decisional modeling).

If we consider their hybrid approach, the next step is to match requirements with the schema of the operational sources. In this approach both ER diagrams and relational schemas are allowed as inputs describing the data sources. This matching stage consists of three steps:

- Requirement mapping: Facts, dimensions and measures identified during the requirement analysis are now mapped over the data sources. According to the kind of data sources considered, the authors introduce a set of hints to map each concept. For example, facts are mapped onto entities or n-ary associations in ER diagrams and onto relations in relational schemas.
- Hierarchy construction: For each fact identified, the data sources are analyzed looking for functional dependencies based on the algorithm already discussed in [GMR98].
- Refinement: This step aims to rearrange the fact schema in order to better fit the user's needs. In this process, we may distinguish among concepts available (mapped from requirements), unavailable (demanded in the requirements but not mappable to the data sources) and what is available and not needed. The authors propose to use this information to reorder dimensions (grafting and pruning the aggregation hierarchies) and / or try to find new directions of analysis.

Prat et al. [PACW06] present a method to derive the conceptual, logical and physical schema of the data warehouses according to the three abstraction levels recommended by ANSI / X3 / SPARC. Starting from end-user requirements, the conceptual phase leads to a *UML* [Grob] model. To this end, UML is enriched with concepts relevant to multidimensionality that will facilitate the generation of the logical schema. The logical phase maps the enriched UML model into a multidimensional schema and finally, the physical phase maps the multidimensional schema into a physical database schema depending on the target implementation tool (in this case Oracle MOLAP). At each phase, they introduce a metamodel and a set of transformations to perform the mapping between metamodels. In this study, we will focus on the method to produce the conceptual and logical schemas and we will avoid to discuss the transformations to be performed to derive the physical schema.

- Conceptual phase: In this first step, the authors embrace requirements elicitation and the conceptual representation of requirements. First, requirements should be captured by means of a UML-compliant system analysis method. Requirements engineering techniques used in transactional design processes may be considered, and for example, they mention interviews, joint sessions, study of existing reports and prototyping of future reports as potential techniques to be used. Next, requirements are represented in a UML class diagram that needs to be enriched to capture multidimensional semantics. To do so, they present an extension of the UML metamodel.

  - Classes which are not association classes are denoted as ordinary classes. Similarly, associations which are not association classes are denoted as ordinary associations.
  - Each attribute of an ordinary class must be identified as an attribute or not. According to authors, it must be decided by the end-user and designers jointly.
  - Each attribute belonging to one-to-one or many-to-one relationships is transferred to the to-many side.
  - Generalizations are transformed to facilitate their mapping to the logical level. Each specialization is mapped to a new class that is related to the superclass by means of an aggregation relationship.
- Logical phase: Creating the logical schema from the enriched conceptual model produced in the first phase is immediate and a set of transformations expressed in *Object Contraint Language* (OCL) [Groa] are presented. They also introduce an ad hoc multidimensional metamodel to represent the logical schema as follows:
  - Every many-to-many association of the conceptual model is identified as a fact of interest and their attributes (if any) are mapped into measures of the fact. This fact would be dimensioned by mapping the ordinary classes directly or indirectly involved in the association. Similarly, every ordinary class containing numerical values of interest is also identified as a fact. In this case, the fact is dimensioned by one dimension level defined by mapping the class (similar to the approach presented in [PD02]).
  - Next, following many-to-one relationships between ordinary classes we give rise to aggregation hierarchies for each dimension level identified in the previous step.
  - Descriptors are defined from those non-identifier attributes from the classes involved in the dimension hierarchy that have not been chosen as measures of interest.
  - Finally, for each measure and for each dimension related to the fact where the measure is defined, it is compulsory to define which aggregation functions preserve a meaningful aggregation.

Mazón et al. [MTL07] present a semi-automatic hybrid approach that obtains the conceptual schema from user requirements and then, verifies and enforces its correctness against the data sources by means of *Query / View / Transformation* (QVT) relations. Their approach work over relational sources and requirements expressed in the *i** framework. The modus operandi of this approach shares many common points with [BCC+01], but in this case, they also provide mechanisms for validating the output schema.

This approach starts with a requirement analysis phase. They introduce a detailed demand-driven stage in which the user should state his / her requirements at high level by means of business goals. Then, the information requirements are derived from the information business goals. Both, goals and information requirements must be modeled by an adaptation of the *i** framework and eventually, the multidimensional conceptual schema must be

derived from this formalization. Finally, the authors propose to express the resulting multidimensional schema by using an ad hoc UML extension (i.e., their own data structure) provided in the paper.

Next, they propose a final step to check the conceptual multidimensional model correctness. The objective of this step is twofold: they present a set of QVT relations based on the *multidimensional normal forms* (MNF) to align the conceptual schema derived from requirements with the relational schema of the data sources. Thus, output schemas will capture the analysis potential of the sources and at the same time, they will be validated according to the MNF. The MNF used in this paper are an evolution of those used in [HLV00], and they share the same objective. By means of five QVT relations that may be semi-automated, this paper describes how the conceptual multidimensional schema should be aligned to the underlying relational schema:

- 1MNF (a): A functional dependency in the conceptual schema must have a corresponding functional dependency in the relational schema.
- 1MNF (b): Functional dependencies among dimension levels contained in the source databases must be represented as aggregation relationships in the conceptual schema. Therefore, they complement the conceptual schema with additional aggregation hierarchies contained in the sources.
- 1MNF (c): Summarized measures that can be derived from regular measures must be identified in the conceptual schema. Therefore, they support derived measures.
- 1MNF (d): Measures must be assigned to facts in such a way that the atomic levels of the fact form a key. In other words, they demand to place the measure in a fact with the correct base (and thus, preserve the proper data granularity).
- 2MNF and 3MNF: These constraints demand to use specializations of concepts when structural *NULLs* in the data sources do not guarantee completeness.

Song et al. [SKD07] present an automatic supply-driven method that derives logical schemas from ER models. This novel approach automatically identifies facts from ER diagrams by means of the *connection topology value* (CTV). The main idea underlying this approach is that facts and dimensions are usually related by means of many-to-one relationships. Concepts at the many-side are fact candidates and concepts in the one-side are dimension candidates. Moreover, it distinguishes between direct and transitive many-to-one relationships:

- First, the authors demand a preprocess to transform ER diagrams into binary (i.e., without ternary nor many-to-many relationships) ER diagrams.
- The CTV value of an entity is a composite function of the topology value of direct and indirect many-to-one relationships. In this formula, direct relationships have a higher weighting factor with regard to transitive ones. Thus, all those entities with a CTV value higher than a threshold are proposed as facts. Note that facts are identified by their CTV and therefore, it would be possible to consider factless facts.

- For each fact entity, its analysis dimensions are identified by means of many-to-one relationships. Moreover, the authors propose to use *Wordnet* and annotated dimensions (that represent commonly used dimensions in business processes) to enrich aggregation hierarchies depicted.

This approach does not introduce any clue to identify measures, levels and descriptors. However, working over ER diagrams, it would be rather easy to assume that measures are identified by means of numerical attributes once a concept has been identified as a fact, whereas descriptors can be identified from those entities identified as dimensions. Furthermore, no clue about how to identify levels is given and indeed, in the examplification provided in the paper, every dimension identified contains just one level (i.e., they do not identify aggregation hierarchies).

2.1.3 Comparison Criteria

In order to provide a comprehensive framework of the multidimensional design methods, we aim to provide a detailed comparison of the methods discussed in the previous section. Setting a basis for discussion will facilitate the mapping of the surveyed methods to a common framework from which compare each approach, detect trends such as features in common or analyze the evolution of assumptions made by the modeling methods. For this reason, this section presents the criteria used in the comparison presented in Section 2.1.4.

These criteria were defined in an incremental analysis of the methods surveyed. For each method we captured its main features that were mapped onto different criteria. If a method introduced a new criterion, the rest of works were analyzed to know their assumptions with regard to this criterion. Therefore, criteria presented below were defined in an iterative process during the analysis of the multidimensional design methods.

We have summarized these criteria in three main categories: general aspects, dimensional data and factual data. A graphical representation of these features is found in Figure 2.1. Next to each criterion, the values it may take are provided (in brackets, the acronyms). For example, the values that we assign for the *paradigm* criterion are *demand-driven* (DD), *supply-driven* (SD), *interleaved hybrid* (IH) or *sequential hybrid* (SH). General aspects refer to those criteria regarding general assumptions made in the method, whereas dimensional and factual data criteria refer to how dimensional data and factual data are identified and mapped onto multidimensional concepts.

General Aspects: The general criteria are summarized into nine different items:

- Paradigm: According to our terminology introduced in Section 2.1.1, multidimensional modeling methods may be classified as supply-driven, demand-driven or hybrid approaches. The reader may found a slightly different classification in [LBMS02].

- Application: Most methods are semi-automatic. Thus, some stages of these methods must be performed manually by an expert (normally those stages aimed to identify factual data) and some others may be performed automatically (normally those aimed to identify dimensional data). In general, only a few methods fully automate the whole process. On

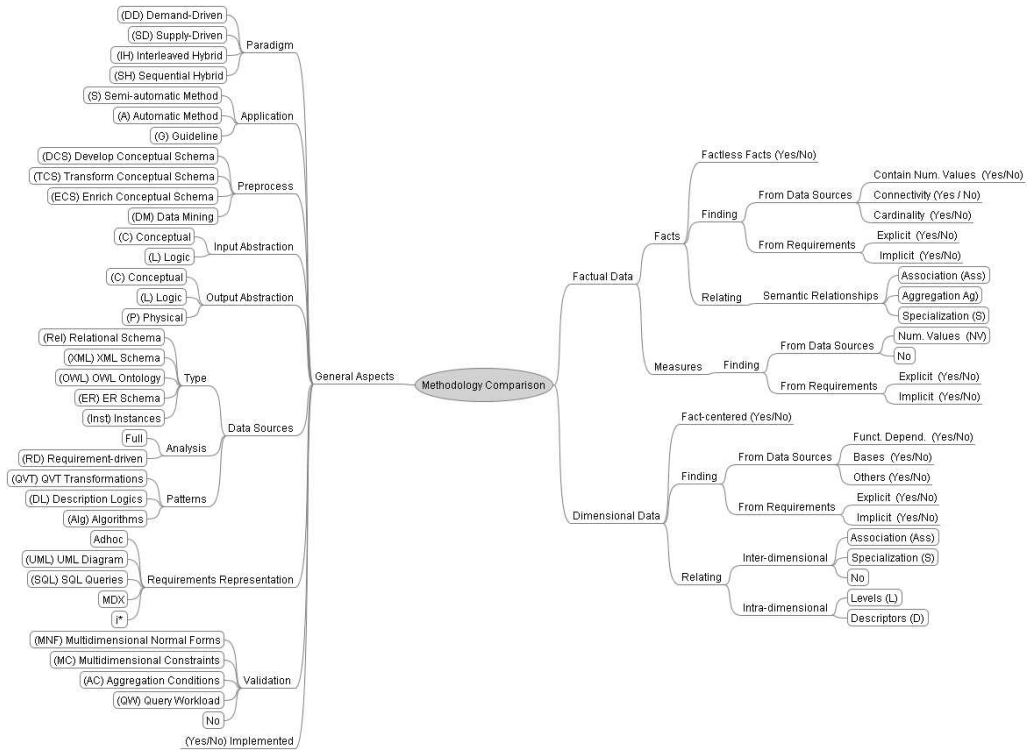


Figure 2.1: Graphical view of the criteria used for comparing the multidimensional design methods

the contrary, most methods present a detailed step-by-step guide that is assumed to be manually carried out by an expert.

- Pre-process: Some methods demand to adapt input data into a specific format that facilitates their work. For instance, these processes may ask to enrich a conceptual model with additional semantics or perform data mining over data instances to discover hidden relationships.

- Input abstraction level: Most methods (mainly those automatable) work with inputs expressed at a logical level (e.g., relational schemas) whereas some others work with inputs at a conceptual level (e.g., from conceptual formalizations such as ER diagrams or from requirements in natural language).

- Output abstraction level: Several methods choose to directly generate a star or snowflake schema, whereas some others produce multidimensional conceptual schemas. Although many approaches argue that the data warehouse method should span the three abstraction

levels, only a few of them produce the conceptual, logical and physical schema of the data warehouse.

- Data sources: There are three items summarizing main features about how data sources are considered in the method.
  - Type of data sources: The input abstraction item informs about the abstraction level of the input, whereas this item specifies the kind of technology of the data sources supported by the method. For example, if the method works at the conceptual level it may work from UML, ER conceptual schemas or OWL ontologies, and if it works at the logical level it may work from relational schemas or XML schemas.
  - Data sources analysis: Most methods perform a fully supply-driven analysis of the data sources. However, some of them also perform a requirement-driven analysis of the data sources. Clearly, this item is tightly related to the paradigm item. Nevertheless, note that a method may follow a hybrid approach but do not consider at all requirements when analyzing the data sources.
  - Pattern formalization: Supply-driven stages usually define design patterns to identify the potential multidimensional role that concepts depicted in the data sources may play. Some methods present these patterns in an informal way, but most of them use some kind of structured language. For example, ad hoc algorithms are the most common representation but some other methods use description logic formulas or QVT Transformations.

- Requirements representation: If requirements are considered, this item summarizes how requirements are represented. For example, most methods use ad hoc representations (like forms, sheets, tables or matrixes), whereas some others use UML diagrams or the *i** framework. Finally, some of them lower the level of abstraction of requirements to a logical level by means of SQL queries or MDX queries [Mic].

- Validation: Some methods integrate a validation process to derive meaningful multidimensional schemas. For example, restricting summarization of data to those dimensions and functions that preserve data semantics or forming multidimensional spaces by means of orthogonal dimensions.

- Implementation: Some methods have been implemented in CASE tools or prototypes.

Factual Data: These criteria summarize how a given method identifies and handles factual data (i.e., facts and measures). First, criteria used to identify measures are summarized as follows:

- Data sources: Up to now, looking for numerical concepts is the only heuristic introduced to identify measures from the data sources.

- Requirements: Most approaches consider requirements to identify measures. We distinguish if the method only considers explicit measures or also implicit ones. Implicit measures are those explicitly stated in the requirements but implicit in the data sources (i.e.,

there is not a concept in the data sources that would correspond to it, but they can be derived from an already existing concept(s) in the data sources). For example, derived measures. Therefore, some kind of reasoning over the data sources is needed.

Next, we introduce criteria used to identify facts. These criteria refer to how facts are identified from the data sources or from requirements, and how they may be semantically related in the resulting schema:

- Factless facts: This kind of facts were introduced by Kimball [KRTR98]. They are also known as empty facts and they are very useful to describe events and coverage and a lot of interesting questions may be asked from them.

- Data sources: Most of the methods demand to explicitly identify facts by means of the requirements, but some others use heuristics to identify them from the data sources. For example, in case of relational sources, most use heuristics such as table cardinalities and the number of numerical attributes that a table contains. Furthermore, some works also look for concepts with high to-one connectivity (i.e., with many potential dimensional concepts).

- Requirements: Similar to measures, if requirements are considered, we distinguish among explicit and implicit facts. However, implicit facts have a slightly different meaning. We denote by implicit facts those that have not been explicitly stated in the requirements but can be identified from a requirement-driven analysis of the sources.

- Semantic relationships: In case of producing a conceptual schema, some methods are able to identify semantic relationships between facts. We distinguish among associations, aggregations (also called roll-up / drill-down relationships) and generalizations. In the multidimensional model, it means that we may perform multidimensional operators such as drill-across or drill-down over them.

Dimensional Data: These criteria analyze how the method identifies and handles dimensional data (i.e., dimensions, levels and descriptors). We have two main groups of items. Those referring to how dimensional data is identified (either from the data sources or from requirements), and how they are semantically related in the resulting schema. The process to identify dimensions, levels and descriptors must be understood as a whole and unlike criteria used to identify factual data we do not distinguish among criteria to look for different dimensional concepts. Roughly speaking, most approaches start looking for concepts representing interesting perspectives of analysis and from these concepts they look for aggregation hierarchies (i.e., levels). The whole hierarchy is then identified as a dimension and level attributes are considered to play a descriptor role:

- Fact-centered: Most methods look for dimensional data once they have identified facts. From each fact, dimensional concepts are identified using a wide variety of techniques according to the method inputs, but always looking for functional dependencies from the fact.

- Data sources: There are several techniques to identify dimensional concepts from data sources. We classify these techniques in three main groups: discovering functional dependencies, discovering bases and others. At the conceptual level, functional dependencies are modeled as to-one relationships, and at the logical level it depends on the technology. For example, in the relational model, dimensional concepts are identified by means of foreign keys and candidate keys. Bases (see Section 1.5 for further information) are used to identify dimensional concepts as well. In this case, the method looks for candidate multidimensional bases in order to identify interesting perspectives of analysis (i.e., levels).

- Requirements: Dimensional concepts are mostly identified from the data sources once facts and measures have been identified. However, demand-driven approaches rely on requirements to identify dimensional concepts and some hybrid approaches also enrich their supply-driven stages with requirements. Like facts, we distinguish between explicit dimensional concepts and implicit ones.

- Intra-dimensional: Most of the methods distinguish between descriptors and levels, but some others do not.

- Inter-dimensional: Some approaches are able to identify semantic relationships between dimensions. In this case, we consider associations and generalizations as potential relationships.

2.1.4 Methods Comparison

In this section we present a detailed summarization of the main features of each method regarding the criteria introduced in previous section, which provides a common framework to compare and discuss methods surveyed. Results are shown in Tables 2.1 and 2.2[2], in which MDBE (see Chapter 3) and AMDO (see Chapter 4) are also considered. Methods surveyed are distributed in these tables according to the chronological order. There, rows correspond to criteria introduced in Section 2.1.3 and columns correspond to each method studied. A given cell contains information for a method for a certain criterion (we address the reader to Figure 2.1 to remind the meaning of each acronym). Most of the criteria are evaluated as *yes / no*, but some other have alternatives. Acronyms used to represent these alternatives may be found in Figure 2.1. Two general values can be found for any criterion: - means that this criterion does not make sense for the method (for example, if it does not consider the data sources then, any of the criteria related to them cannot be evaluated for this method), whereas *none* means that, despite this criterion could be considered for this method, none of the alternatives are considered (i.e., it is overlooked). Therefore, *none* is the equivalent to the *no* value but for criteria having several values.

Analyzing these tables we can find some interesting trends as well as assumptions that have been considered in most of the methods surveyed. First approaches tried to contextualize the multidimensional modeling task by providing tips and informal rules about how to proceed. In other words, they presented the first guidelines to support multidimensional design. Later, when

[2]Note that these tables were used to produce Figures 1.3 and 1.4 in pages 14 and 15.

	[KRTR98]	[CT98a]	[GMR98]	[BvE99]	[HLV00]	[MK00]	[BCC+01]	[PD02]
General Aspects								
Paradigm	DD	IH	SH	SH	DD	SD	SH	SH
Application	G	G	S	G	G	G	S	S
Pre-process	-	DCS	-	ECS	-	DCS	-	-
Input Abstr.	C	C	C/L	C	C	C	C/L	L
Output Abstr.	L	L	C/L/P	L	C	L	L	C
Data Sources								
↪ Type	-	ER	ER/Rel	SER	-	ER	ER	Rel
↪ Analysis	-	RD	Full	Full	-	Full	Full	Full
↪ Patterns F.	-	None	Alg	None	-	None	Alg	Alg
Req. Expr.	ad hoc	ad hoc	ad hoc	ad hoc	ad hoc	-	ad hoc	MDX
Validation	No	No	No	No	MNF	No	No	No
Tool	No	No	Yes	Yes	No	No	No	No
Factual Data								
Facts								
Factless Facts	Yes	No	No	No	No	No	No	No
Requirements	Expl	Expl	Expl	Expl	Expl	-	Expl	No
Data Sources								
↪ C.Num.Val.	-	No	No	No	-	Yes	Yes	Yes
↪ Connectivity	-	No	No	No	-	No	No	No
↪ Cardinality	-	No	No	No	-	No	No	Yes
Semantic Rels.	-	-	Ass	-	Ag	-	-	None
Measures								
Requirements	Impl	Expl	Expl	Impl	Expl	-	Expl	No
Data Sources	-	No	NV	No	-	NV	NV	NV
Dimensional Data								
Fact-centered	No	No	Yes	Yes	No	No	Yes	Yes
Requirements	Expl	Expl	Expl	Expl	Expl	-	Expl	No
Data Sources								
↪ Func. Depend.	-	No	Yes	Yes	-	Yes	Yes	Yes
↪ Bases	-	No	No	No	-	No	No	No
↪ Others	-	No	No	No	-	No	No	Yes
Related								
Interdim.	None	-	None	-	None	-	-	None
Intradim.	L/D	L/D	L/D	L	L/D	L/D	L/D	L

Table 2.1: Summary of the multidimensional design methods comparison (I)

main features with regard to multidimensional modeling were set up, new formal and powerful methods were developed. These new methods focused on formalizing and automating the process. Automation is an important feature along the whole data warehouse lifecycle and multidimensional design has not been an exception. Indeed, first methods were step-by-step guidelines, but in the course of time many semi-automatic and automatic approaches have been presented. This evolution also conditioned the type of inputs used, and logical schemas were considered instead of conceptual schemas. Nowadays, last methods introduced present a high degree of automation. Moreover, we may say that this trend also motivated a change of paradigm. At the beginning, most methods where demand-driven or, in case of being hybrid approaches, they gave much more weight to requirements than to data sources. However, eventually, data sources gained relevance. This makes sense because automation has been tightly related to focusing on data sources instead of requirements. Consequently, first methods introduced gave way to others largely automatable and mostly following a supply-driven framework. Nevertheless, today, it is

	[WS03]	[VBR03]	[JHP04]	[GRG05]	[PACW06]	[RA06]	[MTL07]	[SKD07]	[RA07a]
General Aspects									
Paradigm	DD	SH	SD	SH	DD	IH	SH	SD	SH
Application	G	S	A	S	G	A	S	A	S
Pre-process	-	TCS	DM	-	ECS	-	-	TCS	-
Input Abstr.	C	L	L	C	C	L	C/L	C	C
Output Abstr.	C	L	L	C	C/L/P	C	C	L	C
Data Sources									
↪ Type	-	XML	Inst	ER/Rel	-	Rel	Rel	Rel	OWL
↪ Analysis	-	Full	Full	RD	-	RD	RD	Full	Full
↪ Patterns F.	-	None	Alg	None	-	Alg	QVT	None	DL
Req. Expr.	ad hoc	ad hoc	-	*i**	UML	SQL	*i**	-	-
Validation	No	No	AC	No	AC	MC	MNF	No	MC
Tool	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Factual Data									
Facts									
Factless Facts	No	No	No	No	Yes	Yes	No	Yes	No
Requirements	Expl	Expl	-	Expl	Expl	Impl	Expl	-	-
Data Sources									
↪ C.Num.Val.	-	No	Yes	No	-	No	No	No	Yes
↪ Connectivity	-	No	No	No	-	No	No	Yes	Yes
↪ Cardinality	-	No	Yes	No	-	No	No	No	No
Semantic Rels.	None	-	-	None	None	Ass/S	Ass/S	None	Ass/Ag
Measures									
Requirements	Expl	Expl	-	Expl	Expl	Impl	Impl	-	-
Data Sources	-	No	NV	No	-	No	No	No	NV
Dimensional Data									
Fact-centered	No	Yes	Yes	Yes	No	No	No	Yes	Yes
Requirements	Expl	No	-	Expl	Expl	Impl	Impl	-	-
Data Sources									
↪ Func. Depend.	-	Yes	Yes	Yes	-	Yes	Yes	Yes	Yes
↪ Bases	-	No	No	No	-	No	No	No	Yes
↪ Others	-	No	No	No	-	No	No	No	No
Related									
Interdim.	None	-	-	None	None	Ass/S	S	None	Ass
Intradim.	-	L/D	L/D	L/D	L/D	L/D	L/D	L	L/D

Table 2.2: Summary of the multidimensional design methods comparison (II)

assumed that the ideal approach to design multidimensional data warehouses must be a hybrid approach. In this line, last works introduced are mainly hybrid approaches.

In these tables we can also note the evolution of how the multidimensional model has been considered. First approaches used to produce logical multidimensional schemas but later, most of them generate conceptual schemas. One reason for this situation could be that Kimball introduced multidimensional modeling at the logical level (i.e., as a specific relational implementation). With the course of time, it has been argued that it is necessary to generate schemas at a platform-independent level and in fact, the multidimensional design should span the three abstraction levels (conceptual, logical and physical) like in the relational databases field.

About the kind of data sources handled, most of the first approaches choose conceptual entity-relationships diagrams describing the data sources. ER diagrams were the most spread way to represent operational databases (the most common type of data source to populate the data warehouse) but the necessity to automate this process and the need to provide up-to-date conceptual

schemas to the data warehouse designer motivated that many methods worked over relational schemas instead of conceptual schemas. Almost every method either considers ER diagrams or relational schemas to describe the data sources. Lately, with the relevance gained by the semantic web area, some other works automating the process from XML schemas or OWL ontologies have been presented. About requirements, their representation have varied considerably. At the beginning, ad hoc representations such as forms, tables, sheets or matrixes were proposed but lately, many methods propose to formalize requirements representation with frameworks such as UML diagrams or *i**. Moreover, some works have also proposed to lower the level of abstraction of requirements to the logical level by means of SQL or MDX queries, which opens new possibilities of automation.

Finally, we can also identify a trend to validate the resulting multidimensional schema as well as the importance to provide a tool supporting the method.

About how to identify factual data, there are some trends that most approaches follow. Looking at the data sources, numerical concepts are likely to play a measure role whereas concepts containing numerical attributes or those with a high table cardinality are likely to play a fact role. First methods were mainly demand-driven but later, most of them used these heuristics to identify factual concepts within supply-driven stages. However, these heuristics do not identify facts or measures but concepts likely to play that role. Thus, requirements must be considered to filter the (vast) amount of results obtained, and in the last years requirements have gained relevance again. Capturing inter-relationships between schemas (i.e., facts) have also gained relevance lately, as they open new analysis perspectives when considering multidimensional algebras. Finally, the reader may note that although Kimball introduced the concept of factless facts from the very beginning, it has been traditionally overlooked. Lately, some methods considered them again. One of the reasons could be that it is difficult to automate the identification of facts that do not have measures.

According to our study, dimensional concepts have been traditionally identified by means of functional dependencies. From the very beginning, some methods proposed to automate the identification of aggregation hierarchies. In fact, many methods use requirements to identify factual data and later they analyze the data sources looking for functional dependencies to identify dimensional data. Maybe for this reason, the use of requirements to identify dimensional concepts has not been that relevant as to identify factual data. Another clear trend with regard to dimensional concepts is that, in general, the more automatable a method is, the more fact-centered it is. About relationships among dimensional concepts, inter-dimensional relationships (like relationships between facts) open new perspectives of analysis when considering multidimensional algebras. However, in this case they have been traditionally overlooked; even more than this kind of relationships between facts. On the contrary, intra-dimensional relationships gained more and more relevance from the very beginning. Most methods agree that distinguishing among dimensions, levels and descriptors is relevant for analysis purposes.

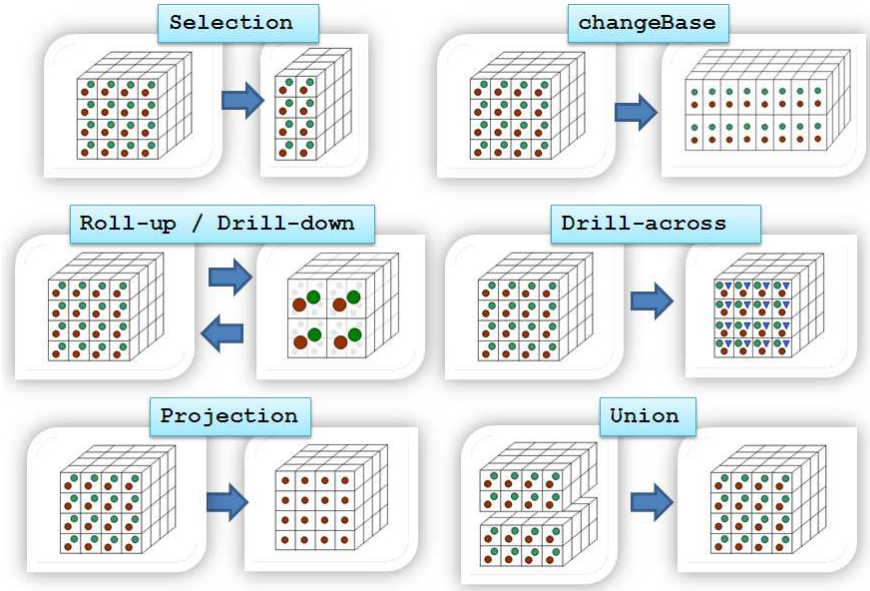


Figure 2.2: Conceptual representation of the reference multidimensional operators

2.2 Multidimensional Algebras

2.2.1 Reference Framework

Due to the lack of a standard multidimensional model, and hence, the lack of a common notation, we need a reference framework in which to translate and compare the multidimensional algebras presented in the literature. Otherwise, a comparison among the different algebras would be rather difficult. In this section we introduce a set of operators (introduced in detail in [ASS06] and conceptually sketched in Figure 2.2) that, together with the data structured introduce in Section 1.5, is used in this paper to concisely and univocally define the multidimensional concepts and operators, as well as to provide a common notation:

- **Selection**: By means of a logic clause C over a descriptor, this operation allows to choose the subset of points of interest out of the whole n-dimensional space. For example, consider Figure 1.1 (see page 5), which depicts a data cube derivable from the conceptual schema shown in Figure 1.2 (see page 9). If we are only interested on analyzing data related to `Barcelona`, we must perform a **selection** (where C would be equal to `Barcelona`) over the `place` dimension.

- **Roll-up**: It groups cells in the data cube based on an aggregation hierarchy. This operation modifies the granularity of data by means of a many-to-one relationship (corresponding to a part-whole relationship) that relates instances of two levels from the same dimension. As argued in [HS97], **drill-down** (i.e., the inverse operator of roll-up) can only be applied if we previously performed a roll-up and did not lose the correspondences between cells. For example, consider again Figures 1.1 and 1.2. If we are interested in analyzing the `sales` by `country`, we must perform a **roll-up** from `city` to `country`. Accordingly, **roll-up** will perform the necessary data aggregation of the factual instances grouped according to the `city` - `country` aggregation relationship.

- **ChangeBase**: This operation reallocates exactly the same instances of the data cube into a new n-dimensional space with exactly the same number of points, by means of a one-to-one relationship. Actually, it allows two different kinds of changes in the space base: we can just rearrange the multidimensional space by reordering the levels or, if more than one set of dimensions determine the data cube cells (i.e., if there are alternative bases), by replacing the current base with an alternative one. Consider again the example discussed previously. On the one hand, we could rearrange the dimensions to show, for example, the `place` dimension in the *y* axis and `product` in the *x* axis. On the other hand, if we had an alternative multidimensional base, for example, $\{\texttt{product} \times \texttt{year} \times \texttt{shareholder}\}$, we could replace the current base $\{\texttt{product} \times \texttt{year} \times \texttt{place}\}$ with the alternative base proposed. Note that both bases must be related by a one-to-one relationship and therefore, the multidimensional space is preserved (in our example, this alternative base would make sense if each `shareholder` controls the `sales` of just one `city`).

- **Drill-across**: This operation changes the subject of analysis of the data cube by means of a one-to-one relationship. The n-dimensional space remains exactly the same, only the cells placed on it change. For example, suppose that we dispose of a star schema analyzing `stock` data and sharing the same analysis dimensions as those depicted in Figure 1.2. In this case, we could **drill-across** from the current data cube shown in Figure 1.1 to another one with exactly the same amount of cells but, instead of just containing `sales` instances it would contain `sales` and `stock` information.

- **Projection**: It selects a subset of measures from those shown in the data cube. Following our example, we can remove the `discount` measure by just **projecting** the `price`.

- **Set Operations**: These operations allow to operate two data cubes if both are defined over the same n-dimensional space. We consider **union**, **difference** and **intersection** as the most relevant ones. For example, consider two data cubes derived from Figure 1.1 by means of **selections**: one **selecting** data concerning `Barcelona` over the `place` dimension and another **selecting** data concerning `Lleida`. These two cubes could be **united** to produce a single data cube showing factual data concerning `Barcelona` and `Lleida`.

The algebra composed by these operators is *closed* (applied to a data cube, the result is another data cube), *complete* (any correct data cube can be computed as the combination of a

finite set of these operators) and *minimal* (none can be expressed in terms of others, nor can any operation be dropped without affecting its functionality). Other operations can be derived by sequences of these operations. This is the case of **slice** (which reduces the dimensionality of the original data cube by fixing a point in a dimension) by means of **selection** and **changeBase** operations. For instance, referring to Figure 1.1, we can **slice** it by fixing the `place` dimension to a specific value (e.g., `Barcelona`) by means of a **selection**, and then change the space base to `time` $\times$ `product` through a **changeBase** without losing **cells** (note that it holds because just after the **selection** we obtain the following multidimensional base: `time` $\times$ `product` $\times$ 1).

2.2.2 The Multidimensional Algebra Vs. The Relational Algebra

In this section, we present *the proper subset of the relational algebra corresponding to our reference framework*. Our objective is twofold: (i) we place the multidimensional framework with regard to a formal, well-known and standardized algebra; and (ii) we improve the understandability of the whole picture by placing the multidimensional algebra introduced with regard to a reference framework and thus, clearly and concisely defining its semantics.

In this study, we aim to keep the comparison between both algebras at the conceptual level and avoid considering the data warehouse implementation aspects. For this reason, we assume, without loss of generality, that each multidimensional data cube (i.e., each navigation path node) is implemented as a relation in the relational database (i.e., as a *denormalized* relational table). Accordingly, the data cube of finest granularity that we may query from the multidimensional schema shown in Figure 1.2 (see page 9), would be implemented with the following relation: {`city_name`, `day`, `product_id`, `country_name`, `month`, `year`, `leap_year`, `product_descr`, `category_name`, `category_desc`, `price`, `discount`}. Where the underlined fields denote the multidimensional base and therefore, the relation primary key. In this section, we will refer to this kind of denormalized relation as the *multidimensional table*. Then, multidimensional tables contain (1) identifier fields (i.e., identifier descriptors that determine a level of detail) determining factual data, for example: `city_name`, `day` and `product_id` in the above example; (2) numerical fields, e.g., `price` and `discount`, representing multidimensional data (i.e., measures) and (3) descriptive fields, e.g., `country_name`, `month`, `year`, `leap_year`, `product_descr` and `category_name` (i.e., non-identifier descriptors).

Finally, we consider the relational algebra presented in [Cod72]. Thus, we consider "selection" (σ), "projection" (π), "union" ($\cup$), "difference" ($-$) and "natural join" ($\bowtie$) as the relational algebra operators. We talk about "natural join", or simply "join", instead of "cartesian product" (the one presented in [Cod72] and from which "join" can be derived) since the "cartesian product" without further restrictions is meaningless in the multidimensional model, as discussed in Section 2.2.3.

Table 2.3 summarizes the mapping between both sets of algebraic operators. Note that we consider the "group by" and "aggregation" as relational operators, and both will be justified consequently below. We use the following notation in the table: $\checkmark_{Measures}$ if the multidimensional operator is equivalent to the relational one but it can be only applied over relation fields representing **measures**, $\checkmark_{Descs}$ if the multidimensional operator must be applied over **descriptors** fields and finally, $\checkmark_{Descs_{id}}$ if it can be only applied over identifier **descriptors** fields. Consequently, a $\checkmark$ without restrictions means both operators are equivalent, without additional restrictions. If

the translation of a multidimensional operator combines more than one relational operator, the subscript + is added. Next, we clearly define the relational algebra *proper subset* mappable from / to the multidimensional algebra (multidimensional concepts are **bolded**, whereas relational concepts are "quoted"):

- The multidimensional **selection** operator is equivalent to a restricted relational "selection". It can only be applied over **descriptors** and then, it is equivalent to restrict the relational "selection" just over **level** data. According to our notation, we express the multidimensional **selection** in terms of the relational algebra as $\sigma_{Descriptors}$.

- Similarly, the multidimensional **projection** operator is equivalent to the relational one restricted to **measures**; that is, specific **Cell** data. In terms of the relational algebra we could express it as $\pi_{Measures}$.

- OLAP tools emphasize on flexible data grouping and efficient aggregation evaluation over groups, and it is the multidimensional **roll-up** operator the one aimed to provide us with powerful grouping and aggregation of data. In order to support it, we need to extend the relational algebra to provide grouping and aggregation mechanisms. This topic has been studied and previous works like [LW96], [Klu82] and [Lar99] have already presented extensions of the relational algebra to what is called the *grouping algebra*. All of them introduce two new operators; one to group data and apply a simple addition, counting or maximization of a collection of domain values and the other one to compute the aggregation of a given attribute over a given *nested* relation. Following the [Lar99] grouping algebra, we will refer to them as the "group by" and the "aggregation" operators. In terms of this grouping algebra, a **roll-up** operator consists of a proper "group by" operation along with an "aggregation" of data.

- **Drill-across** typically consists of a "join" between two *multidimensional tables* sharing the same multidimensional space. Notice that to "join" both tables it must be performed over their common **level** identifiers that must univocally identify each cell in the multidimensional space (i.e., over the data cube **base**). Moreover, once "joined", we must "project" out the columns in the *multidimensional table* **drill-acrossed** to, except for its **measures**. Formally, let $\mathcal{A}$ and $\mathcal{B}$ be the *multidimensional tables* implementing, respectively, the origin and the destination **Cells** involved. In the relational algebra it can be expressed as:

Reference Operator		"Selection"	"Projection"	"Join"	"Union"/"Diff."	"Group by"	"Aggregation"
Selection		✓ $Descs$					
Projection			✓ $Measures$				
Roll-up						✓ $Descs_{id}+$	✓ $Measures+$
Drill-across			✓ $Descs_{id}+$	✓ $Descs_{id}+$			
changeBase	**Add Dim.**			✓ $Descs_{id}$			
	Remove Dim.		✓ $Descs_{id}$				
	Alt. Base		✓ $Descs_{id}+$	✓ $Descs_{id}+$			
Union/Difference					✓		

Table 2.3: Comparison table between the relational and the multidimensional algebras.

$$\pi_{Descriptors_{\mathcal{A}},Measures_{\mathcal{A}},Measures_{\mathcal{B}}}(\mathcal{A} \bowtie \mathcal{B})$$

- **ChangeBase** allows us to rearrange our current multidimensional space either by changing to an alternative **base** (adding / removing a **dimension**, replacing **dimensions**) or reordering the space (i.e., "pivoting" as presented in [FBSV00]).

  When changing to an alternative **base** we must assure it does not affect the functional dependency of data with regard to the data cube **base**. Hence:

  - When adding **dimensions** we must preserve the multidimensional space. Thus, it means that the added **dimension** must be represented as a *fixed point* in the multidimensional space (i.e., it would not introduce a new *axis* in the multidimensional space). It can be achieved either by introducing the new **dimension** at the `All` **level** (note that the `All` **level** represents the whole **dimension** as one instance) or by fixing an instance, at any **level** of detail, by means of a **selection**. Therefore, in the relational algebra, adding a **dimension** is achieved through a "cartesian product" between the *multidimensional table* and the **dimension** table (that would contain a unique instance). Specifically, if $\mathcal{C}$ is the initial *multidimensional table* and $\mathcal{D}$ the relation implementing the added **dimension**, it can be expressed as:

    $$\mathcal{C} \times \mathcal{D}, \quad where\ |\mathcal{D}| = 1$$

  - On the contrary, to remove a **dimension** we need to get rid of the proper **level** identifier projecting it out in the *multidimensional table*.
  - To change the set of **dimensions** identifying each cell, i.e., choosing an alternative **base** in which to place the data, we must perform a "join" between both **bases** and project out the replaced **level descriptors** in the *multidimensional table*. In this case, the "join" must be performed through the identifier **descriptors** of **levels** replaced and **levels** introduced. Formally, let $\mathcal{A}$ be the *multidimensional table*, $\mathcal{B}$ the table showing the correspondence between both **bases** and $\mathtt{d}_1, ..., \mathtt{d}_n$ the identifier **descriptors** of those **dimensions** introduced. In the relational algebra, it is equivalent to:

    $$\pi_{Descriptors_{\mathcal{B}(\mathtt{d}_1,...,\mathtt{d}_n)},Measures_{\mathcal{A}}}(\mathcal{A} \bowtie \mathcal{B})$$

  - Finally, pivoting just asks to reorder the **levels** identifiers using the SQL "order by" operator, not mappable to the relational algebra. For this reason, it is not included in Table 2.3.

- The multidimensional **union** (**difference**) unites (differences) two data cubes defined over the same multidimensional space. In terms of the relational algebra, it is equivalent to "union" ("difference") two *multidimensional tables*.

2.2.3 A Comprehensive Survey

For the sake of comprehension, the reference operators presented in Section 2.2.1 will be **bolded** in this section, whereas the multidimensional operators introduced in each algebra appear "in quotes":

Li and Wang [LW96] introduce a multidimensional algebra as well as its translation to SQL. To do so, they introduce an ad hoc grouping algebra extending the relational one (i.e., with grouping and aggregation operators). This algebra was one of the first multidimensional algebras introduced, and the authors main aim was to construct data cubes from local operational databases.

More precisely, it defines five multidimensional operators representing mappings between either data cubes or *relations* and data cubes. The "add dimension" and "transfer" operators are aimed to rearrange the multidimensional space similar to a **changeBase**: while "Add dimension" adds a new analysis dimension to the current data cube, "transfer" *transfers* a dimension *attribute* (i.e., a descriptor) from one dimension to another via a cartesian product. Since multidimensional concepts are directly derived from non-multidimensional relations, dimensions may be vaguely defined, justifying the transfer operator; the "cube aggregation" operator performs grouping and aggregation over data, being equivalent to **roll-up** and finally, the "rc-join" operator, that allows us to join a *relational* table with a dimension of the data cube, **selects** those dimension values also present in the table. This low level operator is tightly related to the multidimensional model presented, and it is introduced to relate non-multidimensional relations with relations modeling data cubes.

Agrawal et al. [AGS97] present an algebra composed by six operators rather relevant, since they inspired many following algebras. First, "push" and "pull" transform a measure into a dimension and viceversa, as in their model measures and dimensions are handled uniformly. In our framework they would be equivalent to define semantic relationships between the proper dimensions and cells and then, **drill-across** and **changeBase** respectively; "destroy dimension" drops a cube dimension rearranging the multidimensional space and hence, being equivalent to **changeBase**, whereas the "restriction" operator is equivalent to **selection**; "merge" to **roll-up** and "join" to an unrestricted **drill-across**. Consequently, the latter can even be performed without common dimensions between two data cubes, giving rise to a cartesian product. However, a cartesian product does not make any multidimensional sense if it is not restricted, since it would not preserve *disjointness* when aggregating data ([RA05]). Finally, note that we can **project** data by means of "pull"ing the measure into a dimension and performing a "destroy dimension" over it.

Gyssens and Lakshmanan [GL97] present an algebra based on the classical relational algebra operations. Therefore, it includes "selection", "projection", "union" / "intersection" / "difference" and the "cartesian product"; all of them being equivalent to their analogous operators in our reference algebra, except for the latter which is mappable to an unrestricted **drill-across** as discussed in the previous algebra. The "fold" and "unfold" operators add / remove a **dimension**, like in a **changeBase**; whereas **roll-up** is decomposed in two operators: "classification of tables" (i.e., grouping of data) and "summarization of tables" (aggregation of data). Hence, this algebra proposes to differentiate *grouping* (i.e., the conceptual navigation between levels through a part-whole relationship or in other words, the result of mapping data into groups) from *aggregation* (i.e., aggregating data according to an aggregation function).

Thomas and Datta [TD97] and [TD01] present an algebra with eight operators based on

[AGS97]. Therefore, the "restriction" operator is equivalent to **selection**; the "metric projection" to **projection**; the "aggregation" to **roll-up** and the "union" / "difference" operators to those with the same name in our reference algebra. Moreover, similar to [AGS97], measures can be transformed into dimensions and viceversa. Hence, the "force" and "extract" operators are equivalent to the "push" and "pull" ones. Finally, they rename the "join" operator in [AGS97] as "cubic product", and denote by "join" an specific "cubic product" over two data cubes with common dimensions (i.e., preserving disjointness if joined through their shared dimensions) since, in general, a cartesian product does not make multidimensional sense.

Lehner [Leh98] present an algebra composed by five operators. "Roll-up" and "drill-down" and the "split" and "merge" operators are equivalent to **roll-up** and **drill-down**. According to its model data structure that differentiates two analysis phases of data, these four operations are needed because "roll-up" and "drill-down" find and interesting context in a first phase, whereas "split" and "merge" modify the data granularity *dynamically* by the dimensional attributes (i.e., descriptors) defined in the "classification hierarchies" nodes of the data structure. It also introduces two operators to aggregate data: the "implicit" and the "explicit" aggregation. The first one is *implicitly* used when navigating by means of "roll-up"s, whereas the second one can be *explicitly* stated by the end-user. Since they are equivalent, these operators are just differentiated because of the conceptual presentation followed in the paper. Finally, "slicing" operator reduces the multidimensional space in the same sense as **selection**, whereas the "cell-oriented operator" derives new data preserving the same multidimensional space by means of "unary operators" (-, *abs* and *sign*) or "binary operators" (*, +, -, /, *min* and *max*). "Binary operators" ask for two multidimensional objects aligned (i.e., over the same multidimensional space). In our framework it is obtained defining derived measures in design time.

Cabibbo and Torlone [CT98b], [CT97] and [CT98a] present an algebra with nine operators where, similar to [GL97], **roll-up** is decomposed into "roll-up" (i.e., grouping) and "aggregation". "Level description" is equivalent to **changeBase**: it changes a **level** by another one related through a one-to-one relation to it. In our framework we should define a semantic relationship among **levels** involved and perform a **changeBase**; "simple projection" projects out selected measures and reduces the multidimensional space by dropping dimensions: it can just drop measures (equivalent to **projection**), dimensions (to **changeBase**) or combine both. Finally, "abstraction" is equivalent to the "pull" operator in [AGS97] and "selection", "cartesian product" and "natural join" to those discussed along this section.

Hacid and Sattler [HS98] present an algebra based on description logics (DL) and developed from [AGS97]. Therefore, it also introduces "restrict", "destroy" (equivalent to "destroy Dimension") and "aggr" (equivalent to "merge"). Furthermore, the "join" and "Join" operators can be considered an extension of the "join" operator in [AGS97]: both operators restrict the original "join" to make multidimensional sense and consequently, being equivalent to **drill-across**; although the second one also allows to group and aggregate data before showing it (i.e., being equivalent to **drill-across** and **roll-up**).

Pedersen [Ped00] presents an algebra where "selection", "projection", "union" / "difference" and **roll-up** and **drill-down** are equivalent to those with the same name presented in our framework, whereas the "value-based join" is equivalent to **drill-across** and the "identity-based join" to "cartesian product". Moreover, it also differentiates the "aggregate operation" (i.e., grouping) from the "roll-up"; the "duplicate removal" operator is aimed to remove **cells** characterized by the same combination of dimensional values. In our framework it can never happen because of the base definition introduced. Finally, it presents a set of non-atomic operators; the "star-join" operator combines a **selection** with a **roll-up**, by the same aggregation function, over a set of dimensions, and the "SQL-like aggregation" applies the "aggregate operation" to a certain dimensions and projects out the rest (that is, performs a **changeBase**).

Vassiliadis [Vas00] presents an algebra with three operators. "Navigation" allows us to **roll-up**, and according to [Vas98], it is performed by means of "level-climbing" (reducing the granularity of data), "packing" (grouping data) and "function application" (aggregating by an aggregation function). Finally, "split a measure" is equivalent to **projection** and "selection" to the reference **selection**.

Yin and Pedersen [YP04] present an algebra over an XML and OLAP federation: "selection cube" allows us to **select** data; "decoration" adds new dimensions to the data cube (i.e., mappable to a **changeBase**) and "federation generalized projection" (FGP) **roll-ups** the data cube and removes unspecified dimensions (**changeBase**) and measures (**projection**). Note that although **Roll-up** is mandatory, FGP can combine it with a **projection** or/and **changeBase**.

Franconi and Kamble [FK04] present an algebra with four operations. "Slice" and "multi-slice" **select** a single or a range of dimensional values; "union" / "intersection" / "difference" combine two aligned data cubes, whereas "join" is rather close to **drill-across** but in a more restrictive way, forcing both data cubes to share the same multidimensional space. "Derived measures" derives new measures from already existent. In our framework, as already said, derived measures should be defined in design time. Finally, notice that **roll-up** is not included in their set of operators, since it is considered in their model data structure.

Finally, to conclude our survey, we would like to remark that some of these approaches have also presented an equivalent calculus besides the algebra introduced above (like [GL97] and [CT98b]). Moreover, [GMR98] presents a query language to define the expected workload for the data warehouse. We have not included the latter in Table 2.4 because it can not be smoothly compared to the algebraic operators. Anyway, analyzing it, we can deduce that many of our reference operators are also supported by their model like **selection**, **projection**, **roll-up**, **union** and even a partial **drill-across**, as they allow to overlap *fact schemes*.

2.2.4 Algebras Comparison

This section presents a comparison between the multidimensional algebras surveyed in the previous section. To the best of our knowledge, it is the first comparison of multidimensional algebras

Algebra	Operator	Selection	Projection	Roll-up Drill-down	changeBase	Drill-across	Union Difference Intersection	Remarks
[LW96]	"Add Dimension"				✓ p			
	"Transfer"				$\sim$			
	"Cube Aggr."			✓				
	"Rc-join"	✓						
	"Union"						✓	
[AGS97]	"Push"					✓ p		Semantic Rels.
	"Pull"		$\mathcal{D}$		✓ p			Semantic Rels.
	"Destroy Dimension"		$\mathcal{D}$		✓ p			
	"Restriction"	✓						
	"Join"					✓		
	"Merge"			✓				
[GL97]	"Selection"	✓						
	"Projection"		✓					
	"Cartesian Product"					$\sim$		
	"Union/Diff./Inters."						✓	
	"Fold/Unfold"				✓ p			
	"Classification"			$\mathcal{D}$				
	"Summarization"			$\mathcal{D}$				
[TD97]	"Restriction"	✓						
	"Metric Projection"		✓					
	"Aggregation"			✓				
	"Cartesian Product"					$\sim$		
	"Join"					✓		
	"Union/Diff."						✓	
	"Extract"				✓ p			Semantic Rels.
	"Force"					✓ p		Semantic Rels.
[Leh98]	"Slicing"	✓						
	"Roll-up/Drill-down"			✓				
	"Split/Merge"			$\sim$				
	"Implicit/Explicit Aggr."			✓ p				
	"Cell Operators"							Derived Measures
[CT98b]	"Cartesian Product"					$\sim$		
	"Natural Join"					✓		
	"Roll-up"			$\mathcal{D}$				
	"Aggregation"			$\mathcal{D}$				
	"Level Description"				✓ p			Semantic Rels.
	"Scalar Function App."							Derived Measures
	"Selection"	✓						
	"Simple Projection"		✓		✓ p			
	"Abstraction"		✓ +		✓ p +			
[HS98]	"Restrict"	✓						
	"Destroy"				✓ p			
	"join"					✓		
	"Join"			✓ +		✓ +		
	"Aggr"			✓				
[Ped00]	"Selection"	✓						
	"Projection"		✓					
	"Union/Diff."						✓	
	"Identity-based Join"					$\sim$		
	"Aggregate Formation"			✓ p				
	"Value-based Join"					✓		
	"Duplicate Removal"							**Base** definition
	"SQL-like Aggr."				✓ p			
	"Star-join"	✓ +		✓ +				
	"Roll-up/Drill-down"			✓				
[Vas00]	"Navigate"			✓				
	"Selection"	✓						
	"Split Measure"		✓					
[FK04]	"Derived Measures"							Derived Measures
	"Join"					✓ p		
	"Slice/Multislice"	✓						
	"Union/Diff./Inters."						✓	
[YP04]	"Selection Cube"	✓						
	"Decoration"				✓ p			
	"Fed. Gen. Projection"		✓ +	✓ +	✓ +			

Table 2.4: Summary of the comparison between multidimensional algebras.

carried out. In [VS99], a survey describing the multidimensional algebras in the literature is presented. Regarding this previous work, in this study, we include up-to-date references and provide a detailed comparison of the algebras.

Results presented along this section are summarized in Table 2.4. There, rows, representing an algebraic operator, are grouped according to which algebra they belong to (also ordered chronologically), whereas columns represent multidimensional algebraic operators in our framework (note that roll-up and drill-down are considered together since one is the inverse of the other). The notation used is the following: a ✓ cell means that those operations represent the same conceptual operator; a $\sim$ stands for operations with similar purpose but different proceeding making them slightly different; a $\checkmark_p$ means that the operation partially performs the same data manipulation as the reference algebra operator despite the latter also embraces other functionalities, and a $\checkmark_+$ means that this operation is equal to combine the marked operators of our reference algebra, meaning it is not an *atomic operator*. Analogously, there are some reference operators that can be mapped to another algebra combining more than one of its operators. This case is showed in the table with a $\mathcal{D}$ (from *derived*). Note that this last mark must be read vertically unlike the rest of marks. For example, in [AGS97], we can **project** data by means of the "pull" and "destroy dimension" operators. Finally, note that we have only considered those operations manipulating data and therefore, those aimed to manipulate the data structure are not include.

A detailed analysis of Table 2.4 draws interesting conclusions. In short, we are able to identify the *multidimensional backbone* shared by all the algebras. Firstly, **selection**, **roll-up** and **drill-down** operators are considered in every algebra. It is quite reasonable since **roll-up** is the main multidimensional operator and **selection** is a basic one, allowing to select a subset of multidimensional points of interest out of the whole n-dimensional space. **Projection**, **drill-across** and **set operations** are included in most of the algebras. In fact, along the time, just two of the first algebras presented did not include **projection** and **drill-across**. We may include **set operations** in our algebra depending on the transformations that the model allows to perform over data and indeed, it is a personal decision to make. However, we do believe that to unite, intersect or difference two data cubes is a kind of navigation desirable. Finally, **changeBase** is also partially considered in most of the algebras. Specifically, they agree on the necessity of modifying the n-multidimensional space by adding / removing dimensions, and they include it as a first-class operator. Moreover, our framework provides additional alternatives to rearrange the multidimensional space (i.e., to change the multidimensional space base by "pivoting"). In general, we can always rearrange the multidimensional space in any way, if we preserve the functional dependencies of the cells with regard to the levels conforming the multidimensional space base; i.e., if the replaced dimension(s) and the new one(s) are related through a one-to-one relationship. Importantly, according to this study, all the algebras surveyed are subsumed by our reference framework.

Finally, the algebra comparison presented in this section has revealed many implicit agreements about how multidimensional data should be handled. Although this is not the aim of this thesis, we strongly believe that a reference set of operators such as the *multidimensional backbone* identified in our study could be used to develop design methods oriented to improve querying, develop better and more accurate indexing techniques and facilitate query optimization (i.e., provide us with all the benefits of a reference framework). Experiences in the field of

databases have proved that a common framework to work with is crucial for the evolution of the area, and issues such as query optimization or better indexing techniques are even more critical than in an operational database, due to the huge amount of data stored in the data warehouse.

Chapter 3

Integrating Requirements in a Largely Automated Design Approach

" Research is the act of going up alleys to see if they are blind. "

Plutarch

Data warehousing systems were designed to support decision-making within organizations. These systems homogenize and integrate data in a huge repository (i.e., the *data warehouse*) to create a single, detailed representation of the organization from which relevant knowledge can be extracted and applied in the organization's decision-making processes. It is widely accepted that the conceptual schema of a data warehouse must be structured according to the multidimensional model. The multidimensional conceptual view of data is distinguished by the *fact / dimension* dichotomy and represents data as if placed in an n-dimensional space, which facilitates the interpretation and analysis of data in terms of *facts* (the subjects of analysis) and *dimensions* showing the different perspectives from which a subject can be analyzed.

Since a data warehouse is the result of homogenizing and integrating relevant data in a single, detailed view, it is assumed that the multidimensional conceptual schema of the data warehouse must be derived from the organization's data source schemas. Traditionally, this process has been performed manually, but automation is essential as it removes the dependency on an expert's ability to properly apply the method chosen and the need to analyze the data sources, which is a tedious and time-consuming task (which can be unfeasible when working with large databases). In recent years, several approaches have been proposed for automating this process, most of which follow a data-driven model in which data sources are analyzed thoroughly to derive the data warehouse schema in a reengineering process that overlooks the end-user mul-

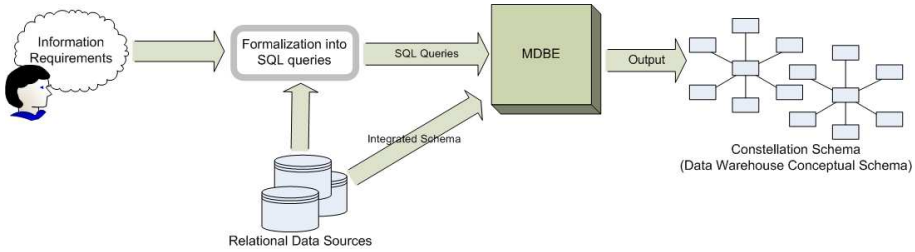


Figure 3.1: Overview of the MDBE method

tidimensional requirements. However, as discussed in [WS03], a requirement analysis phase is crucial in ensuring that end-user needs and expectations are met. Otherwise, end-users may become frustrated as they would not be able to analyze data of interest to them, which would result in the failure of the whole system. The literature contains several requirement-driven methods, but all of them must be carried out manually. Automating requirement-driven approaches would require the formalization of end-user requirements (i.e., translating them into a computer understandable language), whereas current methods handle requirements that are mainly stated in languages (such as natural language) which lack the required degree of formalization.

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

In this chapter we present a largely automated approach for supporting multidimensional design based principally on *Multidimensional Design By Examples* (MDBE), which is an automated method conciliating both types of paradigms. Unlike other hybrid approaches, MDBE does not carry out two well-differentiated phases (i.e., data-driven and requirement-driven) that need to be conciliated a posteriori but instead performs both phases simultaneously. Consequently, each paradigm benefits from feedback obtained by the other, and eventually MDBE is able to derive more valuable information than approaches in which the two phases are carried out sequentially (a detailed list of the main advantages of MDBE over previous approaches is given in Section 3.1).

In our approach we derive multidimensional *conceptual* schemas from *relational sources* according to end-user *requirements*. There are two steps: requirement formalization and the MDBE method (see Figure 3.1). As in previous requirement-driven methods (or requirement-driven stages within hybrid methods), a prior requirements elicitation step is required. However, our approach is not based on a step-by-step manual process in which the requirements and data sources that will eventually derive the multidimensional schema are analyzed in detail, but rather

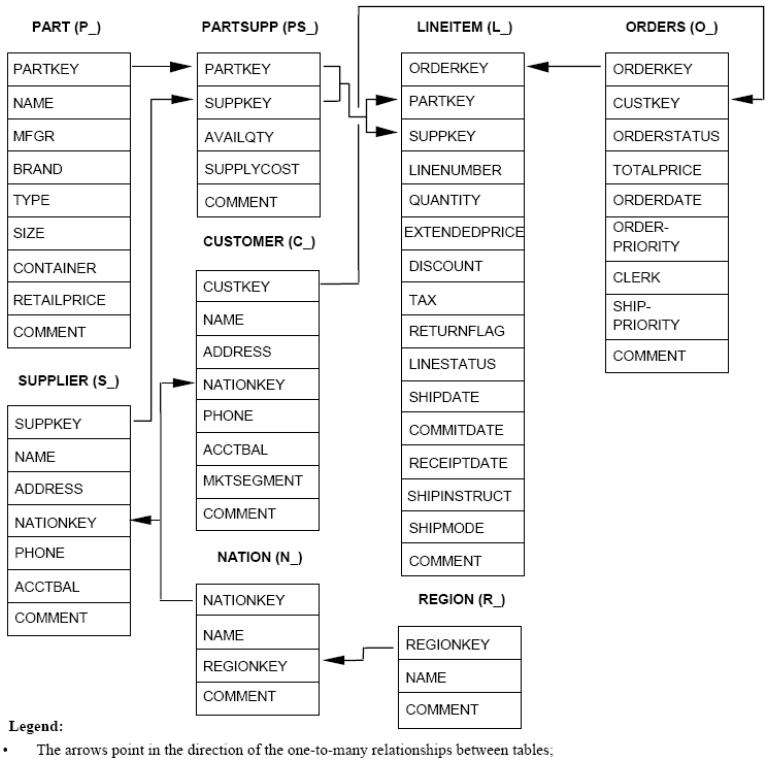


Figure 3.2: The TPC-H relational schema

on a largely automatable approach.

Requirements are typically expressed at a high level of abstraction and need to be formalized prior to automation of the analysis step. In our framework, requirements are expressed as SQL queries over the relational data sources (i.e., at the logical level over the data sources). SQL queries provide a clearly defined structure that will facilitate full automation of the MDBE method (the second step in our approach). Although requirement formalization must be performed manually, translating requirements into SQL queries requires considerably less effort than carrying out any of the step-by-step requirement-driven approaches in current use (see Section 3.1 for further discussion of this issue). In our approach we have reduced the amount of manual operations as much as possible (i.e., removing ambiguous semantics by formalizing the requirements) and delegated most of the design workload to the MDBE method, which will use the semantics captured in the requirements and the data sources to automate the rest of the process.

The inputs of the MDBE method are the end-user information requirements (expressed as SQL queries) and the *integrated* logical model of the data sources. The output is a *constella-*

tion schema [KRTR98] (i.e., a conceptual schema for each fact identified) derived from the data sources and capable of retrieving data requested in the input requirements. Briefly, MDBE validates whether each input SQL query represents a valid multidimensional query (i.e., if the query retrieves data that can be analyzed from a multidimensional perspective). Note that we translate requirements into regular SQL queries over the transactional data sources and do not require a specific translation that would make multidimensional sense. MDBE analyzes each input SQL query to validate whether it represents a multidimensional requirement and notifies if it is able to derive at least one *multidimensional schema* that can retrieve data requested in the SQL query. Conciliation of the schemas proposed for each query produces the output constellation schema.

To illustrate a practical application of our approach we now introduce the TPC benchmark H (TPC-H) [Tra09]. TPC-H is a decision support benchmark that introduces a relational database logical schema (see Figure 3.2) and a suite of 22 business-oriented queries. This benchmark was designed to represent a real-world information system, so its database schema and queries have been chosen for their industry-wide relevance. The database schema presented portrays the activity of a wholesale supplier. TPC-H does not represent the activity of a particular business sector but rather that of any industry in which a product needs to be managed, sold or distributed internationally (e.g., car rental, food distribution, parts, suppliers, etc.). Queries presented in the benchmark have been given a realistic context and were chosen to be representative and to answer to real-world questions. The queries are defined by the following components:

- A high-level description of the business question, which illustrates the context in which the query could be used. For example, *"report the amount of business that was billed, shipped, and returned"* (Q1), *"list the revenue volume done through local suppliers"* (Q5), *"determine the value of goods shipped between certain nations to help in the re-negotiation of shipping contracts"* (Q7) or *"identify customers who might be having problems with the parts that are shipped to them"* (Q10).

- The functional query definition, which uses the SQL-92 standard to define the function to be performed by the query. As an example, business query #5 (Q5) is expressed in SQL as:

```
SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
FROM customer, orders, lineitem, supplier, nation, region
WHERE c_custkey = o_custkey and l_orderkey = o_orderkey and
l_suppkey = s_suppkey and c_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;
```

TPC-H is a decision support benchmark, and it would make sense to propose a multidimensional schema (i.e., develop a data warehouse) for analyzing this data. This has already been considered in data warehouse research. One example is the *Star Schema benchmark* (SSB) [P. 09], which was devised from the TPC-H benchmark and introduces a multidimensional schema derived manually from the TPC-H relational schema. We use TPC-H to demonstrate how to derive the multidimensional schema with our approach and then, we confirm the reliability of the result

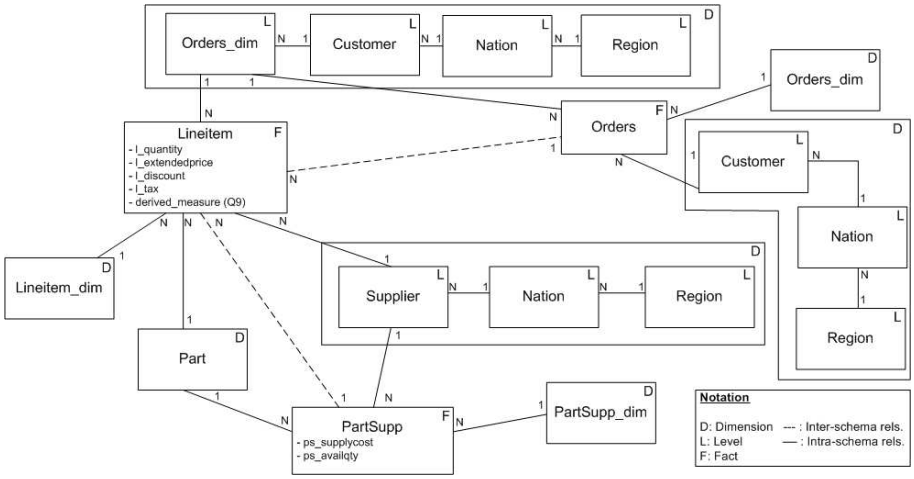


Figure 3.3: Constellation schema derived by MDBE from the TPC-H benchmark case study

obtained by comparing it to that of the multidimensional schema proposed in SSB (see Section 3.5.5.1).

We consider the TPC-H relational schema as the integrated relational schema of the data sources, and the high-level descriptions of the business queries as the end-user *information requirements* (gathered in the requirements elicitation step). TPC-H also provides the SQL query for each end-user requirement (i.e., the requirement formalization required in our approach). Consequently, it provides all of the inputs needed to launch the MDBE method. Eventually, MDBE will generate a set of multidimensional schemas (see Figure 3.3) from the data sources (in this case, the TPC-H logical schema) that meet the end-user requirements (the 22 TPC-H SQL queries).

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

This chapter provides a detailed description of our approach. In Section 3.1 we highlight the main advantages of MDBE over alternative approaches. In Section 3.3 we present our framework and discuss the foundations of our approach in depth. Section 3.4 focuses on the the core of our approach, the MDBE method. Finally, in Section 3.5 we present the statistics of our approach over the TPC-H case study and compare the results with those presented in the Star Schema

Benchmark.

3.1 Contributions

For the sake of understandability, we present two discussions introducing our contributions. The first one focus on the MDBE contributions regarding demand-driven approaches[1],whereas the second one focus on the contributions regarding automated supply-driven approaches (for a detailed related work, we address the reader to Section 2.1).

3.1.1 Demand-driven approaches

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

- MDBE, like current automatable methods, works exclusively with relational data sources. This means that the organization has a relational transactional system, which makes it very likely that a number of employees will possess the skills required to create the queries; the database administrator, for example, would have a perfectly suited profile. On this point, we make the same assumptions as used in current automatable approaches.

- Although our approach follows a hybrid paradigm, most of the tasks are automated. Briefly, this means that the user is not required to *deeply analyze* the sources. This contrasts with current demand-driven approaches, which require detailed and exhaustive analysis of the data sources; hints and tips are given (in case of the most formal methods, multidimensional patterns) that the data warehouse expert must then manually search for across all data sources (logical or conceptual schemas, depending on the approach chosen). The time-consuming nature of this task can render it unfeasible when large databases are used.

  For example, [MK00] introduces a commonly cited method for deriving the multidimensional schema from an Entity-Relationship [Che76] (ER) schema. It requires each ER entity to be classified as a *transactional* (the basis for fact tables), *component* (details or components of business events that will produce dimensions) or *classification* (that will be used to shape the dimension hierarchies) entity. The authors give advice on how these entities can be identified. Thus, *"transactional entities must describe events that happen at a point in time and contain measures or quantities that can be summarized"*.

[1] Although many times we just refer to demand-driven or supply-driven approaches, we, indeed, also refer to any approach or stage within a hybrid approach following that paradigm.

Formal rules are given for each type of entity to give shape to the multidimensional schema. For example, a typical rule is discovering *functional dependencies* (FDs) to identify dimensional data. Manual discovery of FDs is an unfeasible task for most systems [RCARM09, DKM08, TZ04], and automatic methods for identifying FDs need to address this task at the instance level (i.e., using the instance semantics). These methods have various drawbacks, propose solutions that are computationally expensive, and register drops in performance when a large number of attributes or instances are processed [HCTJ93, MR92, SBHR06, TZ04].

Other important demand-driven approaches, such as [HLV00], use a formal framework to derive the multidimensional schema. However, little information is given about how to identify the multidimensional concepts over the data sources. This scenario is repeated in the most recent demand-driven approaches; for example, [PACW06] derives the multidimensional conceptual, logical and physical (using the Oracle MOLAP Tool) with a UML [Grob] based method that introduces a metamodel and a set of transformations to perform the mapping between each metamodel. However, this approach suffers from the same drawback as previous approaches, and analysis of the sources may still be unfeasible if it has to be performed manually. More comprehensive descriptions of these methods and how they work can be found in Section 2.1.2.

In contrast, our approach introduces a manual formalization step but the user does not need to analyze the data sources or perform the complete mapping of the requirements over the data sources. Importantly, the exhaustive analysis and iterative application of the multidimensional patterns are delegated to the MDBE method. In other words, the counterpart to our method would require the entire automated process described in Section 3.4 to be carried out manually (by properly applying the multidimensional patterns introduced in Sections 3.3.1 and 3.3.2), which is the case of current demand-driven methods. Therefore, the manual workload in our proposal is considerably lighter than in previous approaches.

3.1.2 Automatable approaches

The MDBE method was designed to overcome the limitations shared by current automatable methods. To our knowledge, (i) MDBE is the first method with an automated demand-driven stage. Our approach requires end-user requirements to be formalized as SQL queries, after which MDBE validates each SQL query to determine whether it makes multidimensional sense (see Section 3.3.1 for further information). The main contribution in this area is that (ii) MDBE validates the explicit and implicit multidimensional knowledge in the query. For example, relationships between concepts depict the potential multidimensional role that each concept could play, and *joins* stated in the WHERE clause identify relationships (i.e., *concept associations*) explicitly stated by the user that, in some cases, may not be in the logical schema of the data sources. For example, consider a database overlooking foreign keys. In these cases, previous approaches that rely on primary key - foreign key relationships would overlook this information. In contrast, in our approach, these missing relationships will be stated (if they are of relevance for the user) by means of concept associations (i.e., joins) in the SQL query. Thus, if a join attribute is identified as dimensional data, this multidimensional role is propagated to its join counterpart

(like a supply-driven approach would do if the proper primary key - foreign key relationship were defined). Another example would be a denormalized database. In this case, if the query performs data grouping (i.e., it contains a GROUP BY clause) or contains comparison clauses in the WHERE clause, the attributes involved in these clauses are identified as dimensional data. In other words, the SQL queries may provide additional relevant knowledge to that captured in the sources. Nevertheless, we also harness the knowledge contained in the data sources (as in supply-driven approaches), such as foreign key and candidate key constraints, if present. In addition, (iii) MDBE works at the *attribute level* (SQL queries handle attributes), whereas other automatable methods work at the table level. Consequently, relational attributes can be labeled as dimensional or factual data and, in turn, relational tables are identified as dimensional data, factual data or tables containing factual data and dimensional data [KRTR98]. We can therefore identify the role played by each attribute in each relation and split it into different concepts in the resulting multidimensional schema. Thanks to these contributions, (iv) MDBE is able to handle denormalized relational schemas to some extent. The analysis of requirements at the attribute level allows MDBE to identify dimensional or factual attributes that previous approaches would overlook. However, regarding dimensional data identified from denormalized relations, MDBE cannot automatically generate the dimension hierarchies as the domain FDs needed to shape hierarchies are missing in the source schema. In other words, each requirement (i.e., SQL query) will identify attributes representing interesting analysis perspectives, but the relationships between these attributes (i.e., the dimension hierarchies) cannot be extracted from denormalized data sources. In these cases, the designer will be responsible for restructuring this kind of dimensional data.

MDBE also provides the advantage of carrying out the demand-driven and supply-driven stages simultaneously in many aspects. This means that we are able to produce more and better-quality outputs than methods in which the two stages are performed sequentially. For example, (v) MDBE can derive implicit knowledge according to the input query and the data sources. Some attributes in the query may not play a relevant role in the output produced, in which case they could be overlooked. However, we analyze all of the potential alternatives, as well as metadata in the logical schema, and consider how these alternatives would affect the output schema, in some cases deriving interesting alternatives overlooked by the user. This contribution is important because it is often assumed in data warehouse modeling that the user may not recognize the analytical potential of all the data sources and, therefore, may overlook potentially useful analytical alternatives. However, analyzing all of the data sources can be expensive and produce too much noise in the final result [WS03]. We present an intermediate solution, in which concepts are analyzed to determine their analytical potential if they are implicitly related to concepts already stated in the end-user requirements (see step 6 in section 3.4.1 for further details).

In addition, (vi) MDBE can derive new concepts that are not stated in the logical schemas. Since we handle requirements automatically, we can analyze them in depth and identify information such as concept specializations or newly derived measures (see Section 3.3.1.1 for further details). (vii) MDBE also keeps track of relevant metadata extracted from the requirements, which will be relevant in the implementation stage: specifically, interesting data granularity within a fact (see Step 2 in Section 3.4.1) and data summarizability properties (see Steps 1 and 3 in Section 3.4.1).

3.2 Validating SQL Queries as *Cube-Queries*

As discussed in Chapter 2 there is no agreement on the multidimensional model integrity constraints nor in the set of multidimensional operators. However, if we aim to automate the data warehouse design task, we must guarantee that the conceptual schemas produced are aligned with the multidimensional model. Section 2.2 surveyed and compared current multidimensional algebras in the literature. By a detailed analysis of this comparison, we shown that there is an implicit agreement on how to manipulate multidimensional data. As presented in Section 2.2.4, this *backbone* is strictly subsumed by the reference algebra introduced in Section 2.2.1.

This chapter introduction sketches the idea behind our approach. The end-user requirements must be expressed as *SQL queries* over the *relational sources*. Then, MDBE validates whether each input SQL query represents a valid multidimensional query (i.e., if the query retrieves data that can be analyzed from a multidimensional perspective) and eventually, derives multidimensional schemas from the relational sources that meet the multidimensional requirements. At this point, the question is immediate; *how do we know if a SQL does really make multidimensional sense?*

[KRTR98] introduced the template query (also known as *cube-query*), to retrieve a **Cell** of data from the relational database management system (according to the SQL'92 standard):

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

The FROM clause contains the "Cell table" and the "level tables". These tables are properly linked in the WHERE clause. Additionally, the WHERE clause can also contain logic clauses restricting an specific level attribute (i.e., a descriptor) to a constant $\mathcal{K}$ by means of a comparison operator (i.e., equality, inequality, major, minor, etc.). The GROUP BY clause shows the identifiers of the levels used to aggregate data. Those columns in the grouping must also be selected in the SELECT clause in order to identify the result (i.e., we must select the multidimensional base to give rise to the multidimensional space). Finally, the ORDER BY clause sorts the output of the query by these identifiers.

Note, however, that navigating and analyzing the data warehouse goes far beyond than just retrieving a Cell of data. In a navigation path, Cells may be combined and, in general, manipulated, by the multidimensional algebra. Thus, *how this template would look like when capturing a whole navigation path?* and importantly, *will it always be correct?* To answer these questions, we carried out the following studies:

- Section 2.2.4 shows that our reference algebra subsumes all the multidimensional operators surveyed. From this starting point, we studied how each of the multidimensional operators in the reference framework should be translated into SQL (see Section 3.2.1).

- Next, we analyze the potential problems we must deal with when combining two or more multidimensional operators in the same cube-query (see Section 3.2.2).

By the analysis of the results got in these two studies, we identify the constraints that a SQL query must guarantee to be aligned with the multidimensional model and make multidimensional sense.

3.2.1 Translating the Multidimensional Operators into SQL Queries

MDBE requires to express the end-user requirements as SQL queries over the relational sources. Thus, in our study, we need to analyze how the multidimensional algebra must be translated into SQL. Importantly, note that this translation is also *implicitly* performed by ROLAP tools (see Section 1.3.1 for further details about ROLAP tools). As discussed in Section 1.2.1, OLAP users are able to navigate (i.e., query and analyze) data in real-time. The user provides a *navigation path* in which each node (resulting in a data cube) is derived from the previous node in the path (and thus we say that the user *navigates* the data). Each node is transformed into the next one in the path by applying specific multidimensional operators. In a relational implementation of the OLAP tool (i.e., in a ROLAP tool), the navigation path is eventually translated (in a transparent way to the user) into SQL. Interestingly, to know if a SQL query makes multidimensional sense we need to analyze how a ROLAP tool translates the multidimensional operators into SQL and identify which constraints must satisfy a SQL query to be a cube-query (i.e., to make multidimensional sense).

In this section, we first analyze how each multidimensional operator in our reference multidimensional algebra (see Section 2.2.1) is expressed as a cube-query. First, for the sake of understandability, we present a practical scenario to be used as example in this section. Consider a snowflake implementation of the conceptual schema depicted in Figure 1.2 (see page 9). The cube-query that would retrieve the `sales` Cell depicted in the figure is:

```
SELECT d.day, p.id, c.name, s.price, s.discount
FROM sales s, day d, product p, city c
WHERE s.product_id = p.id AND s.day = d.day
AND s.city_name = c.name
```

Note that no grouping is needed as we are just retrieving an atomic Cell. Accordingly, we use *atomic cube-query* to denote a cube-query retrieving a materialized Cell from the relational database management system. Next, we show how this cube-query would be modified by each multidimensional operator[2] (a summarization of the results obtained is shown in Table 3.1):

- **Selection**: In SQL, it means to *and* the corresponding comparison clause to the WHERE clause. For example, consider the atomic cube-query presented as example. If we want to analyze the `sales` data regarding to the `city` of `Barcelona`, we must perform a **selection** over the `city` dimension (see Figure 3.4).

- **Roll-up**: In SQL, it entails to replace the identifiers of the level from where we **roll-up** with those of the level that we **roll-up** to. Thus, the SELECT, GROUP BY and ORDER BY clauses must be modified accordingly. Measures in the SELECT clause must also

[2]For a detailed discussion on this issue, we address the reader to [ASS03].

Clause	**Selection**	**Roll-up**	**ChangeBase**	**Drill-across**	**Projection**	**Union**	
SELECT		Replace (LevelID)	Replace (LevelID)	Add (Measure)	Remove (Measure)		
FROM			Add (Levels)	Add (Cell)		Union (Cells and Levels)	
WHERE	AND (conditions)		Add (links)	Add (links)		Union (links)	OR (conditions)
GROUP BY		Replace (LevelID)	Replace (LevelID)				
ORDER BY		Replace (LevelID)	Replace (LevelID)				

Table 3.1: Summary of the modifications brought in a cube-query by each multidimensional operator

be summarized using an aggregation function. In our example (see Figure 3.4), we perform two different **roll-ups**: on the one hand, we **roll-up** from `product_id` to the `All` **level**. On the other hand, we **roll-up** from `city` to `country`. Note that the `country` table is added to the FROM clause, and we replace the `city` identifier with that of the `country`level in the SELECT, GROUP BY and ORDER BY clauses. Finally, we add the proper links in the WHERE clause. About **rolling-up** from `product` to the `All` level, note that it is equivalent to remove both the `product` identifiers and its links.

- **ChangeBase**: In SQL it can be performed in two different ways. If we reorder the base (i.e., when "pivoting"), we just need to reorder the identifiers in the ORDER BY and SELECT clauses. But if **changing the base**, we need to add the new level tables to the FROM and the corresponding links to the WHERE clause. Moreover, identifiers in the SELECT, ORDER BY and GROUP BY clauses must be replaced appropriately. Following with the same example shown in Figure 3.4, we can change from $\{\texttt{day} \times \texttt{country} \times \texttt{All}\}$ to $\{\texttt{day} \times \texttt{country}\}$. Note that both bases are conceptually related by means of a one-to-one relationship. Specifically, this case typically applies when dropping a dimension (i.e., **rolling-up** to its `All` level and then **changing the base**). We **roll-up** to the `All` for representing the whole dimensions instances as a single one and therefore, producing the following base: $\{\texttt{day} \times \texttt{country} \times \texttt{1}\}$. Now, we can **changeBase** to $\{\texttt{day} \times \texttt{country}\}$ without introducing aggregation problems (since we **changeBase** through a one-to-one relationship).

- **Drill-across**: In SQL, we must add a new Cell table to the FROM clause, its measures to the SELECT, and the corresponding links to the WHERE clause. In general, if we are not using any semantic relationship, a new **Cell** table can always be added to the FROM clause if both Cells share the same base. In our example, suppose that we have a `stock` Cell sharing the same dimensions as the `sales` Cell. Then, we could **drill-across** to the `stock` Cell and show both the `stock` and `sales` measures (see Figure 3.4).

- **Projection**: In SQL it entails to remove measures from the SELECT clause. Following our example, we can remove the `discount` measure by projecting the `stock` and `price` measures.

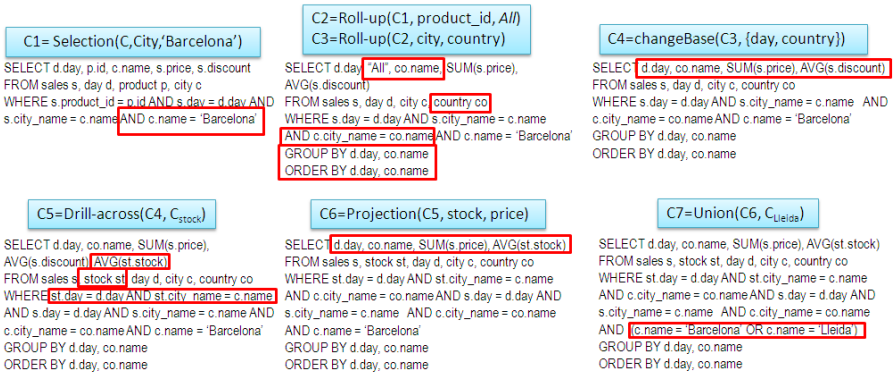


Figure 3.4: Exemplification of an OLAP navigation path translation into SQL queries

- **Union**: In SQL, we unite the FROM and WHERE clauses of both SQL queries and finally, we *or* the **selection** conditions in the WHERE clauses. Importantly, note that we can only **union** queries over the same Cell table. Intuitively, it means that, in the multidimensional model, the **union** is used to undo **selections**. We can unite our example query to one identical but querying for data concerning `Lleida` instead of `Barcelona`. As previously stated in section 2.2.1, these considerations can be easily extended to **difference** and **intersection**.

3.2.2 Potential Translation Conflicts

As discussed in previous section, OLAP users navigate the multidimensional data by providing a navigation path in which each node (resulting in a data cube) is derived from the previous node in the path by means of the multidimensional operators. For example, consider Figure 3.4, where a whole navigation path is depicted. At a given point, a node may combine a finite set of multidimensional operators. For instance, in the fifth node, this cube-query combines a **selection**, a **changeBase**, a **drill-across**, a **projection** and two **roll-ups**. The mapping to SQL of a single multidimensional operation does not represent a problem, but when combining the modifications brought about by a set of operations in a single SQL query, some conflicts could appear. Therefore, if these problems are not detected and treated appropriately, the automatic translation can retrieve unexpected results. In this section, we define and classify conflicts raised when automatically translating a navigation path to SQL.

Suppose an arbitrary navigation path. The user chooses a source data cube from where starting to operate and automatically, the ROLAP tool will conform a cube-query to retrieve the demanded data cube. Note that this data cube is our starting point so that it has not been yet manipulated by any operation. Consequently, it is placing a Cell of data on the n-dimensional space formed by its analysis dimensions. In a relational implementation, this Cell could have

been materialized. If it was, the ROLAP tool will retrieve the materialized data. Otherwise, it will look for an appropriate Cell, in a lower aggregation level, from where to obtain the needed Cell by means of roll-ups. For example, according to Figure 1.2 (see page 9), we could start our analysis from the materialized Cell (i.e., the `daily sales` per `product` and `city`) or from a non-materialized one; e.g., `annual sales` per `product` and `city`. As the latter is not materialized, we need to perform an *implicit* **roll-up** over the atomic Cell, from `month` to `year`, to get the needed data.

As presented in Table 3.2, certain operations may pop up a conflict when combined with an specific *source cube-query*. We denote source cube-query to an atomic cube-query modified by a sequence (note that we talk about sequence, because, in the multidimensional model, order matters) of operations. If no operation has been performed over the atomic cube-query we consider the empty sequence ($\emptyset$). Hence, a cell is crossed ($\times$) when the sequence of operations in the source cube-query contains a specific operation that may cause a conflict with the next one to be performed. For example, it may happen if our source cube-query includes a **selection** and next operation to be carried out is a **roll-up**.

Note that all the conflicts shown in Table 3.2 are caused by data aggregation anomalies. As introduced in [LS97], operations performed must satisfy the disjointness, completeness and compatibility of the summarization (i.e., the compatibility of the *dimension*, the *aggregation function* and the k*ind of measure* involved in the summarization) to guarantee its correct summarization. Otherwise, two operations that, as a whole, do not preserve the three conditions will raise up a conflict. Therefore, as presented in Section 3.2.1, **roll-up** is the only operator performing data aggregation and consequently, it is the only one that may directly raise up conflicts when combined with other operators in the same cube-query. Importantly, **roll-up** is the most relevant multidimensional operator, as it allows to modify the data granularity. Specifically, according to Table 3.2, all conflicts are related to **roll-up** and **drill-across**. The rest of operations except for **selection**, propagate conflicts if already present in the cube-query, but do not introduce new ones. Consequently, **projection**, **union** and **changeBase** never raise a conflict. Intuitively, **projection** removes measures from the SELECT clause and dropping a measure just means to discard one column of the Cell table; **union** *or*es conditions of two data cubes with the same n-dimensional space not removing / adding any point; and **changeBase** always asks for a one-to-one relationship, avoiding conflicts due to its own nature.

Operation/Source	$\emptyset$	Selection	Roll-up	Projection	Drill-across	ChangeBase	Union
Selection							
Roll-up	X	X	X		X		
Projection							
Drill-across	X		X		X		
ChangeBase							
Union							

Table 3.2: Summary of cube-query conflicts

Oppositely, **drill-across** and **selection** may introduce conflicts in the translation to SQL of the navigation path. **Drill-across** asks for a one-to-one relationship but sometimes, a one-to-many

relationship is enough. In these cases, due to not materialized Cells, we need to perform implicit **roll-ups** to get the necessary one-to-one relationship and consequently, potentially raising up the same conflicts caused by a **roll-up**. Similarly, it may happen with non-materialized atomic cube-queries that would need to perform implicit **roll-ups**. A **selection** may cause an specific conflict along with a **roll-up** if we select a subset of points of the data cube and later **roll-up**, which would prevent the ROLAP tool of using the pre-aggregated data (as done in the general case). Consequently, note that it is enough to analyze the potential conflicts between each pair of operators, since all of them are caused by conciliating multiple aggregations of data in just one cube-query and therefore, the order performed between the operators, at the cube-query level, does not matter.

Since all conflicts are due to data aggregation anomalies, we have classified them in three groups according to the three necessary conditions needed to guarantee a correct data summarizability: those performing multiple aggregation functions in a query (not preserving compatibility of data), those raising hidden many-to-many relationships (not preserving disjointness) and finally, those related to the selection granularity (not preserving completeness).

3.2.2.1 The Multiple Aggregation Problem

The first conflict is related to the functions used to aggregate data when combining more than two **roll-ups** in the same cube-query. To analyze this problem, we consider two scenarios: (i) if the **roll-ups** are performed over the same dimension or (ii) over different ones. In the first case, we can always solve the problem disregarding the first **roll-up** and just performing the second one. This assumption holds because, in a given time, multidimensional data can only be showed at a certain aggregation level for each dimension. Thus, in the worst scenario, we can solve this conflict by **rolling-up** from the atomic level. Oppositely, when performed over different dimensions, we must aggregate data for each of the dimensions. SQL does not allow to aggregate data by means of two different functions in the same query, and this conflict can not be solved in a single cube-query.

For example, in the first case, if we **roll-up** the `sales` Cell showed in Figure 1.2 (see page 9) from `day` to `month`, and later we **roll-up** from `month` to `year`, the whole sequence of **roll-ups** can be directly expressed as:

```
SELECT y.year, p.id, c.name, SUM(s.price), AVG(s.discount)
FROM sales s, product p, city c, day d, month m, year y
WHERE s.product_id = p.id AND s.day = d.day
AND s.city_name = c.name AND d.month_id = m.month
AND m.year_id = y.year
GROUP BY y.year, p.id, c.name
ORDER BY y.year, p.id, c.name
```

On the contrary, if we first **roll-up** from `day` to `month`, and later from `city` to `country`, nested queries are compulsory:

```
SELECT p.id, co.name, m.month, AVG(s.price), AVG(s.discount)
FROM (SELECT p.id, c.name, m.month, AVG(s.price), AVG(s.discount)
   FROM sales s, product p, city c, day d, month m
   WHERE s.product_id = p.id AND s.day = d.day
   AND s.city_name = c.name AND d.month_id = m.month
   GROUP BY p.id, c.name, m.month
   ORDER BY p.id, c.name, m.month), country co
WHERE s.product_id = p.id AND s.day = d.day
AND AND s.city_name = c.name AND c.country_name = co.name
GROUP BY p.id, co.name, m.month
ORDER BY p.id, co.name, m.month)
```

Even if SQL allowed to perform more than one aggregation function in the same query, we would face another problem: the order between the aggregation functions. For example, note that, in the above query, the `price` measure is aggregated by means of the `average` function over the `time` dimension, and by means of the `sum` function over the `place` dimension. Thus, it is important to realize that our own multidimensional conceptual design fixes the order of the aggregation functions when exploring the Cell hierarchy. Thus, order does really matter since `sum` of `averages` is different from an `average` of `sums`.

The above conflict could be avoided if SQL allowed to perform more than one aggregation function per query, and set up an order between them. For example, as showed below, an SQL extension stating explicitly two GROUP BY's (very similar to SQL'99 GROUPING SETS modus operandi), would avoid using nested queries when combining more than one conflictive **roll-up**. First GROUP BY would be related to the first aggregation function and analogously to second one:

```
SELECT p.id, co.name, m.month, SUM(s.price), AVG(s.discount)
FFROM sales s, product p, city c, day d, month m, country co
WHERE s.product_id = p.id AND s.day = d.day
AND AND s.city_name = c.name AND d.month_id = m.month AND
c.country_name = co.name
GROUP BY p.id, c.name, m.month
GROUP BY p.id, co.name, m.month
ORDER BY p.id, co.name, m.month
```

Although this problem has been presented as a **roll-up** plus **roll-up** problem, it goes far beyond, as it may happen when obtaining non materialized Cells from materialized ones. For example, if we start our navigation path from the `monthly sales per city` Cell that has not been materialized, ROLAP tools will need to perform a **roll-up** from `day` to `month` to obtain the needed data. So that, we have already performed an *implicit* **roll-up** that could arise conflicts if we next perform an explicit one from `city` to `country`. Similarly, as presented in Section 3.2.2.2, implicit **roll-ups** may also occur when performing a **drill-across** from a non materialized Cell (indeed, implicit **roll-ups** can also appear when **changingBase**, but in this case, the implicit and explicit **roll-ups** are performed over the same dimension -see the (i) case above- and thus, avoiding conflicts).

3.2.2.2 The Fan-Shaped Problem

In this section we introduce a family of problems that occur when disjointness of data aggregation is not preserved. It typically appears related to **drill-across**, either through semantic rela-

tionships or shared dimensions. **Drill-across** asks for a one-to-one relationship, but sometimes a one-to-many relationship is enough. For example, consider Figure 3.4. There, we have shown how to **drill-across** from the `daily sales per country` to the `daily stock per country`. Clearly, these two Cells are related by means of a one-to-one relationship. However, if they are not materialized, they give rise to a hidden many-to-many relationship. Note that, prior to performing this **drill-across**, we have dropped the `product` dimension and this is why this query that, at first sight seems correct, gives rise to a many-to-many relationship.

As enounced in [LS97], the aggregation of data must be disjoint, and in this case, it is not. In fact, what should be a one-to-one relationship turns into a many-to-many one calling up a fan-shaped matching. Thus, we should use a nested query performing first one **roll-up** and later, the other one, being the "join" last performed. This problem could be solved if SQL allowed to state a priority between "joins" and GROUP BY's.

Finally, also note that when carrying out a **drill-across** to a non materialized Cell, a ROLAP tool will need to perform internal **roll-ups** to obtain the appropriate aggregation level from where **drill-across**. Internal **roll-ups** followed by an explicit **roll-up** may cause the conflict stated in Section 3.2.2.1.

3.2.2.3 The Selection Granularity Problem

This problem is closely tied to **selection** and raises when completeness is not guaranteed. **Selection** allows to reduce the current multidimensional space by means of a logic clause over a certain descriptor. For example, selecting those cells of `monthly sales per city` related to `Barcelona`. Now, if we decide to materialize this Cell in the data warehouse, we cannot take advantage of it in those navigation paths not considering this **selection**. In the general case, ROLAP tools use materialized Cells to speed up the query processing, but note that a navigation path not preserving the Cell data granularity would not benefit from it, as data completeness is not guaranteed.

For example, if we **roll-up** from `daily sales per city` to `monthly sales per city` we cannot take advantage of the `monthly sales in Barcelona` to answer this query. Simply, we do not dispose of data for the rest of cities in this materialized Cell (i.e., completeness is not preserved). In this case, using the appropriate data granularity (in the worst case, the atomic Cell) and performing internal **roll-ups** is mandatory.

In short, this conflict invalidates pre-aggregated data (i.e., materialized Cells) not containing the same (or a finer) data granularity level with regard to the current navigation path.

3.2.3 Discussion: The Multidimensional Integrity Constraints

In this section we have analyzed how the multidimensional algebra must be translated into SQL. As result, we have been able to identify the constraints a SQL query must satisfy to make multidimensional sense: it must follow the cube-query pattern (i.e., *it must retrieve a data cube*) and it must be *free of summarizability problems*. Formally, *we say that a SQL query is a correct cube-query if it retrieves data that can be analyzed from a multidimensional perspective*. I.e.,:

- Factual data is arranged in a multidimensional space (i.e., it forms a data cube). Thus, each

instance of factual data is identified (i.e., placed in the multidimensional space) by a point in each of its analysis dimensions.

  - As consequence, we must be able to identify a minimal set of levels identifying the cells placed in the multidimensional space. According to our terminology, we denote by base to this minimal set of levels *determining* the factual data.

- Data summarization must be correct, which is ensured by guaranteeing three necessary conditions (which, intuitively, are also sufficient) [LS97]: (1) **Disjointness** (the sets of objects to be aggregated must be disjoint); (2) **Completeness** (the union of subsets must constitute the entire set); and (3) **Compatibility** of the dimension, the type of measure being aggregated and the aggregation function.

3.3 Problem Context

The main aim of our approach is to support the data warehouse design process. It consists of two steps: requirement formalization and the MDBE method (as shown in Figure 3.1). Furthermore, as in any classical design process, a requirement elicitation pre-process is needed. Although this pre-process falls outside the scope of this work, some relevant features should be noted here. Data warehousing systems differ in various aspects from conventional operational systems (since they are designed to support decision-making) and need specialized requirement elicitation processes [MTL07, WS03]. However, this issue has been studied in depth, and there are several methods that can be used in preliminary step (for example, [GRG05, MTL07, PSG04, SLB02, WS03]). Nevertheless, note that we gather *information requirements* in this step. Information requirements [WS03] are designed to meet end-user information necessities, which is the objective of a data warehouse [MTL07]. Unlike in other systems, end-users can easily determine their information necessities because they consist of business queries posed in their daily decision-making processes. Consequently, information requirements can be stated in the end-users' own words and closely reflect their reality. For example, *"examine stocks provided by suppliers"* or *"analyze customer purchases with regard to region, product and time"* would be typical information requirements.

The next step in our approach formalizes the requirements gathered. As discussed previously, we aim to automate the manipulation of requirements (i.e., integrate them in a fully-automated method), so they must be translated into a computer understandable language. In our approach, end-user requirements are expressed as SQL queries over the relational data sources (i.e., at the logical level over the data sources). This step must be carried out by a database expert capable of lower the level of abstraction of the input requirements to the logical level (see Section 3.1.1 for a detailed discussion of the advantages and disadvantages of this step).

As shown in Figure 3.1, the next step in our approach is to apply the MDBE method, which has two inputs: the end-user information requirements (expressed as SQL queries) and the logical model of the data sources. As output, MDBE presents a multidimensional schema derived from the data sources, which allows the user to retrieve data demanded in the input requirements. In this step, MDBE determines *whether each input SQL query represents a valid multidimensional query*, i.e., if the query retrieves data that can be analyzed from a multidimensional perspective;

this is the case 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 3.3.1 for further information). These constraints can be summarized as follows: data retrieved should be (1) free of data summarizability anomalies, and (2) placeable in a multidimensional space. If these constraints are satisfied, we may find a set of multidimensional operators which would retrieve that data from the proposed multidimensional schema. Finally, note that each query (i.e., each multidimensional requirement) produces a potential multidimensional schema. The last step in the MDBE method would allow the user to conciliate those results into a minimal set of conceptual schemas that meet all of the requirements (i.e., a constellation of multidimensional schemas).

3.3.1 Foundations

In this section we identify and refresh all the concepts introduced up to now in this thesis, that conform the foundations of the MDBE approach.

[Notation]: In this document, we use the multidimensional notation introduced in Section 1.5. Briefly, multidimensionality is based on the fact / dimension dichotomy. **Dimensional concepts** produce the multidimensional space in which the **fact** is placed. **Dimensional concepts** are those concepts likely to be used as a new analytical perspective, which have traditionally been classified as **dimensions**, **levels** and **descriptors**. Thus, we consider that a **dimension** consists of a hierarchy of **levels** representing different granularities (or levels of detail) for studying data, and a **level** containing **descriptors** (i.e., **level** attributes). In contrast, a **fact** contains **Cells** which, in turn, contain **measures**. We consider that a **fact** may contain not just one but several different levels of data granularity. Therefore, one **Cell** represents a class of individual cells of the same granularity that contain data relating to the same **fact**. Specifically, a **Cell** of data is related to one **level** for each of its associated **dimensions** of analysis. Finally, one **fact** and several **dimensions** for its analysis produce a star schema.

Next, we present the multidimensional constraints upon which the study is based, namely those used to validate the input SQL query (i.e., the information requirement) as a suitable multidimensional requirement. As discussed in Section 3.2, we found that multidimensional data manipulation (i.e., multidimensionality) focuses on two aspects: (i) the placement of data in a multidimensional space; and (ii) the correct summarizability of the data. If the data retrieved satisfy both constraints they can be depicted as a data cube (i.e., orthogonal dimensions fully functionally determining the fact) free of summarizability problems. In other words, this query would represent the translation of a set of multidimensional operators into SQL. Below, we provide formal definitions of the basic axioms identified in our study (for the sake of understandability, the multidimensional concepts are **bolded**):

[Definition 1] The cube-query template: The standard SQL'92 query template to retrieve a Cell of data from the relational database management system was introduced and discussed in Section 3.2:

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

The FROM clause contains the "Cell table" and the "Level tables". These tables are properly linked in the WHERE clause by "joins" that represent *concept associations*. The WHERE clause also contains logical clauses restricting a specific **level** attribute (i.e., a **descriptor**) to a constant using a comparison operator. The GROUP BY clause, if present, shows the identifiers of the **levels** at which we want to aggregate data. Those columns in the grouping must also be in the SELECT clause to identify the values in the result. Finally, the optional ORDER BY clause is designed to sort the output of the query.

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

[Definition 2] The multidimensional space arrangement: Dimensions arrange the multidimensional space in which the **Cell** under study is depicted. Each instance of factual data is identified (i.e., placed in the multidimensional space) by a point in each of its analysis **dimensions**. Conceptually, this means that a **Cell** must be related to each analysis **level** by a to-one relationship; that is, every instance of the **Cell** (i.e., every cell) is related to exactly one instance of an analysis **dimension**, and every **dimension** instance may be related to many instances of the **Cell**.

[Definition 3] The Base concept: We use **base** to denote a *minimal* set of **levels** functionally determining a **Cell**. This guarantees that two different instances of data cannot be placed in the same point of the multidimensional space; in other words, given a point in each of these **dimensions**, they determine only one instance of data. In addition, **dimensions** (and thus, **levels**) that produce a **base** must be *orthogonal* (i.e., functionally independent) [ASS06]. Otherwise, we would use more **dimensions** than strictly needed to represent data, which would generate empty meaningless zones in the space. In a relational implementation of the data warehouse, the **base** concept would be implemented as the **Cell** primary key.

[Definition 4] The necessary conditions for correct summarizability: Data summarization must be correct, which is ensured by applying three necessary conditions (which, intuitively, are also sufficient) [LS97]: (1) **Disjointness** (the sets of objects to be aggregated must be disjoint); (2) **Completeness** (the union of subsets must constitute the entire set); and (3) **Compatibility** of the **dimension**, the type of measure being aggregated and the aggregation function. Compatibility must be satisfied, since certain functions are incompatible with some **dimensions** and types of measures. For example, we cannot aggregate `Stock` over `Time` **dimension** by means of sum, as some repeated instances would be counted. Unfortunately, compatibility cannot be automatically verified. Nevertheless, our method keeps track of the compatibility information extracted from the requirements (see

Steps 1 and 3 of Section 3.4.1; there, data summarizability properties identified from the query are considered and properly stored).

The MDBE validation process checks if each SQL query can produce a meaningful multidimensional cube. Below, we present a set of criteria directly applied by MDBE, which can be derived easily from the definitions presented above. For our purposes, *an SQL query is considered to make multidimensional sense if it satisfies the following semantic constraints*:

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

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

3.3.1.1 Additional Considerations

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

Relaxing Completeness and Disjointness Our method can identify when disjointness and completeness are not preserved in the logical schema of the data sources. However, end-user requirements may include new concepts that have not been captured in the data sources but which may still be derived from them. Specifically, (i) in the SQL query, the user may require a concept specialization not preserving completeness in the logical schema or ii) a derived measure not preserving disjointness. Consider Figure 3.2. The first case would apply if, for example, a `lineitem` does not require a `suppkey` when inserting data (i.e., if `suppkey` allows *NULL* values). Note that it would make sense, as we may assign the `supplier` later, by means of another task that, for example, minimizes expenses. In this case, if we want to use `supplier` as an analytical perspective of `lineitem` it would not preserve completeness regarding the data sources. According to the criteria discussed in this section (see [C5]), *NULL* values should not be allowed when relating factual data to dimensional data. However, this relationship (and thus, this query) would make sense if we are interested in analyzing only those `lineitem with supplier` (i.e., a specific specialization of `lineitem` not depicted in the sources). Thus, we may relax [C5] and produce this result if the user is interested in this specialization. Nevertheless, it is important to note that [C5] is relaxed regarding the data sources, but it is guaranteed by considering the specialization.

In the second case, two values not satisfying the disjointness constraint may produce a meaningful derived measure; for example, if the measure is properly weighted in the query. TPC-H Q9 is an example of this case. See the graph in the right side of Figure 3.9 (see page 89). This query selects factual data from two different nodes (`lineitem` and `partsupp`). However, these nodes are related by means of a many-to-one relationship that, in principle, will produce double-counting (factual data from `partsupp` may be considered several times when joining it to factual data in `lineitem`). According to our criteria (see [C6]), double-counting must be forbidden to preserve disjointness. However, double-counting may happen and yet make multidimensional sense. Indeed, this is the case of Q9. This query, weights the `l_quantity` attribute of `lineitem` with the `ps_supplycost` of `partsupp` (one `ps_supplycost` may be related to many `l_quantity` values) and this value is taken away from the income obtained from the sale. In other words, it is calculating the profit obtained on a given line of parts. Clearly, despite not preserving disjointness, the semantics of Q9 does make multidimensional sense. In general, our method produces results which satisfy [C5] and [C6]. However, in the MDBE method we apply the following rule:

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

In Section 3.4, we will clearly note those steps in which this assumption stands. Steps affected by this assumption will try to guarantee both constraints, but if no result is produced the steps have to be relaunched and [C5] and [C6] relaxed. In other words, if [C5] and [C6] are not preserved regarding the data sources, we propose alternative solutions preserving them (i.e., considering specializations or derived measures not captured in the sources). In such cases, MDBE may produce schemas that do not preserve the completeness or disjointness of the data sources, and we inform the user that results are only correct if either a concept specialization or a new derived measure is considered.

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

Consider Figure 3.2 and a query containing two different join paths between `lineitem` and `nation` in the WHERE clause. For example, one path following the `lineitem - orders - customer - nation` foreign key - primary key relationships, and another following the `lineitem - partsupp - supplier - nation` foreign key - primary key relationships. Clearly, these joins are used to select `lineitems` having as `customer` and `supplier` people living in the same `nation` (i.e., we select factual data having the same value for both paths). But this query would not make multidimensional sense: it does not exist any multidimensional operator performing such operation and thus, there is no OLAP tool that could produce this query. Consequently, MDBE should disregard this query. However, it may happen that both paths are equivalent (i.e., both values of `nation` always coincide for each and every `lineitem`). For example, it would be the case if the following integrity constraint holds within the organization: *every* `customer` *will be provided with items supplied by a* `supplier` *of his / her own country*. Indeed, such constraint would make sense in many organizations.

For this reason, if the query contains alternative join paths between two concepts we may distinguish two cases: whether the user guarantees that both paths are equivalent (i.e., instances selected through each path are exactly the same) then, we can rewrite it in such a way it preserves [C7]. Otherwise, the query should be disregarded. In the first case, the query must be rewritten by defining two different *alias* for `nation` (for example, `nation n1` and `nation n2`) and use each in one of the paths. The resulting query would be semantically equivalent to the previous one and it will make multidimensional sense (i.e., this query could be generated by OLAP tools).

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

3.3.2 Internals

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

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

In our approach, MDBE aims to validate the multidimensional graph (i.e., the input query) as a suitable multidimensional requirement. It *labels* each graph node and its attributes as mul-

tidimensional concepts, in such a way that the whole labeling satisfies the multidimensional constraints described in Section 3.3.1. Thus, MDBE tries to find a multidimensional meaning for the relational schema fragment captured in the graph (roughly speaking, we may say that it looks for a *multidimensional interpretation* of the graph).

This section discusses how each relational concept may be labeled. Here, we present the criteria used to identify the multidimensional role of attributes, nodes and edges according to our foundations (see Section 3.3.1). Later, Section 3.4.1 introduces an algorithm that applies these criteria. The algorithm analyzes the SQL query and the relational schema to look for the labeling criteria introduced in this section and accordingly, labels the graph.

3.3.2.1 Attribute Labeling

A given relational attribute of multidimensional interest may play a dimensional or a factual role. If it has a useful analytical value it will be labeled as a **measure** (i.e., factual data) and if it represents an interesting analytical perspective for the 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 a **descriptor**.

We use the 7 criteria introduced in Section 3.3.1 to identify the multidimensional role played by an attribute in the SQL query. Section 3.4.1 introduces an algorithm for analyzing the SQL query and the data source schema according to each criterion and label attributes. Steps 2, 3, 4 and 6 show how MDBE determines the multidimensional role an attribute plays.

Note that a given attribute may be labeled as both a **dimensional concept** and a **measure**. In their multidimensional model, Agrawal et al. [AGS97] proposed handling **measures** and **dimensions** *uniformly* (they presented two multidimensional operators to transform **measures** into **dimensions** and vice versa). This idea was also incorporated into later multidimensional models (see [RA07b]). MDBE allows multiple labeling of a relational attribute (which we will refer to as *dual attributes*), so the final multidimensional schema will contain a **measure** and a **dimensional concept** derived from the same attribute. It will happen if a given attribute is labeled as factual data according to any of our criterion, and as dimensional data by another.

3.3.2.2 Node Labeling

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

- *Dimensional data* (*L*): If the node contains attributes representing a useful analytical perspective for the multidimensional data, it will be labeled as a **level** (i.e., as *L*).
- *Factual data* (*CM* or *C*): If the node contains factual data, we label it as a **Cell**. However, we distinguish between two different types of **Cells**:
  - *Cell With Measures* (*CM*): These nodes represent **Cells** that contain **measures**. According to [C3], these nodes will also contain **dimensional concepts** that determine

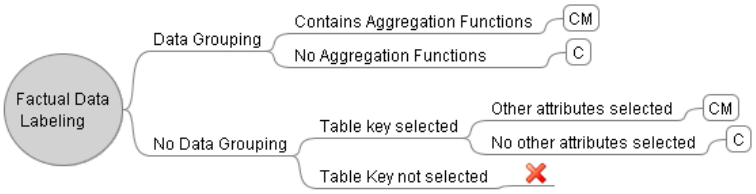


Figure 3.5: MDBE: decision diagram for labeling nodes representing factual data

the multidimensional space in which to place the data. There are three possible cases: the **Cell** directly contains (i) the multidimensional **base**, (ii) a *candidate* **base** (i.e., a set of attributes preserving a one-to-one relationship with the multidimensional **base**) or (iii) a set of attributes fully determining the multidimensional **base**.

To preserve [C3], in the (i) and (ii) cases, it means that either the multidimensional **base** (candidate **base**) corresponds to a table CK (also represented in the node) or, if performing data aggregation in the query, the GROUP BY clause is compound of attributes of the node. For the (iii) case, consider the TPC-H business query #5 previously introduced, and the TPC-H relational schema shown in Figure 3.2. In this query, the `name` attribute (from the `nation` node) forms the multidimensional **base** and `lineitem` plays a **Cell** role. To preserve [C3], `lineitem` is properly *linked* to `name` in such a way that every instance of factual data is related to just one `nation name` value. In other words, `lineitem` functionally determines the `nation` of the `supplier` (indeed, this dependency is properly captured in the relational schema by means of FKs). We use *link attributes* to denote the **dimensional concepts** contained in a **Cell** placing factual data in the multidimensional space (i.e., the (i), (ii) or (iii) cases discussed).

– *Cell* (*C*): These nodes represent "factless facts" [KRTR98]. This definition is equivalent to the previous one, but this type of **Cell** does not contain **measures**. These facts are very useful for describing events and coverage and can be used to formulate many interesting questions [KRTR98].

To determine the factual label of a node, we follow the decision diagram shown in Figure 3.5, which generates questions about the query and the table metadata. These questions derive directly from constraints introduced in Section 3.3.1, and we distinguish between two possible scenarios: one in which the current input query performs data grouping (i.e., it contains a GROUP BY clause) and another in which it does not. In the first case, and according to [C1], if the SELECT clause contains an *aggregated attribute* (i.e., summarized by an aggregation function), that attribute will play a **measure** role. Consequently, the node is labeled as *CM*. Otherwise, if no aggregated attribute is selected, it is labeled as a factless fact *C* (i.e., the **Cell** does not contain **measures**).

Similarly, if no data grouping is performed in the query but we are able to produce a multidimensional space (i.e., a table CK is selected), the node will be labeled as a **Cell**: *CM* if attributes

other than the key are selected (i.e., if it contains **measures**); otherwise, *C*. According to [C6], any other alternative would not make multidimensional sense as a **Cell** (depicted in the figure by the *X* mark).

When checking if any **measure** other than a table key is selected, we do not only consider numerical attributes. Traditionally, numerical attributes produce **measures** because they are perfectly additive but, as discussed in [KRTR98], semi-additive or non-additive values could be of interest to the end-user. Moreover, there are some areas in which non-numerical values are additive. For example, the *spatial databases* area contains algorithms for the aggregation of text values representing geographical information (see [CMTV00]).

Note the multidimensional semantics involving each alternative in the decision diagram discussed above. **Cells** identified without grouping will represent "atomic factual data" [ASS06] (i.e., the finest granularity of data in the data warehouse), whereas those **Cells** identified by data aggregation will represent "aggregated factual data" (i.e., coarser data granularities of interest).

Similarly, to determine the dimensional role of a node we follow the decision diagram shown in Figure 3.6. Again, it generates questions about the query and the table metadata. First, we check if the input query performs data grouping, and according to [C3], attributes in the GROUP BY (i.e., contains attributes being part of the multidimensional **base**) will play a **level** role. Consequently, the nodes containing these attributes are labeled as *L*. If no data grouping is performed or none of the node attributes are used to group data, we check the WHERE clause. We distinguish between two possible scenarios: if any of the node attributes are involved in a *comparison clause* or in a *join*. In the first case, according to [C7], selections are performed over dimensional data and thus, that attribute will be identified as dimensional data. In the second case, according to [C1], joins in the WHERE clause represent *conceptual associations*. Thus, if a node contains an attribute joined to a dimensional attribute (i.e., an attribute already identified as dimensional data) then, both attributes represent dimensional data. Any other scenario would not make sense as dimensional data and the node is not labeled (see the the *X* mark in the figure).

Supporting Denormalization: As discussed in Section 3.1, our method can handle denormalized input schemas, which means that a given node may play a factual and dimensional role simultaneously. This scenario occurs when a graph node is labeled as factual data by the decision diagram shown in Figure 3.5, and as dimensional data by the decision diagram shown in

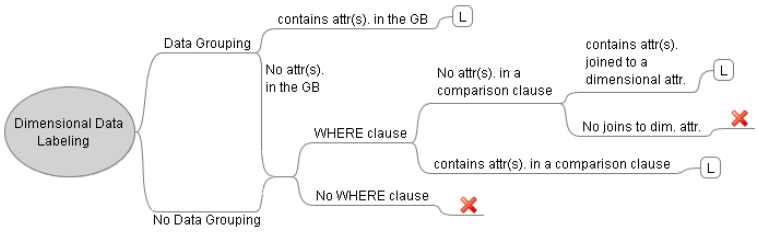


Figure 3.6: MDBE: decision diagram for labeling nodes representing dimensional data

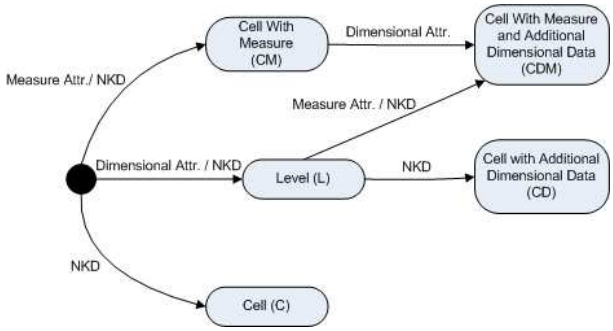


Figure 3.7: MDBE: state diagram showing the transition between node labels

Figure 3.6.

In this case, we introduce two new labels to identify *hybrid nodes* containing factual *and* dimensional data. Note, however, that a factual node (i.e., those labeled as *CM* or *C*) always contains dimensional data forming the multidimensional space (i.e., the *link attributes*). However, hybrid nodes contain *additional* dimensional data: either attributes playing a degenerated dimension [KRTR98] role and / or attributes playing the role of denormalized dimensional data (i.e., partial or whole denormalized dimension hierarchies):

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

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

For example, consider the TPC-H business query Q5 previously introduced. If this query contained an additional comparison clause such as `l_shipdate = '12-02-2009'` in the WHERE clause, MDBE would identify `lineitem` as a hybrid node: according to the decision diagram shown in Figure 3.5, `lineitem` is labeled as *CM* (because the query contains grouping and `lineitem` contains two aggregated attributes -i.e., `l_extendedprice` and `l_discount`- in the SELECT clause) and, according to the decision diagram shown in Figure 3.6, it will also identify `lineitem` as dimensional data (because `lineitem` does not contain any attribute in the GROUP BY clause, but it contains `l_shipdate`, which is involved in a comparison clause in the WHERE). As result, MDBE labels this node as a hybrid node (*CDM*) since it contains **measures**, the *link attributes* and also an additional dimensional attribute.

Once we know how to label attributes (by means of the 7 criteria introduced in section 3.3.1) and nodes (by means of the decision diagrams previously introduced in this section), the node labeling state diagram can be produced, as shown in Figure 3.7. The transitions between possible labels are shown. Every node is unlabeled in its initial state (i.e., at the beginning of the labeling

CK_{n1}	CK_{n2}	FK_{n1}	FK_{n2}	NN_{n1}	NN_{n2}	Relationship	Multiplicity
×	×	×	×	?	?	$Attr. \rightarrow Attr.$	$N - M$
✓	×	×	✓	✓	✓	$CK \rightarrow FK + NN$	1 -o N
✓	×	×	?	✓	?	$CK \rightarrow Attr.$	1 o-o N
×	✓	✓	×	✓	✓	$FK + NN \rightarrow CK$	N o- 1
×	✓	?	×	?	✓	$Attr. \rightarrow CK$	N o-o 1
✓	✓	✓	✓	✓	✓	$CK + FK \rightarrow FK + CK$	$1 - 1$
✓	✓	✓	×	✓	✓	$CK + FK \rightarrow CK$	1 o- 1
✓	✓	×	✓	✓	✓	$CK \rightarrow CK + FK$	1 -o 1
✓	✓	×	×	✓	✓	$CK \rightarrow CK$	1 o-o 1

Table 3.3: Summary of rules used to infer the relationship multiplicities from relational sources

process) and the label is then updated according to the explicit knowledge extracted from the query. For example, from the initial state, we can label each node as either *CM* (if one of its attributes is identified as a **measure**) or *L* (if one of its attributes is identified as a **dimensional concept**). From the *CM* state, we can keep the same label if any other **measure** is identified or update it to *CDM* if an attribute playing a dimensional role and not part of the *link attributes* is identified (i.e., if this node contains factual and dimensional data).

Some transitions shown in the state diagram are labeled with the *NKD* (*New Knowledge Discovery*) tag. In MDBE, a state transition can take place due to either the explicit knowledge extracted from the query or the implicit knowledge derived both from the input query and the data source metadata. The latter case represents a scenario in which either the query does not *explicitly* establish a node role (thus, the node is not yet labeled) or the implicit knowledge available suggests an *alternative* labeling. In these cases we analyze every labeling alternative for the node in question. As discussed in Section 3.1 and presented in detail in Section 3.4.1 (see Step 6), this process is used to derive new multidimensional knowledge that is not stated in the requirements.

3.3.2.3 Edge Labeling

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

Next, we introduce the edge labeling process:

- First, for each join between tables in the WHERE clause, we first infer the relationship multiplicity with regard to the schema constraints of the join attributes (i.e., FKs, CKs or not *NULL* values). In the relational model, the multiplicity of a relationship depends on how attributes involved are defined in the schema: Whether they (as a whole, since we consider multi-attribute joins) play the role of a relation CK and / or if they are defined as a FK to the other attribute(s) and / or if they allow *NULL* values. Joining to a CK guarantees to match at most one instance of the relation[3]. Otherwise it may match many of them. Similarly, an attribute not allowing *NULL* values and being defined as FK will surely match one and just one instance. Otherwise, it may introduce zeros. Table 3.3 summarizes all those relationship multiplicities that we may find in the relational model with regard to the attributes metadata. There, each row represents an specific relationship between nodes (i.e., a kind of join). Notation used is the following: first six columns represent all possible combinations with regard to the constraints of join attributes (the subscripts *n1* and *n2* refer to each one of the attribute sets joined): As CK, as a FK pointing to the other attribute(s) or as NN (*not NULL*) attribute(s). If an specific cell is ticked (i.e., ✓), it means that that attribute is constrained according to that column. Otherwise, it is marked with a $\times$ mark. Notice that not all the combinations are allowed and some columns determine the following ones. For instance, CK attribute(s) can not accept *NULL* values. Moreover, a cell is marked with a ? mark if previous columns already determine a certain multiplicity, meaning that this constraint does not affect the obtained multiplicity. Finally, last two columns inform about the specific join depicted as well as the multiplicity inferred. There, an `Attr.` represents unconstrained attribute(s); that is, not defined neither CK nor FK and allowing *NULL* values.
- Next, according to the semantics of the multiplicity inferred, we label each edge with those multidimensional relationships it could represent (i.e., the multidimensional concepts it could relate). Potential edge labels are shown in Table 3.4, and those combinations making

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

Multiplicity	*Level - Level*	*Cell - Cell*	*Level - Cell*	*Cell - Level*
1 - 1	✓	✓	✓	✓
1 o- 1	✓	✓	$\checkmark_c$	✓
1 o-o 1	✓	✓	$\checkmark_c$	$\checkmark_c$
N - 1	✓	✓	$\times$	✓
N o- 1	✓	✓	$\times$	✓
N o-o 1	✓	✓	$\times$	$\checkmark_c$
N -o 1	✓	✓	$\times$	$\checkmark_c$
N - *M*	$\times$	$\checkmark_d$	$\times$	$\times$
N -o *M*	$\times$	$\checkmark_{dc}$	$\times$	$\times$
N o- *M*	$\times$	$\checkmark_{dc}$	$\times$	$\times$
N o-o *M*	$\times$	$\checkmark_{dc}$	$\times$	$\times$

Table 3.4: Valid multidimensional relationships in a relational schema

multidimensional sense (according to [C2], [C5] and [C6]) are marked with a ✓. For example, a many-to-one relationship, depending on zeros, could represent a *Cell - Level*, *Cell - Cell* or a *Level - Level* relationship but not a *Level - Cell* relationship, since it would not satisfy [C2]. However, completeness could eventually be relaxed to identify concept specializations, as explained in Section 3.3.1.1. These cases, in which completeness would be relaxed a posteriori, are shown in Table 3.4 as $\checkmark_c$.

It can also be seen that many-to-many relationships would not generally produce valid labeling. According to the constraints presented in Section 3.3.1, a many-to-many relationship is meaningless in the multidimensional model. Nevertheless, there is one case in which we may consider many-to-many relationships, since we could eventually relax disjointness to identify derived measures, as explained in Section 3.3.1.1; this exception, in which disjointness would be relaxed a posteriori, is shown in Table 3.4 as $\checkmark_d$.

Finally, and as previously stated, we would like to remark that a node required to play a dimensional role by an edge label, can only be labeled as *L* and not as *CDM* or *CD*. Although these two labels represent hybrid nodes (and thus, they also contain dimensional data), their semantics are different from those of the *L* label. Importantly, edges relate nodes, and they determine the role that the related nodes may play according to the join conditions. Consider again Table 3.4. A node may play a **level** role whether: (i) it is placed in the to-one end of a relationship (see second, fourth and fifth column) or (ii) it is placed in the to-many end of a *Level - Level* one-to-many relationship (see second column).

By definition, the cardinality of factual data within a hybrid node is greater than (or in a degenerate case, equal to) that of the dimensional data it contains. Thus, in the (i) case, when an edge relates a node *n* to a hybrid node *h* by means of a to-one relationship, the link relates *n* to the factual data in *h*. Otherwise, if the link were relating *n* to the dimensional data in *h*, it would not raise the to-one multiplicity. For this reason, hybrid labels cannot be used in this case. In the (ii) case the reason is subtler. According to Table 3.4, the node in the to-many end may represent a **level** (see second column) or a **Cell** (see third and fifth column). However, labeling it as a hybrid node entails that this node contains factual and dimensional data and, by the same reasoning as in the previous case, we are relating the factual data in *h* (i.e., the hybrid node) to the data in *n* (i.e., its counterpart node) by means of a many-to-one relationship. For this reason, the semantics of this edge would capture a *Cell - Level* or a *Cell - Cell* relationship (depending on the role of *n*), but never a *Level - Level* relationship. Indeed, a *Level - Level* relationship can only be obtained by considering *h* to play a strict dimensional role (i.e., labeling it as *L*).

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

3.4 MDBE: Multidimensional Design Based on Examples

The MDBE method has two inputs: the end-user information requirements (expressed as SQL queries) and the logical model of the data sources. As output, our method produces a constel-

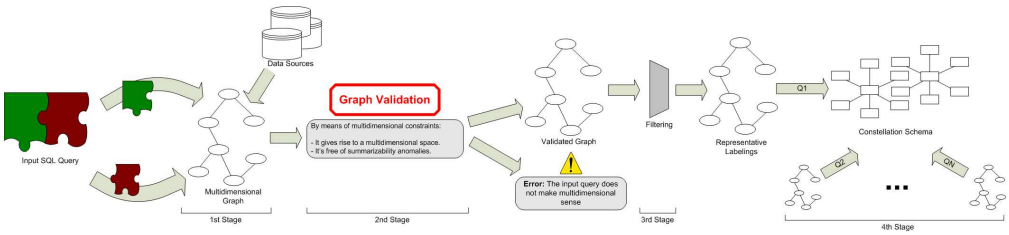


Figure 3.8: Summary of the MDBE process

lation schema from the data sources, which allows the user to retrieve the data requested in the input requirements. In this scenario, each query is analyzed to derive a multidimensional schema that meets the information requirements. This automatic process is depicted in Figure 3.8 and can be divided into four different stages:

- For each input query, the first stage (see Section 3.4.1) extracts the multidimensional knowledge contained in the query (i.e., the multidimensional role played by each concept in the query and the conceptual relationships between concepts), which is properly stored in the multidimensional graph. For this purpose, we apply the labeling methods discussed in Section 3.3.2. In this stage, the role played by the data sources will be crucial in inferring the conceptual relationships between concepts.

- The second stage (see Section 3.4.2) validates the multidimensional graph created in the first stage according to the constraints introduced in Section 3.3.1. The aim is to check whether the concepts and relationships stated in the graph collectively produce a data cube. From the graph building perspective, the first stage of the MDBE method is designed to derive a multidimensional labeling (i.e., label attributes, nodes and edges) to be validated in the second stage (i.e., checking the overall soundness of the graph). Therefore, this stage determines whether we would be able to use a set of multidimensional operators to retrieve data requested in the input query from the multidimensional schema represented by the multidimensional graph. If the validation process fails our method ends, since the required data cannot be analyzed from a multidimensional perspective (i.e., we are not be able to retrieve the requested data simply by using multidimensional operators). Otherwise, the resulting multidimensional schema is directly derived from the multidimensional graph.

- The third stage (see Section 3.4.3) finds the most representative results among those obtained. The step in which new multidimensional concepts are discovered may introduce new results (i.e., labelings) of potential interest, and we introduce a rule for determining which results should be presented to the user.

- Finally, the fourth stage (see Section 3.4.4) conciliates the multidimensional schemas obtained for each query. The result is a minimal constellation schema subsuming each of the schemas obtained for the input queries.

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

3.4.1 First Stage: Concept Labeling

The first stage is designed to build the multidimensional graph in 6 steps by applying the labeling standards introduced in Section 3.3.2. In this section, we introduce a detailed algorithm in pseudo-code (the *MDBE algorithm*) for implementing the first MDBE stage. This algorithm is followed by a brief explanation and an example of the execution of each step (based on the TPC-H schema). For the purposes of the study, the comprehensibility of the pseudo-code took priority over its performance (nevertheless, some optimizations have already been applied for its implementation in the MDBE tool):

declare MDBE ALGORITHM **as**

1. **For each** table **in the** FROM clause **do**
   (a) **Create** a node and **Initialize** node properties;
2. **For each** attribute **in the** GROUP BY clause **do**
   (a) **Label** attribute **as Level**;
   (b) node = $get\_node(attribute)$; **Label** node **as Level**;
   (c) **For each** attr2 **in** $follow\_conceptual\_relationships$(attribute, WHERE clause) **do**
      i. **Label** attr2 **as Level**;
      ii. node = $get\_node(attr2)$; **Label** node **as Level**;
3. **For each** attribute **in the** SELECT clause **not in the** GROUP BY clause **do**
   (a) **Label** attribute **as Measure**;
   (b) node = $get\_node(attribute)$; **Label** node **as Cell with Measures selected**;
4. **For each** comparison **in the** WHERE clause **do**
   (a) attribute = $extract\_attribute(comparison)$;
   (b) **if** !(attribute **labeled as Level**) **then**
      i. **Label** attribute **as Descriptor**;
      ii. node = $get\_node(attribute)$; **Label** node **as Level**;
   (c) **For each** attr2 **in** $follow\_conceptual\_relationships$(attribute, WHERE clause) **do**
      i. **if** !(attribute **labeled as Level**) **then**
         A. **Label** attribute **as Descriptor**;
         B. node = $get\_node(attribute)$; **Label** node **as Level**;
5. **For each** join **in the** WHERE clause **do**
   (a) /* Notice a conceptual relationship between tables may be modeled by several equality clauses in the WHERE */
   (b) $set\_of\_joins = look\_for\_related\_joins(join)$;
   (c) $multiplicity = get\_multiplicity(set\_of\_joins)$; relationships fitting = {};
   (d) **For each** relationship **in** $get\_allowed\_relationships(multiplicity)$ **do**
      i. **if** !($contradiction\_with\_graph(relationship)$) **then**

A. relationships fitting = relationships fitting + {relationship};

(e) **if** !($sizeof(relationships fitting)$) **then** return $notify\_fail$("Node relationship not allowed");

(f) **Create** an edge($get\_join\_attributes(set\_of\_joins)$); **Label** edge **to** relationships fitting;

(g) **if** ($unequivocal\_knowledge\_inferred(relationships\_fitting)$) **then** propagate knowledge;

6. **for each** g **in** New_Knowledge_Discovery(graph) **do**

(a) output += validation_process(g); //A detailed pseudo-code of this function can be found in section 3.4.2

return output;

The algorithm analyzes each query clause according to Def. 1:

Step 1: Each table in the FROM clause is represented as a node in the multidimensional graph. As presented in Section 3.3.2, MDBE will try to label every node, attribute and edge depicted in the query. Each node will keep track of relevant metadata inferred during the process. Specifically, we retain relevant metadata related to the query and referring to the data cube retrieved (if it makes multidimensional sense): the data cube base and *compatibility* information.

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

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

Current methods has thus far relief on foreign keys to identify dimensional data, so results depend on the degree of normalization of the data sources (see Section 3.1 for further information). In our approach we are not tied to design decisions affecting the data source logical schemas and can identify them from the requirements. For example, that the user state relationships not depicted in the logical schemas of the data sources (for instance, data grouping). Consequently, every attribute identified in this step is labeled in the multidimensional graph as an interesting **level** of analysis.

In these steps, each time an attribute is labeled, the label of the node to which it belongs will be properly updated according to the decision diagram shown in Figure 3.7. Finally, we add the identified data cube base to the graph metadata.

Example: Attribute `n_name` from node `nation` is labeled as a **level** and accordingly (see Figure 3.7), `nation` is labeled as a node containing dimensional data (i.e., *L*). Furthermore, to propagate that knowledge, we verify any concept association in the WHERE clause in which `n_name` is involved. However, there is no join involving that attribute. If `c_nationkey` had been used in the GROUP BY clause instead of `n_name`, `s_nationkey`

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

Step 3: This step is designed to find explicit factual data. Aggregated attributes in the SELECT clause (see [C1]) play a **measure** role. However, if the input query does not contain a GROUP BY clause we do not have to aggregate **measures** in the SELECT clause, and this step cannot identify them (these types of **Cells** and those not containing **measures** will be identified in Step 6). If the query does not perform a GROUP BY, we store the primary key used as the data cube **base** in the graph metadata (see Figure 3.5 for further details). Finally, we also track the *compatibility* information identified in the node metadata.

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

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

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

Step 5: The previous steps are aimed at creating and labeling nodes and their attributes whereas this step creates and labels edges (i.e., concept associations). Conceptual relationships are depicted in an SQL query by joins in the WHERE clause (see [C1]). In the multidimensional graph joins are represented as edges, and this step is designed to label them following the process described in section 3.3.2.3.

A list of potential edge labels is inferred according to the multiplicity inferred for a conceptual association in the WHERE clause (see Table 3.4). These alternatives are checked prior to labeling the edge, and a label is overlooked if it contradicts current knowledge depicted in the graph. For example, this may occur if a node has already been labeled and the edge label requires it to be relabeled in an incompatible way. An incompatible labeling

happens when a node labeled as *C*, *CM*, *CD* or *CDM* is required to play a dimensional role.

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

Example: First, we infer the relationship multiplicity for each conceptual relationship in the WHERE clause. In this example, each conceptual relationship is defined by a single-attribute join, although in practice they might be depicted by multi-attribute joins; for example, *l_orderkey* = *o_orderkey* represents a relationship between `lineitem` and `orders`. According to Table 3.3 (second row), this join produces a many-to-one relationship between `lineitem` and `orders` that allows zeros in the to-many side of the relationship (since `o_orderkey` is defined as the primary key of `orders` and `l_orderkey` is defined as a foreign key to `o_orderkey`).

Next, according to Table 3.4, this one-to-many relationship may represent a *Level - Level*, a *Cell - Cell* or a *Level - Cell* relationship. However, the *Level - Level* relationship contradicts current knowledge in the graph since `lineitem` has been labeled as *CM* and this edge label requires it to be labeled as dimensional data. In contrast, the *Cell - Cell* relationship is consistent with current knowledge depicted in the graph. Although `orders` has already been labeled as dimensional data in Step 4, according to Figure 3.7 it could also be considered a hybrid node (see the NKD transition), which means that it could also be labeled as either *CDM* or *CD*. In this case, according to Figure 3.5 it should be labeled as *CD* (since the query performs data grouping but there is no `orders` attribute aggregated in the SELECT clause). Finally, the *Level - Cell* relationship is allowed, so the current edge is labeled with both possibilities (*Level - Cell* and *Cell - Cell*). A graphical representation of the multidimensional graph after step 5 can be seen in Figure 3.9.

Once these steps have been completed 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 requested by the user (and nodes to which they belong) will be labeled after Step 5. This is because when writing the SQL query of a given requirement we may need to introduce *intermediate* concepts to relate explicit concepts stated by the user. In general, nodes containing intermediate concepts remain unlabeled after Step 5 (unless they have been labeled by the propagation rule of Steps 2 and 4). In addition, some nodes already labeled after Step 5 may have potentially interesting alternatives, which can occur if the query structure does not clearly identify **measures** (see Step 3) or if we are looking for interesting factless facts. We will refer to intermediate nodes and nodes with interesting alternative labels as *implicit* nodes.

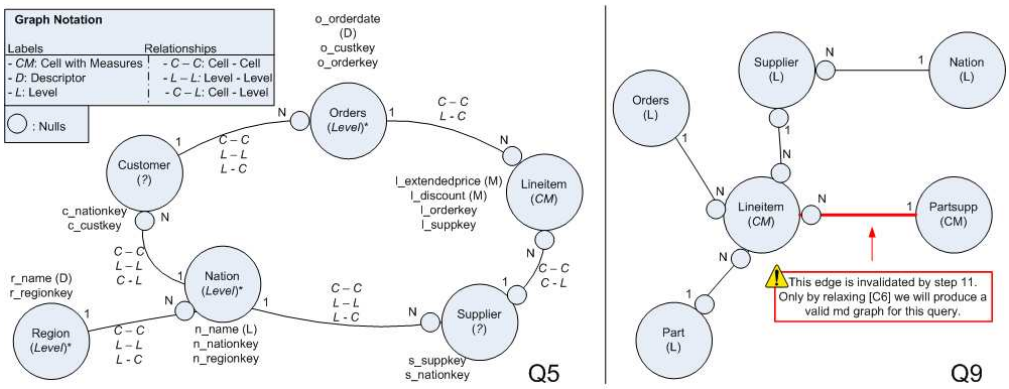


Figure 3.9: MDBE: left, the graph for Q5 after Step 5; right, the graph for Q9 after Step 12

As discussed in Section 3.1.2, demand-driven approaches rely on exhaustive requirements (and thus, they do not rely on the data sources to infer alternative analysis capabilities), whereas supply-driven approaches perform an exhaustive analysis of the data sources and produce too many results. In our approach, we propose a middle ground solution (i.e., we do not fully rely on requirements, but we do perform an exhaustive analysis of the data sources) for automatically derive new multidimensional knowledge not considered by the user. In our approach, we focus on the implicit concepts of the query, and analyze the available labeling alternatives. The aim of this step is to determine how these alternatives would affect the output schema, deriving (in some cases) interesting analytical alternatives that may have been overlooked by the user.

Step 6: This step is designed to derive new multidimensional knowledge from unlabeled nodes or, according to the NKD transitions in Figure 3.5, to test alternative labels for nodes already labeled. Each unlabeled node can be considered to play a dimensional role (i.e., labeled as *L*) or a factual role (labeled as *C* or *CM*, according to Figure 3.5). However, nodes with potential alternatives of interest will introduce an alternative label. For each possible combination of new labels, an *alternative graph* is created if the labels do not contradict knowledge already depicted in the graph. Subsequently, each of these graphs will be validated as explained in Section 3.4.2, and only those that make multidimensional sense will finally be considered. Therefore, a query could produce several valid multidimensional graphs. In that case, MDBE would be able to derive multiple multidimensional schemas for a single query.

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

Example: We have two unlabeled nodes (i.e., `customer` and `supplier`, labeled as *?*

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

Table 3.5: MDBE: graph labelings generated after the first stage of MDBE

in Figure 3.9) and three nodes that, according to Figure 3.7, may play a factual role in addition to their current dimensional role (i.e., `orders`, `nation` and `region`, marked with a * in Figure 3.9). For each combination that does not contradict knowledge depicted in the current graph, an alternative graph is generated.

After Step 6, we have 5 nodes with two potential labeling alternatives that produce 8 different multidimensional graphs. Note that we do not generate 32 graphs (i.e., 2^5 combinations) because many of them are meaningless in the multidimensional model. From Step 5, a given labeling is overlooked if it contradicts knowledge depicted in the graph. For example, consider the following labeling alternative in which `customer`, `orders`, `supplier` and `region` are labeled as *C*, whereas `nation` is labeled as *L*. According to the edge between `region` and `nation`, if `region` is labeled as *C* then `nation` should also be labeled as *C*; otherwise, it would not make multidimensional sense (see Table 3.4). This type of contradiction removes 24 of 32 possible combinations. The remaining 8 combinations (shown in Table 3.5) will then be validated in the second stage of MDBE. As we will see, most of these will be invalidated, and only two will eventually be found to make multidimensional sense (the last column of Table 3.5 shows which step invalidates which combination).

3.4.2 Second Stage: Multidimensional Graph Validation

In this stage we validate each of the multidimensional graphs generated in the previous stage. The validation process also guarantees the *multidimensional normal forms* presented in [LAW98] for validating the output multidimensional schema, and the summarizability constraints discussed in [MLT09]. Again, we use a detailed algorithm in pseudo code (the *validation_process algorithm*) to implement our method, followed by a brief explanation and an example of each one of its steps. This algorithm is called once for each alternative graph generated in Step 6 (see Step 6a of the *MDBE algorithm* in the previous section):

declare VALIDATION_PROCESS **as**

7. **If** $!connected(graph)$ then return $notify\_fail$("Aggregation problems because of cartesian product.");

8. **For each** subgraph of **Levels in the** multidimensional graph **do**

(a) **if** $contains\_cycles(subgraph)$ **then**

i. /* Alternative paths must be semantically equivalent and hence raising the same multiplicity. */

ii. **if** $contradiction\_about\_paths\_multiplicities(subgraph)$ **then** return $notify\_fail$("Cycles can not be used to select data.");

iii. **else** ask user for semantic 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 semantic validation; **Create** context nodes(Cells path);

11. **For each** element **in** $get\_1\_to\_N\_context\_edges\_and\_nodes$(Cells path) **do**

(a) **If** $CM\_at\_left(element)$ **then** return $notify\_fail$("Aggregation problems between Measures.");

12. **If** $exists\_two\_1\_to\_N\_alternative\_branches(Cells\ path)$ **then** return $notify\_fail$("Aggregation problems between Cells.");

Step 7: The multidimensional graph must be *connected* to avoid the "Cartesian Product" ([C6]). Furthermore, the multidimensional graph should be composed of valid edges that produce a path between **Cells** (factual data) and connected subgraphs of **levels** (dimensional data) surrounding it - these constraints will be properly checked in the following steps.

Example: In our example, the 8 multidimensional graphs to be validated (see Table 3.5) are connected.

Step 8: This step validates **levels** subgraphs (i.e., subgraphs only containing **level** nodes) with regard to **Cells** placement: According to [C4], two different **levels** in a subgraph can not be related to the same **Cell** (Step 8b); to satisfy [C5] and [C6], **level** - **level** edges raising aggregation problems in **Cells** with selected **measures** must be forbidden (Step 8c). Finally, every subgraph must represent a valid **dimension** hierarchy (i.e., not being used to select data) [C7]. Thus, we must be able to identify two nodes in the **level** subgraph which represent the *top* and *bottom* **levels** of the hierarchy, and if there is more than one alternative path between these nodes, they must be semantically equivalent (8a). As discussed in Section 3.3.1.1, this step may eventually relax [C5] and [C6] (i.e., disjointness) if required in Step 8c.

Example: i) Step 8a: In our example, none of the graphs contain a cycle within a **level** subgraph so that all of them satisfy this step. ii) Step 8b: Consider the alternative graph

depicted in row 8 of Table 3.5. All of the nodes except for `lineitem` are labeled as **levels**. Therefore, this alternative does not preserve 8b, since `orders` and `supplier` belong to the same **level** subgraph and both are related to `lineitem`. Consequently, the validation process fails and this alternative is discarded because the **Cell** is related to two different points of the same analysis perspective (which does not make multidimensional sense). iii) Step 8c: In our example, `lineitem` is the only node labeled as **Cell** with **measures**, but it does not raise any aggregation anomalies in any of the graphs.

Step 9: **Cells** determine multidimensional data and must be related in the graph to produce a single **Cell** path. If this is not the case, they cannot retrieve a single data cube [C1]. For every pair of **Cells** in the graph, we aim to validate the paths between them as a whole, inferring and validating the multiplicity raised as follows: (i) if a one-to-one correspondence between two **Cells** exists, we replace all of the relationships involved in that correspondence with a one-to-one *context edge* between the two **Cells** (i.e., a context edge replaces the subgraph representing the one-to-one correspondence). As shown in Figure 3.10.1, this means that a whole **Cell** CK is linked by one-to-one paths to the whole CK of the other **Cell**. (ii) Alternatively, if both CKs are related by one-to-many paths or the first CK matches the second one partially, we replace the relationships involved with a one-to-many directed context edge (see Figure 3.10.1). (iii) From a data source perspective, many-to-many relationships between **Cells** should be invalidated because they do not preserve disjointness [C6]. Nevertheless, this step may eventually relax disjointness, as discussed in Section 3.3.1.1.

Example: In our example 3 of the 7 remaining labeling alternatives produce incoherent context graphs that do not satisfy the multidimensional constraints. For example, the labeling alternative shown in the third row of Table 3.5 would produce a forbidden many-to-many relationship between `customer` and `supplier` in the context graph. There are only four viable alternatives for this step: if every node in the graph cycle is considered as factual data (rows 1 and 2 of Table 3.5) or if `orders` (row 6) or `supplier` (row 7) are considered to play a factual role. Any other alternative would produce an invalid context graph.

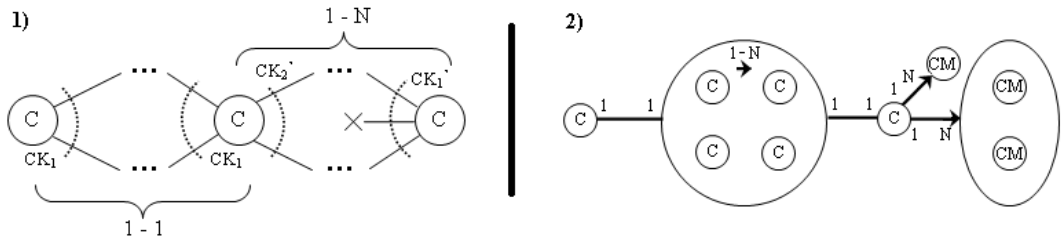


Figure 3.10: MDBE: examples of **Cells** paths in a context graph

Steps 10, 11 and 12: The previous step validated the correspondences between **Cells**, whereas these steps validate the **Cell** path (multidimensional data retrieved) as a whole: According

to [C7], Step 10 validates cycles in the path of **Cells** to ensure that they are not used to select data, similarly to the validation of **levels** cycles (see 8a). Once the cycle has been validated, the **Cells** involved are clustered in a *context node* labeled with the cycle multiplicity, as shown in Figure 3.10.2. According to [C5] and [C6], Steps 11 and 12 look for potential aggregation problems; the first looks for **Cells** with **measures** selected at the left side of a one-to-many context edge or node, and the second looks for alternative branches with one-to-many context edges or nodes each, which would produce a forbidden many-to-many relationship between the **Cells** involved (as depicted in the right side of Figure 3.10.2). Finally, as in any step involving [C5] and [C6], this step may eventually relax disjointness as discussed in Section 3.3.1.1.

Example: i) Step 10: In our example, this would be the case if every node in the graph cycle was considered to play a factual role (row 1 of Table 3.5) or if all of them except `region` played a factual role (row 2 of Table 3.5). In both cases, identified cycles would not make multidimensional sense because they do not preserve disjointness of `lineitem` (which contains **measures**). Consequently, both alternatives are discarded. ii) Steps 11 and 12: At this point, we only have two valid alternatives (rows 6 and 7), but neither produces aggregation problems with `lineitem`.

Another interesting example is the graph obtained from Q9 (see Figure 3.9), in which a one-to-many relationship is shown between two **Cells** (`lineitem` and `partsupp`). According to Table 3.4 this edge is allowed between **Cells** but Step 11 invalidates it. This query does not preserve disjointness [C6] between both **Cells** with **measures** (a **Cell** with **measures** is selected at the left side of a one-to-many context edge). In this case, no result would be produced and, according to R1 (see Section 3.3.1.1 for a detailed definition of this rule) we relax [C6]. Now, we obtain a valid multidimensional graph, but the user is informed of the situation and asked to validate the result obtained. In this example, according to the query semantics, we are calculating a derived measure (weighting the `l_quantity` with the `ps_supplycost`), which determines the profit made on a given line of parts (see [Tra09]). Clearly, this derived measure (and thus, this query) makes sense and must be considered a valid multidimensional data cube.

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

Example: At the end of the validation process, two of the eight initial labeling alternatives (see rows 6 and 7 of Table 3.5) are sound and make multidimensional sense. Therefore, MDBE would produce two different multidimensional schemas that would satisfy Q5 (these can be determined from Figure 3.9 and rows 6 and 7 of Table 3.5).

3.4.3 Third Stage: Finding Representative Results

Step 6 in the first stage of the MDBE method may produce several alternative graphs for the same query. Unlabeled nodes (and those with interesting alternatives, according to Figure 3.7) are proved to be factual and dimensional data in alternative graphs, which are validated in the second MDBE stage. Eventually, those graphs that make multidimensional sense will be considered

in the conciliation process (see next section). Consequently, more than one multidimensional schema can be produced for a given query. However, an alternative graph could make multidimensional sense but not represent a new and potentially interesting analytical perspective. Indeed, dimensional data could always be considered as an alternative factless fact, although in most cases it will not be relevant to the end-user. Therefore, this step is designed to determine the representativeness of new alternatives produced by Step 6, according to the following rule:

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

Two sibling graphs differ only in the labeling of one node. Therefore, they have exactly the same labels except for one node, which is considered to play a factless fact role in one graph and a strict dimensional role in the other. As an example, consider the following table, which depicts the alternative graphs obtained after the validation step for a given query:

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

According to the previous definition, alternative Graphs 1 and 2 (which only differ in the label of B), and Graphs 2 and 3 (differing in the label of D) are siblings. In this case, and according to $R2$, for the first sibling relationship we disregard the first graph and choose Graph 2 as the most representative; for the second pair we disregard Graph 3 and choose Graph 2 again. Eventually, this query will produce a single multidimensional schema. In short, sibling graphs do not provide new interesting analytical perspectives. MDBE uses them to analyze the potential factual data that a **dimension** may contain. However, in most cases, the end-user would not be interested in this type of analysis. Knowledge inferred from Step 6 is therefore disregarded when it produces sibling graphs and is only considered and presented to the user in one of the following two cases: (i) Firstly, if we identify a dimensional node that may also play a factual role with **measures**. This scenario can only arise in a query without data grouping, in which case Step 6 would identify an atomic **Cell** with **measures** (see Section 3.3.2.2 for further details). Note that this type of node is relabeled in Step 6 as *CDM* and, as such, does not fit the sibling definition (since the alternative sibling graph labeling that node as *L* will be missing) and will not be pruned in this step. (ii) In the second possible case, we have a factless fact that cannot play a dimensional role (i.e., there is no sibling graph for this labeling).

Example: The latter case (ii) occurs in the Q5 validation process. Consider Table 3.5. The two valid labels are shown in rows 6 and 7; since they do not have sibling graphs, none will be pruned in this step. Essentially, MDBE highlights that Q5 will make multidimensional sense if either `supplier` or `orders` plays a factless fact role (the query semantics can be checked to confirm that this is consistent with the query definition).

3.4.4 Fourth Stage: Conciliation

MDBE validates each input requirement and obtains a potential set of multidimensional schemas for each query (see the three previous stages presented above). In this section we present an

algorithm that conciliates the results for the input queries into a minimal set of schemas covering all of the queries.

Before proceeding to the conciliation, a pre-process must be carried out to *normalize* the multidimensional graphs; each hybrid node in every multidimensional graph is normalized. This means that any node labeled as *CDM* or *CD* will produce two different nodes: according to the discussion introduced at the end of Section 3.3.2.3, hybrid nodes contain factual data (and thus, like any other **Cell**, the *link attributes*) and additional dimensional data (that in the general case will introduce redundancy of data). In fact, hybrid nodes could be represented as factual data (i.e., a node labeled as *CM* or *C*) related by a many-to-one (or in a degenerate case, a one-to-one) relationship to dimensional data (i.e., a node labeled as *L*); in other words, we could normalize them. We then apply the following algorithm (for clarity, the comprehensibility of the algorithm took priority over its performance):

- (1) MDBE looks for all the facts identified in the multidimensional graphs, and creates a new factual class[4] for each one (every class will eventually produce a multidimensional schema at the end of the conciliation process). Two other tasks are performed in this step: i) we enrich each class by adding the measures identified in the graphs as attributes of the factual class; and ii) we draw the conceptual relationships between facts depicted in the graphs by semantic relationships between classes.

  Example: Consider a simplified scenario of the TPC-H case study in which we only need to conciliate the multidimensional graphs created for Q5 and Q9 (see Figure 3.9). First, we create four factual classes (`lineitem`, `orders`, `supplier` and `partsupp`) for each node labeled as either *CM* or *C*. Then, we add the measures identified in these graphs to each class. Consequently, `l_extendedprice`, `l_discount` (from Q5 and Q9) and `l_quantity` (from Q9) are added to the `lineitem` class and `ps_supplycost` (from Q9) is added to `partsupp`. The remaining classes will not contain measures as they were identified as factless facts (see Q5).

  In addition, since `partsupp` is related to `lineitem` in Q9, we keep track of this conceptual relationship by drawing a semantic relationship between the two classes. The same is done with `lineitem` and `order`, and `lineitem` and `supplier` (Q5).

- (2) Next, we conciliate the dimension hierarchies identified by the input queries. We first look for compatible hierarchies. Two hierarchies are compatible if they share their *atomic level*[5]. Every set of compatible hierarchies must be conciliated (i.e., produce a single dimension subsuming all of them). This process is carried out by checking the *hierarchies graphs*. From the perspective of the multidimensional graph, a hierarchy is represented by the subgraph containing the nodes that form the dimension. For example, the `customer` → `nation` → `region` hierarchy identified in Q5 (see Figure 3.9 and Table 3.5) is directly derived from the subgraph formed by these three nodes.

  Therefore, a hierarchy *h subsumes* a hierarchy *h'* if the subgraph representing *h'* is contained (except for the descriptors) in the subgraph representing *h*. At this point, it should

[4] In this step we are devising the multidimensional conceptual schema. We therefore talk about classes and attributes in this section, but we could use the notation from any conceptual multidimensional model. For example, [ASS06].

[5] An atomic level is the finest granularity level within a dimension hierarchy and is directly related to the fact [ASS06].

be noted that a one-to-one relationship is contained in a one-to-many or a many-to-one relationship. Having said that, we conciliate a set of compatible dimensions by applying the following properties iteratively:

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

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

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

- (3) Finally, a pruning step is carried out. MDBE identifies those star schemas that are semantically poor. We can also introduce a non-representative requirement, which would produce an unneeded star, for example: *every star schema composed of just one dimension is proposed to be disregarded* (note that we could use any other criterion introduced in the literature [SKD07, RA07a], if desired). However, the final decision is taken by the user, since the star schema is derived from the end-user requirements and he/she must decide if it really makes sense or it was an error.

  Example: In the TPC-H case study, this would be the case of `supplier` and `customer`. Both have been identified as factless facts during the process, but their star schemas are rather simple (one **dimension** each). After considering the requirements from which they

were derived, we may decide to eliminate them (as was the case in the final schema shown in Figure 3.3).

Two main points should be made about this process. First, it does not introduce a *summarizability* problem, because we are only merging compatible labels (i.e., factual data and only fully compatible dimensional data). It is also very important to note the relevance of semantics in the conciliation process. In a data warehousing design task, semantic relationships must be carefully considered. For example, two different relationships between the same concepts *A* and *B* must produce two different perspectives. The reason is clear: each relationship relates a different set of instances from the two classes and, therefore, produces two different analytical perspectives. This explains why two dimension hierarchies such as A $\rightarrow^{r_1}$ B (where r_1 identifies the relationship between A and B) and A $\rightarrow^{r_2}$ C $\rightarrow^{r_3}$ B cannot be conciliated as A $\rightarrow^{r_2}$ C $\rightarrow^{r_3}$ B. Had we proceeded like this, we would have lost semantics. It should be considered that we are working with relational sources, so if we travel from A to B along two different paths there must necessarily be two different conceptual paths between them. As explained in Step 2.3 of the conciliation process, the hierarchies should be conciliated as: B $\leftarrow^{r_1}$ A $\rightarrow^{r_2}$ C $\rightarrow^{r_3}$ B; i.e., with two alternative branches starting from A (the common part).

The second point is that the *orthogonality* of the multidimensional spaces that may be produced is not lost, since we keep track of the metadata inferred from each query at the constellation level (see section 3.4.1; steps 2 and 3). Note that each input query represents a data cube of interest. Consequently, our output schema retains the metadata about these datacubes: the multidimensional space depicted (i.e., the cube base) and the information about the compatibility of the data summarization performed (i.e., which function may be used for their measures and in which dimensions). This type of information will be relevant for the OLAP tool once it has been implemented.

Finally, this stage, like the three previous stages, is fully automatic and we therefore obtain a star schema for each fact identified; this, as a whole, produces a constellation schema (see Figure 3.3). Note that this figure only shows **facts**, **measures** and **dimension hierarchies** identified in the process, whereas **descriptors** have been overlooked to avoid disrupting the final result. Nevertheless, it should be stressed that MDBE works at the attribute level and keeps track of the role assigned to each attribute when deriving partial schemas from each query. Consequently, we are able to split some tables (for example, `orders` produced two different concepts in the multidimensional schema, since the dimensional attributes contained in the relational `orders` table are represented explicitly in `orders_dim`).

3.5 A Practical Case: The TPC-H

In this section we discuss several issues about the overall TPC-H case study. MDBE was carried out for the 22 TPC-H queries that together produced the constellation schema shown in Table 3.3. Below, we focus on five interesting aspects of this case study: the specificity of requirements needed, the expressiveness and quality required in the sources, the degree of automation achieved, the computational complexity of the algorithm and the quality of results obtained (i.e., the output correctness and the extra knowledge obtained in the output thanks to the novel contributions of MDBE). Note that our study is exhaustive regarding the four axis discussed in Section

1.7, and we also provide a study of the performance (and thus, feasibility, of our proposal). Later, we will also use this case study to provide a comprehensive framework in which compare the MDBE and AMDO methods.

3.5.1 Requirements Specificity

MDBE requires to gather the end-user informational requirements and formalize them into SQL queries. Thus, one interesting aspect deserving further study is the number of queries needed to produce the resulting conceptual schema. In the output schema we identify 3 factual classes (containing 9 measures) and 9 dimension hierarchies (containing a total of 18 level classes and 39 descriptors):

- In the worst case, we would need 11 queries to identify all of the factual classes and dimension hierarchies in the multidimensional model. In other words, some queries are redundant and are not relevant to the final result. Had we executed them in the worst possible order, we would have identified all the multidimensional classes and most of the attributes even with 11 queries (8 out of 9 measures and 17 out of 39 descriptors).

- In contrast, in the best case, we would have been able to identify all of the factual classes and dimension hierarchies with just 4 queries (and 6 out of 9 measures and 10 out of 39 descriptors -it would also be possible to give more relevance to descriptors and then identify 16 out of 39 descriptors but 5 out of 9 measures, also with 4 queries-). For example, Q5 is a key requirement as it identifies 4 dimension hierarchies and 2 factual classes. Indeed, Q5 and Q9 identify all three factual classes and 4 measures of the resulting multidimensional schema.

If we considered a random order of input queries (i.e., without any consideration other than choosing the order of the query execution at random) we would need an average of 8 queries (i.e., the average of the worst case -i.e., 11 queries- and the best case -i.e., 4) to identify the main structure of the schema (i.e., facts, measures and dimension hierarchies). This result is sound as it is relatively easy to identify the multidimensional classes with only a small number of queries. Indeed, the multidimensional design task proposed in this chapter is incremental, and it is up to the user to decide when to stop adding new queries. Once most of the structure has been defined, it can be customized as in traditional approaches.

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

To continue with our example, at this point the `region` level class would contain `r_regionkey` and `r_name` but the `r_comment` attribute in the relational table would not have been selected

yet. However, it would be easier to browse the region attributes and add r_comment to the output schema than to launch a new query specifically for the purpose.

3.5.2 Data Source Expressiveness

The TPC-H relational schema is well-formed and captures a fair picture of the business domain. For example, foreign keys are used to identify semantic relationships between attributes in different tables, and the schema is in 3NF.

Importantly, note that MDBE is an interleaved hybrid approach, which analyzes the end-user and the data sources simultaneously, but the requirement analysis leads the process. Consequently, the quality of the sources required in our approach is considerably lower than in previous approaches working from relational sources. Indeed, as discussed in Section 2.1, current approaches demand that the source relational schemas capture the functional dependencies (i.e., to-one relationships) existing in the domain. This kind of relationships, typically represented at the logical level by means of foreign and candidate key constraints, are crucial to identify the multidimensional concepts and specially, the dimensional concepts. For this reason, the quality of the output obtained by current approaches decreases drastically for relational sources between a denormalized schema and a logical schema in 3NF. On the contrary, MDBE is able to produce high-quality results, even from denormalized sources, by means of two key features:

- MDBE exploits the candidate - foreign key knowledge captured in the data sources. Nevertheless, if this information is missing, we are able to extract it from the requirements (in case it is relevant for the final result). For example, consider the TPC-H relational schema introduced in Figure 3.2, and the TCP-H #5 query used as example all over the chapter. In Q5, the c_custkey = o_custkey logic clause involves two concepts related by means of a primary - foreign key relationship and thus, it can be exploited by most of the current methods. However, if we discard the candidate - foreign key relationships in the schema, none of these approaches would be able to exploit it anymore. Relevantly, this does not affect MDBE, since we consider requirements. Indeed, SQL query joins represent concept associations explicitly stated by the user and thus, we are able to exploit them even if the attributes joined are not explicitly related in the sources. Consequently, the multidimensional knowledge inferred for attribute involved in joins in the WHERE clause is automatically propagated to its counterpart (see Section 3.1.2 for further details). Specifically, the orders node is initially labeled as a level in Step 4 (see Section 3.4.1) and therefore, its attributes involved in the query are identified as dimensional concepts (i.e., o_orderdate, o_custkey and o_orderkey). Thus, by means of the c_custkey = o_custkey join, we propagate the knowledge inferred for o_custkey to c_custkey: i.e., it is also identified as a dimensional concept. Interestingly, later, Step 6 proposed orders to play a Cell role as well. This alternative is not prune in the third stage of the algorithm (see Section 3.4.3) and eventually, MDBE produces two results for the Q5 query (see Section 3.4.2). However, even in this case, o_custkey and c_custkey will still play a dimensional role: in this scenario, o_custkey would have not been identified as a dimensional concept, but customer is labeled as level and accordingly, c_custkey as dimensional concept. Consequently, o_custkey is identified

as a dimensional concept by means of the association in the WHERE clause. This is sound, since MDBE is identifying o_custkey as orders *link attribute* (see Section 3.3.2) and thus, it is part of the dimensional concepts forming the multidimensional space.

- Furthermore, MDBE smooths the impact of denormalization on the output produced. Consider now a unique relation capturing the whole TPC-H relational schema (i.e., the TPC-H *universal relation*); i.e.:

```
TPC-H(lineitem_attrs, orders_attrs, partsupp_attrs, part_attrs, supplier_attrs,
                customer_attrs, nation_attrs, region_attrs)
```

where lineitem_attrs refers to the whole set of attributes in the lineitem relation, and similarly for the rest. Functional dependencies would not be extracted from such a relation, and current approaches (i) would not be able to identify any dimensional concept or (ii) they would produce loads of meaningless results. On the contrary, MDBE is able to identify dimensional concepts from such a relation by means of the end-user requirements (see Section 3.1.2 for further details). For example, consider the TPC-H Q5 business query over the universal relation introduced above:

```
Select nation_name, sum(lineitem_extendedprice * (1 - lineitem_discount) as revenue
FROM TPC-H
WHERE region_name = '[REGION]' and orders_orderdate >= '[DATE]' and
orders_orderdate < '[DATE]' + '1' year
GROUP BY nation_name
ORDER BY revenue desc;
```

In this case, the multidimensional graph would be compound of just one node (i.e., TPC-H), which would be labeled as *CDM*, since nation_name, region_name and orders-_orderdate would be identified as dimensional concepts (see Steps 2 and 4 in Section 3.4.1), and lineitem_extendedprice and lineitem_discount as measures (see Step 3 in Section 3.4.2). However, regarding dimensional data identified from this kind of relations, MDBE cannot automatically generate the dimension hierarchies, since requirements provide additional knowledge about the role played by each attribute but, under no circumstances, knowledge about the missing to-one relationships (i.e., functional dependencies) is provided. For example, considering just the dimensional concepts identified for Q5, and according to the requirements stated in the TPC-H benchmark, we should manually form the place dimension (in which nation_name can be aggregated into region_name) and the order_date dimension. We can only overcome this drawback by mining the instances, but mining the instances is computationally expensive (see [JHP04], which already proposes to mine the instances to identify functional dependencies) and can be unfeasible for large databases. Finally, note that, although MDBE does not generate the dimension hierarchies from denormalized sources, it does identify the dimensional concepts and therefore, they do not have to be derived from scratch, but from the set of dimensional concepts identified for the fact (i.e., we do not shape dimensions by exploiting all the to-one relationships in the schema, but only those between concepts identified as dimensional concepts).

3.5.3 Automation

The automation degree obtained in our approach is, in the worst case, as good as in equivalent approaches. Importantly, the whole process is automated once the requirements are expressed into SQL queries (the reader will note that there is no approach automating the end-user requirement elicitation process and thus, no automated counterpart can be used for this pre-process), and the user is only needed in the following cases:

- According to rule R1, introduced in Section 3.3.1.1, if, for a given query, MDBE cannot produce any output, our approach tries to identify relevant derived measures or concept specializations by relaxing [C5] and [C6] respectively. In this case, if any result is generated, the user must validate the derived measures or concept specializations generated. In the TPC-H case study, only two queries (TPC-H #9 and #12 queries) required to relax these criteria and thus, the user is only asked to validate two queries out of the 22 used as input.

- If [C7] is relaxed, we may allow selections by means of joins. In an OLAP tool, this scenario can only be considered if the selection done through the *join paths* are equivalent. Otherwise, the selection would not make multidimensional sense (see Section 3.3.1.1 for further details). Thus, if [C7] is relaxed, the user is responsible for validating the join paths stated in the query as equivalent. For example, following the example introduced in Section 3.3.1.1, MDBE would ask the user to validate if the `lineitem - orders - customer - nation` join path is equivalent to the `lineitem - partsupp - supplier - nation` one. If the end-user guarantees that they are equivalent (for example, if a business constraint guarantee that `customers` are only supplied with `supplier` from their own country) then, MDBE automatically rewrites the query to make multidimensional sense (by using two different alias for the `nation` table). Otherwise, it is discarded.

- In case of dealing with denormalized data sources, the user is asked to shape the dimension hierarchies, by arranging the dimensional concepts identified. In the TPC-H case study this does not hold, and dimension hierarchies are automatically derived. Regarding the universal relation example introduced in previous section, it would embrace shaping the `place` and `order_date` dimensions manually.

In the first case, note that MDBE relaxes [C5] and [C6] regarding the data sources, and proposes derived measures or concept specializations, which guarantee the completeness and disjointness of the result proposed. However, since these new measures or specializations are not explicitly captured in the sources, only the user can validate them, and his / her participation is compulsory. Similarly, the second case can only be guaranteed by the user, since the semantics of each path are not captured in the data sources. Finally, in the latter case, there is no alternative to infer this knowledge from the logical schema. Indeed, this is inherent to relational schemas that, in the general case, are semantically poorer than conceptual schemas and therefore, relevant knowledge about the domain may be missing. Consequently, all the approaches working at the logical level suffer from this drawback.

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

Table 3.6: MDBE statistics for the TPC-H case study

3.5.4 Computational Complexity & Performance

Finally, we discuss our approach feasibility by presenting an in in-depth analysis of the 22 business queries in the TPC-H benchmark, and use the findings as the basis for discussing the complexity and performance of the MDBE algorithm. Table 3.6 summarizes some of the relevant statistics for each query. Statistics for their subqueries, if any, are shown in brackets (briefly, subqueries must be validated by their own, as they can be considered a materialized factual table and must therefore make multidimensional sense as well). The first column represents the query id and the other columns should be read as follows: the second column shows the number of implicit nodes we have for the query (i.e., nodes that remain unlabeled up to Step 6 or which are relabeled at that point). According to the number of implicit nodes, we can produce $2^{\#implicit\ nodes}$ label combinations (note that Step 6 only tries two label alternatives for unlabeled nodes). However, as discussed previously, many of these combinations are not even generated, since they raise contradictions with knowledge already depicted in the graph and, therefore, do not satisfy the multidimensional constraints. Ungenerated combinations are shown in the third column, and the fourth column shows the number of many alternative graphs (to be validated) generated for each query. The fifth column shows the number of multidimensional graphs that are discarded in the MDBE validation stage, and the sixth column shows the number of graphs that are are collapsed, according to the sibling rule introduced in Section 3.4.3.

The MDBE tool execution time for the TPC-H benchmark is negligible ($\sim$ 1 second[6]). Our approach only has a potential combinatorial explosion in Step 6 (note that the conciliation process carried out - see Section 3.4.4- is linear regarding the number of schemas obtained and, therefore, it does not raise the computational complexity of the MDBE process). However, most combinations of labels generated by Step 6 are discarded on the basis of edge semantics (see the third column), which produces a tractable algorithm. In all queries, the final set of graphs to be validated is considerably smaller than $2^{\#implicit\ nodes}$ (see the fourth column). This statement is based on the empirical results provided, but we can intuitively identify why Step 6 will never generate an exponential number of combinations: the whole multidimensional graph must be semantically valid, which means that several nodes and edge labelings will not be allowed. For example, Table 3.4 shows 25 forbidden combinations (we count those allowed by relaxing [C5] and [C6], as they will only be considered if no result is generated. Thus, if considered, we obtain just one result at most). Consequently, many combinations of labels will fail to make multidimensional sense. Furthermore, the first five steps of the MDBE process always label most of the nodes/edges for multidimensional requirements. Only *implicit* nodes (see the discussion prior to Step 6 in Section 3.4.1 for further details) can produce unlabeled nodes. Consequently, the exponent value in the $2^{\#implicit\ nodes}$ expression will be typically a small number. For example, in the statistics shown, only two queries (Q7 and Q8) have more than 6 implicit nodes. However, Q7 invalidates 51 of 64 alternative graphs (and is computed in $\sim$0,1 s) according to edge semantics, and 225 of 256 in Q8 (computed in $\sim$0,12 s). Let us consider a query with a large number of implicit nodes. In this case, we will only generate all possible combinations of labels (i.e. exponential computational complexity) if these tables are related by one-to-one relationships with a double FK pointing between each pair (see Tables 3.3 and 3.4). However, this would be an unlikely real-world scenario and, in any case, an SQL query is unlikely to have a large number of tables in its FROM clause. The MDBE validation process takes an average of 0.007 s, so in the worst-case scenario discussed above a query with 10 unlabeled tables in the FROM clause would generate 1024 label combinations, which would be processed in 7.168 secs.

3.5.5 Output Quality

In this section we measure the quality of the output produced by MDBE. We do so by means of the result correctness (by comparing the output obtained with the multidimensional schema proposed in the Star Schema Benchmark), and the additional output inferred regarding both, the Star Schema Benchmark and previous approaches.

3.5.5.1 Output Correctness

The Star Schema Benchmark (or SSB) [P. 09] presents a multidimensional logical schema that is derived *manually* from the TPC-H schema. This schema was devised to improve the querying performance of the data warehouse by denormalization. Data denormalization, achieved by implementing a logical star schema [KRTR98], is fairly common in data warehouse systems and is used to speed up certain queries [KRTR98]. Unlike SSB, the MDBE method produces a conceptual schema, but the SSB logical schema can be obtained by applying the same design decision

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

taken by the SSB authors: to implement the output schema as a logical star schema (i.e., denormalize dimensional data as much as possible). Therefore, we can obtain the SSB *logical* schema from the MDBE output as follows:

- By denormalizing the **dimensions** of analysis as much as possible. Therefore, `nation` and `region` data will be denormalized within `supplier`, `partsupp` and `customer`. Dimensional attributes that produced new level classes (i.e., `orders_dim`, `lineitem_dim` and `partsupp_dim`) must also be denormalized.
- By merging the three schemas that form our constellation (since we have three factual classes) into a single schema. Therefore, `lineitem`, `orders` and `partsupp` will produce a single table. This decision is consistent with our conceptual schema, which relates these three facts (therefore, our conceptual schema allows us to "drill-across" [RA07b] between them, obtaining the same results as if they were merged in a single table).

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

3.5.5.2 Additional Output Inferred

In this section we measure the impact of the main contributions of MDBE on the output. The first six columns of Table 3.6 show statistics about the MDBE process, whereas the last three show statistics about the results for each query. The seventh column shows the number of final star schemas retrieved by MDBE for the corresponding query (i.e., the number of alternative graphs that have been completely validated). The next column shows the number of factless facts identified for the query, and the final column shows the number of new dimensional attributes identified in the process (if any). These attributes are those identified as dimensional data in a hybrid node (i.e., dimensional attributes in *CD* and *CDM* nodes). The † symbol denotes that [C5] or [C6] have been relaxed to produce the output result for that query. These results highlight some interesting features of our method:

- First, the MDBE validation process. In this stage, 14 of 22 queries invalidate alternative graphs that may initially appear to be correct. Consequently, simply labeling nodes is not sufficient and it is necessary to consider the semantics of the results proposed as a whole.
- Our process for obtaining new knowledge from implicit nodes is carried out in most of the queries. In the TPC-H case study, the sixth step of our method labels (or relabels) nodes in 20 of 22 queries (see the second column), which reveals the importance of this step in retrieving additional information to that explicitly requested by the user.
- Denormalization is very important in our approach. Although the TPC-H logical schema is normalized and well-formed, from a multidimensional perspective some of its tables contain degenerated dimensions, as a result of which many nodes have been labeled as *CD* or *CDM* during the labeling process. For example, in one of the solutions proposed for Q5, `orders` is labeled as a factless fact *CD* with `o_orderdate` as a degenerated dimension [KRTR98]. This result is sound, since `time` and `date` are typical analysis **dimensions**

in any data warehouse and, in fact, some current methods always complement their results with these two **dimensions** (e.g., [PD02]). Therefore, in our final result these concepts are explicitly stated according to their multidimensional role. In our example, 15 of 22 queries identify at least one new dimensional attribute (i.e., not identified in the other queries) for the corresponding query (see the final column); for example, `shipdate`, `returnflag` and `shipmode` from `lineitem`. In addition, we may also identify *dual* attributes (see Section 3.3.2.1); for example, `ps_supplycost` from `partsupp`. Consequently, the resulting conceptual schema contains a **dimensional attribute** and a **measure** derived from this relational attribute.

- Our process can identify interesting additional information that is traditionally overlooked by other methods. Our method supports factless facts (see column 8) and new derived measures (see †) and specializations (none identified for the TPC-H case study) that are not captured in the relational sources but which can be derived from them.

3.6 The MDBE Tool

As depicted in Figure 3.11, 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 3.11 (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 3.12 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.

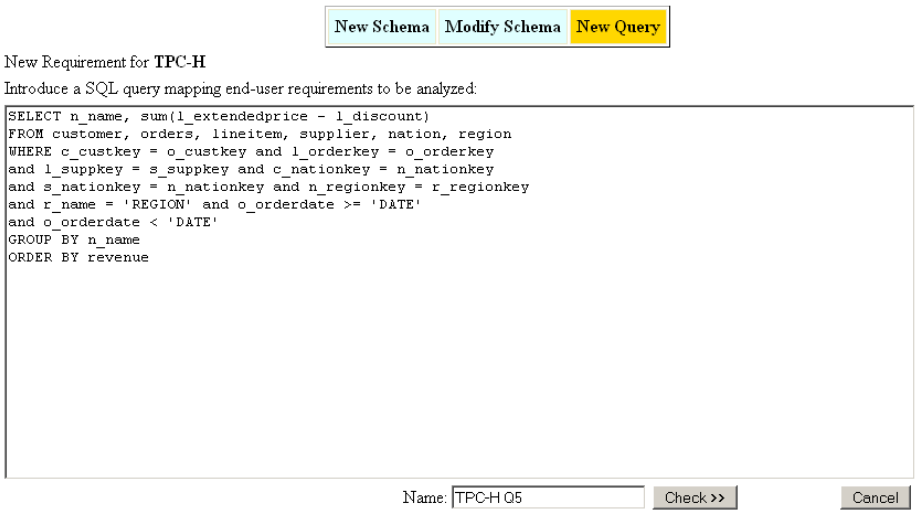


Figure 3.11: The MDBE tool: uploading the TPC-H Q5 query

3.7 Conclusions

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

We also demonstrated the practical application of our method, using the TPC Benchmark H case study to illustrate the potential of the approach and to provide a detailed example of how the method is executed. The MDBE method opens up a range of new research opportunities. For example, our approach is incremental and provides a solid foundation for the maintainability and evolution of the conceptual schema, which is a topic that has gained importance in recent years [RALT06].

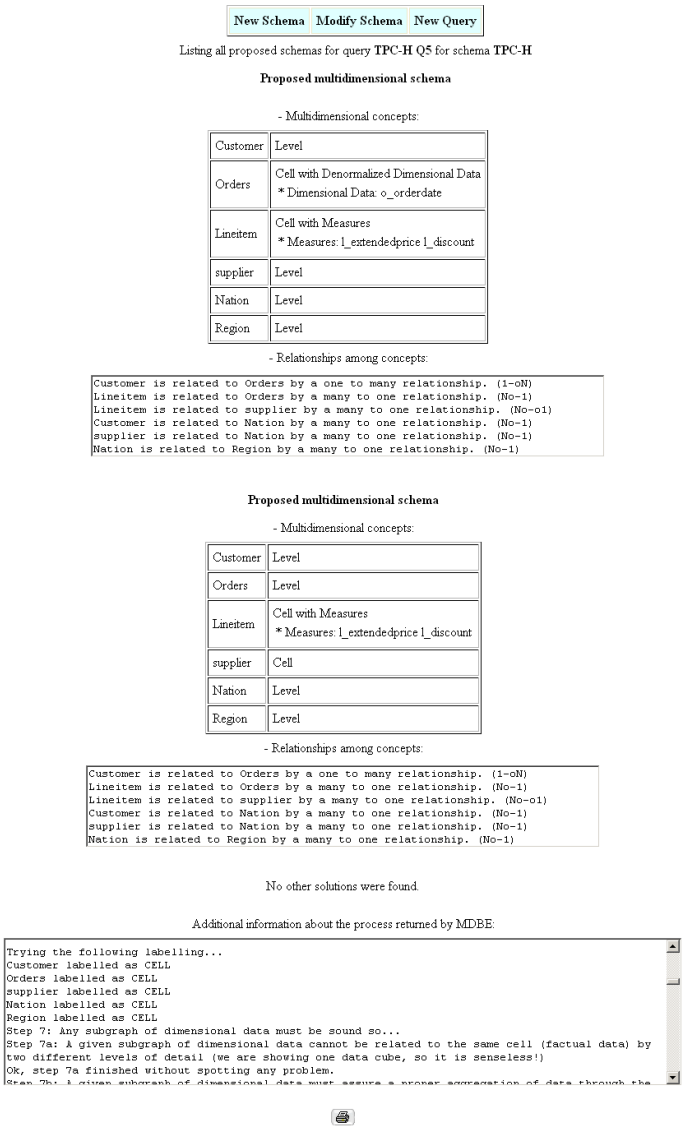


Figure 3.12: The MDBE tool: results retrieved for the TPC-H Q5 query

Chapter 4

Multidimensional Design from Ontologies

> " There is nothing like looking, if you want to find something. You certainly usually find something, if you look, but it is not always quite the something you were after."
>
> J.R.R. Tolkien

Some research efforts have proposed the automation of the data warehouse design in order to free this task of being (completely) performed by an expert, and facilitate the whole process. Mostly, these approaches carry out this process from relational OLTP (*On-Line Transaction Processing*) systems, assuming that a relational database management system is the most common kind of data source we may find, and taking as starting point a relational schema (i.e., a logical schema).

Starting from a logical schema, however, may present some inconveniences. A logical schema is tied to the design decisions made when devising the system and these decisions either made to fulfill the system requirements (for example, improve query answering, avoid insertion / deletion anomalies, preserve features inherited from legacy systems, etc.) or naively made by non-expert users, have a big impact on the quality of the multidimensional schemas got by current automatable approaches. In fact, these approaches require a certain degree of normalization in the input logical schema to guarantee that it captures as much as possible the to-one relationships (i.e., a specific case of *functional dependency*, as discussed later) existing in the domain. As shown in Section 2.1.4, discovering this kind of relationships is crucial in the design of the data warehouse, and the most common way to represent them at the logical level is by means of "foreign" (FK) and "candidate key" (CK) constraints. This scenario can be clearly seen in the example shown in Figure 4.1.

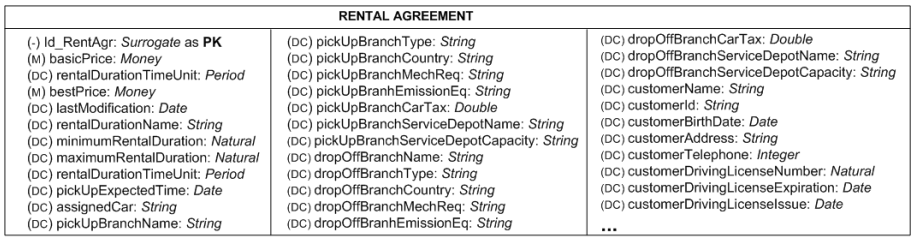


Figure 4.1: A fully denormalized relational schema of a car rental agreement

There, a single relation (namely, `rental agreement`) models data related to a `car rental agreement` in a relational database management system (RDBMS). Each row represents an attribute of the relation (in *italics* its data type). The relation primary key is identified by the *PK* label and the capital letters in brackets next to each attribute represent the multidimensional role that attribute should play according to its semantics (*M* stands for *measure*; i.e., interesting business measures of our fact of study, and *DC* for *dimensional concept*; i.e., interesting perspectives of view of our fact -a detailed definition of the multidimensional concepts may be found in Section 1.5-). Only those concepts that would play a meaningful role in the multidimensional schema are shown in the figure, but additional attributes could be found in the relation (depicted by the ellipsis at the end). In this case, current methods would either i) overlook all the dimensional concepts (since they are not involved in any CK or FK), or ii) identify all the non-numerical attributes as dimensional concepts (i.e., even those not making multidimensional sense and not shown in the figure). Furthermore, even if they were able to identify any dimensional concept they would not be able to identify potential aggregation paths (or *roll-up* relationships) that would give rise to *dimension hierarchies*. Thus, they are not able to answer the following questions: is each dimensional concept conforming a dimension by itself? which of them would form the same dimension hierarchy (i.e., which are *levels* and which *descriptors* within the same dimension)? which belong to the same dimension and which to dimensions semantically related?

In this sense, note that MDBE (see Chapter 3) partially overcomes this major drawback by considering end-user requirements as first-class citizens. Indeed, the analysis of requirements allows MDBE to identify dimensional or factual attributes that the other approaches would overlook. However, regarding dimensional data identified from denormalized relations, MDBE cannot automatically generate the dimension hierarchies as the domain FDs needed to shape hierarchies are missing in the source schema (see Section 3.1.2). In short, requirements provide additional relevant knowledge, but the relationships between these attributes cannot be extracted from denormalized data sources.

Dimension hierarchies are crucial in the multidimensional model which is based on two main features: (i) placement of data in the multidimensional space and (ii) summarizability of data (see Section 3.2.3 for further details). A bad design of the dimension hierarchies would directly impact on the aggregation paths we may have. Modify data granularity when showing data to the user is a key feature of OLAP tools (performed through the roll-up and drill-down operators; see Section 2.2.1). Thus, overlooking aggregation paths in the design task would impact on the

success of the whole system. Indeed, any intermediate situation between a denormalized schema and a logical schema in 3NF would affect the output quality of current multidimensional design methods.

This scenario can be avoided by modeling the data warehouse from a conceptual formalization of the domain. The role of a conceptual layer on top of information systems has been discussed in depth in the literature (see, for example, [Oli04]). In case of reengineering processes, like the data warehouse conceptual design, the benefits are clear: the conceptual layer provides more and better knowledge about the domain to carry out this task. For example, consider now the ontology represented in Figure 4.2[1]. This ontology plays a conceptual role regarding the logical implementation depicted in Figure 4.1. The piece of ontology depicted in the figure (that will be used as example in this chapter) refers to a `car rental agreement` between a `branch` and a `costumer`. For a given `rental agreement` (which can still be ongoing -i.e., an `opened rental`- or already closed -i.e., a `closed rental`- and / or be booked by `reservation` with or without `guaranteed canceled`), a `car` is `assigned`. Several information about the `branch` is captured, such as `pendant car models` to be assigned to `rental agreements`, the `demand` for a given kind of `car group` or the `service depot` associated to a `branch`. Moreover, a `car` belongs to a `branch` and it is assigned to a `service depot` when `maintenance needed`. There, each relevant concept of the domain is clearly stated as well as its relationships with the other concepts, and for example we will be able to propose a `car` to be summarizable into two different aggregation paths (into `car model` and `car group` but also through the `branch` path up to the `country`, `branch type` or `service depot` it belongs to), that would form, as a whole, the `car` dimension.

In this chapter we introduce AMDO (*Automating Multidimensional Design from Ontologies*), our approach for automatically deriving the multidimensional schema from a domain ontology. Our goals are mainly two: i) we want to improve the quality of the output got (by working over a conceptual formalization of the domain instead of a logical one) and ii) we want to automate the process. This second goal is the main reason for choosing ontologies instead of other conceptual formalizations, as ontology languages provide reasoning services that will facilitate the automation of our task. Our work, however, is not tight to a specific ontology language. In general, we assume OWL DL [W3C], a W3C recommendation, as our input ontology language, but we show later that any Description Logics (DL) language providing the necessary expressiveness can be considered in our framework (indeed, less expressible DL are enough, as discussed in Section 4.4.2.5).

Nowadays ontology languages are widely used in different areas like data integration [Len02] and the Semantic Web [BLHL01], but in other areas, like software engineering, UML [Grob] and Entity-Relationship (ER) [Che76] are the most common choices. In these cases, our approach requires a *pre-process* to generate a DL ontology from the UML or ER diagram. This process can be automated nowadays [ACK$^+$07, BCG05, CCDGL02, GDD07] and the expressivity needed in DL to capture UML / ER diagrams has already been addressed in the literature [ACK$^+$07, BCG05, CCDGL02]. At this point it is important to note that when a conceptual formalization of the domain is not available then, by means of reverse engineering we may extract the ontology from the logical schema. However, the output obtained by AMDO in this case

[1] This schema captures a piece of the EU-Car Rental introduced in Appendix B.

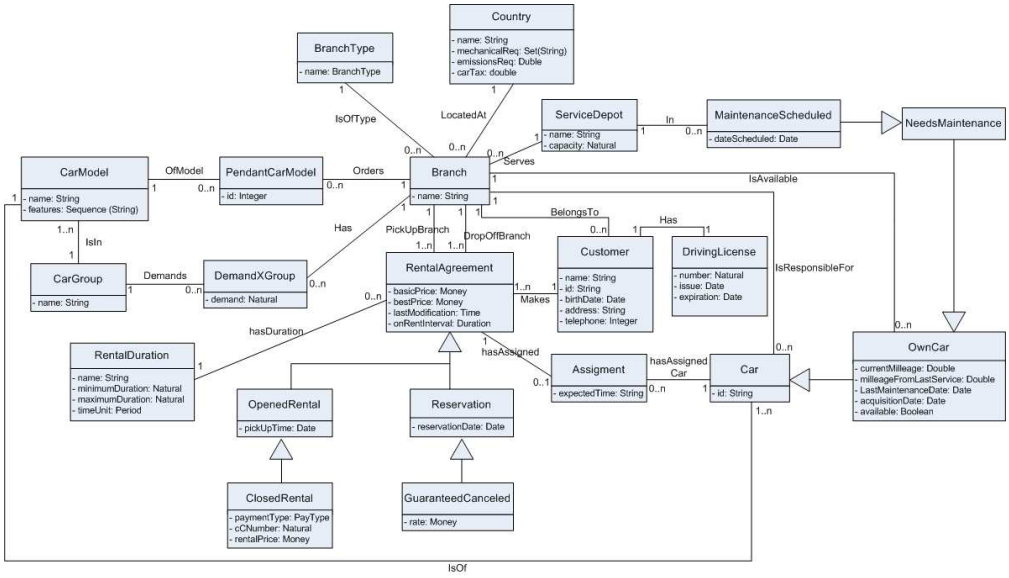


Figure 4.2: Diagrammatic representation (based on UML notation) of a piece of a car renting ontology

would be equivalent to results obtained by those approaches automating the design from logical schemas. The reason is that the ontology derived would reflect the logical design decisions made and thus, its potential lack of semantics would entail the problems described previously.

4.1 Contributions

Our proposal is a reengineering process to derive the multidimensional schema from a conceptual formalization of the domain. Working from conceptual formalizations improves the quality of the output, as earlier discussed. Although other approaches already proposed to work at the conceptual level, AMDO is the first method presented in the literature automating the whole process: i.e., identifying facts, measures and dimension hierarchies. Relevantly, AMDO also introduces a fully automated method to identify bases of interest by using and exploiting the ontological knowledge.

Previous approaches working at the conceptual level mainly rely on their requirement elicitation stages to discover the multidimensional concepts rather on an accurate analysis of the data sources (see, for example, [BvE99, BCC+01, CT98a, GRG05, GR09, HLV00, MK00, PACW06, MTL07, WS03], which are discussed in detail in Section 2.1.2). AMDO follows a completely different framework based on a thorough and fully automatic analysis of the sources and then, carrying out a guided requirement elicitation stage a posteriori, as discussed in Section 4.3. Therefore, unlike previous approaches, the automatic analysis of the sources leads the process.

AMDO considers all the multidimensional concepts in depth by analyzing their semantics and how they should be identified from the sources. As result we propose new and original design patterns. For example, a more accurate heuristic to discover facts, based on the ontology topology (see Section 4.3 for further details), is provided; we handle measures and dimensional concepts uniformly in an automatic way (see Section 4.3.1); we are able to identify *aggregate measures* (see Section 4.3.2), which have been completely overlooked in the literature and we introduce formal rules to distinguish between *descriptors* and *levels* in a dimension hierarchy as well as identify semantic relationships between dimensions (see Section 4.3.5).

A possible reason why previous approaches that work at the conceptual level have overlooked the automation of the process could be that ER (or UML) are conceptual formalizations thought to graphically represent the domain, and unlike ontologies, not thought for querying and reasoning. To our knowledge, our approach is the first one considering the data warehouse design from ontologies. Hence, we do believe that this work opens new interesting perspectives. For example, we can extend the data warehouse and OLAP concepts to other areas like the Semantic Web, where ontologies play a key role providing a common vocabulary. One consequence would be that despite the data warehouse design has been typically guided by data available within the organization, we would be able to integrate external data from the web into our data warehouse to provide additional up-to-date information about our business domain (this novel concept of data warehousing is known in the literature as *Web-Warehousing* [RALT06]).

As an additional and relevant contribution, we also propose a novel approach for discovering bases of interest by exploiting the ontological knowledge. Bases are, indeed, the multidimensional *keys*. Currently, we may find several works for computing functional dependencies (note that the traditional key concept is a specific case of functional dependencies; see, for example [AHV95]) and / or keys (e.g., [DT95, DKM08, FS99, Lim97, M. 92, SBHR06] among others), but they work either at the logical or data level, and they share some inherent constraints. Similar to the discussion earlier presented in this chapter, approaches working over the logical schema are tied to the design decisions made when devising the system (for example, denormalization of data) and these decisions have a big impact on the data semantics captured in the schema. Therefore, to avoid missing some important data dependencies, these approaches make some unrealistic assumptions such as completeness of the data structures (i.e., all the constraints of the domain of interest are captured at the logical level). For this reason, most automated approaches for identifying keys require to address this task at the instance level. However, these methods have various drawbacks: they tend to overlook composite keys (essential when dealing with bases), propose solutions that are computationally expensive, and register drops in performance when a large number of attributes or instances are processed.

Importantly, in our approach, we guide the process at the conceptual level and we introduce a set of pruning rules for improving the performance by reducing the number of key (i.e., bases) hypotheses generated, and to be verified with data. Our algorithm is relevant because, despite the importance of object identification, most DL do not provide identification mechanisms, and only very expressive DL (that are not suitable for real world applications due to their computational complexity) incorporate them [CDGL$^+$08].

4.2 Method Foundations

Our goal is to generate multidimensional schemas in an automated way and this section aims to concisely define the criteria our proposal will be based on; i.e., criteria allowing us to identify ontology concepts making multidimensional sense. Similar to the MDBE foundations introduced in Section 3.3.1, these criteria derive from the study introduced in Section 2.2.4 and discussed and formalized in Section 3.2.3. However, unlike MDBE, we do not need to consider how these criteria apply for SQL queries. Concisely, multidimensionality pays attention to two main aspects; placement of data in a multidimensional space and correct summarizability of data:

- **[Notation]**: AMDO produces conceptual schemas structured according to the multidimensional model. Nowadays, it is widely accepted that data warehouses must be exploited by OLAP tools and thus, structured according to multidimensionality. As detailed in Section 1.5, multidimensionality is based on the fact / dimension dichotomy (see **bolded** terms). **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** (i.e., **level** attributes). On the other hand, a **fact** contains **measures** of analysis, and one **fact** and several **dimensions** to analyze it give rise to a multidimensional schema.

- **[C1] 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** (and by extension, to **dimensional concepts**) by a many-to-one conceptual 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**. Importantly, note that this is a specific case of *functional dependency*. The **fact** determines the **dimension**; but unlike traditional functional dependency theory, we enforce that the **fact** is related to the **dimensional concept** by means of a *mandatory* relationship (in terms of mathematical relations, it would entail that the relation is *complete*). For the sake of understandability, from here on we force the notation and denote this kind of relationships by *complete functional dependencies*, or simply, *complete fds*.

- **[C2] The base integrity constraint**: We denote by **base** a *minimal* set of **dimensions** functionally determining a **fact**. Thus, two different instances of data cannot be placed in the same point of the multidimensional space. In other words, given a point in each of the analysis **dimensions** forming the **base**, it only determines one, and just one, instance of data. Dimensions giving rise to a **base** must be *orthogonal* (i.e., functionally independent) [ASS06]. Otherwise, we would use more **dimensions** than strictly needed to represent data, and it would generate empty meaningless zones in the space. According to our current framework, note that the **base** is the *multidimensional object identifier*.

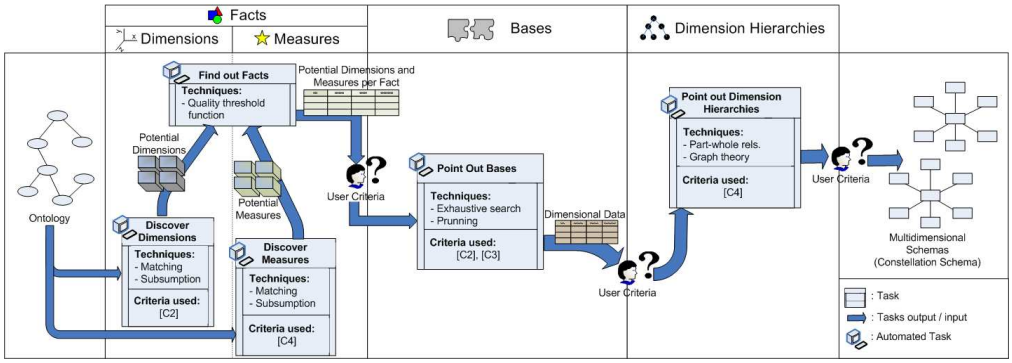


Figure 4.3: AMDO: method overview

- **[C3] The summarization integrity constraint**: Data summarization performed must be correct, and we warrant this by means of the three necessary conditions (intuitively also sufficient) [LS97]: (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 example, we cannot aggregate `stock` over the `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 example, a list of compatibilities could be asked to the user for each **measure** identified).

4.3 AMDO: Automatic Multidimensional Design from Ontologies

This section presents a detailed view of AMDO and how it applies the criteria exposed in Section 4.2. Figure 4.3 depicts a schematic overview of AMDO. There are three well-differentiated tasks:

- The first task looks for potential subjects of analysis (i.e., facts). In the literature we can find different approaches to discover facts but most of them are hardly automatable. Identifying facts automatically is a difficult task [PD02], and most methods rely on heuristics such as table cardinalities or numerical attributes that may easily identify false facts or overlook real ones. The rest of approaches demand to identify facts manually. According to the multidimensional paradigm, the analysis of data must facilitate the decision making within organizations and in this sense, the better knowledge you have, the better decisions you make. We say, thus, that an ontology concept is likely to play a fact role if it has as many measures as possible and it can be analyzed from as many different perspectives as

possible. Eventually, this fact may not be of interest for the user (this will be considered later in our approach), but objectively, it will provide many different measures to analyze from many different perspectives.

This task, therefore, is divided in two main subtasks; (1) discover potential dimensional concepts and (2) point out potential measures. Note that we do not talk about dimensions but about dimensional concepts. This step will find potential points of view to analyze the subject of analysis but, at this point, we are not able to distinguish their dimensional role (i.e., a dimension, a level or even a descriptor). This job is carried out in the third task of the algorithm, where dimension hierarchies will be shaped.

Now, for each ontology concept we can estimate its likeliness of being a fact. In general, concepts with the most potential dimensional concepts and measures are good candidates, but we may weight each input according to our preferences. In our approach we define f as a function that, given the number of dimensional concepts and measures of a concept c, it evaluates c as a promising fact. The quality function will prune those concepts below a given *threshold*. This threshold will depend on the quality function provided (AMDO is not tied to a specific function). In fact, we can use any function we would like to, like the "Connection Topology Value" (CTV) [SKD07] introduced in the literature that only gives weight to dimensional concepts found, or develop our ad hoc formula. For example, it would also be possible to rate facts according to the relevance of a concept in the application domain. Finally, potential facts not pruned are ranked according to its f value, and presented to the user. From each fact selected by the user, the second and third tasks are launched once, and eventually producing a multidimensional schema for each fact. Thus, as a whole, our approach produces *constellation schema* [KRTR98].

- The second task discovers sets of concepts likely to be used as a base (see [C2]) for each fact identified. Bases are compound of concepts identified as dimensional concepts in previous step. In short, we look for concepts being able to univocally identify objects of analysis (i.e., factual data) and produce interesting data cubes. Bases identified are pruned according to the sparsity of the multidimensional space generated. Similar to the previous step, too sparse data cubes are filtered and not presented to the user. Eventually, the user will choose, among the bases proposed, those of his /her interest.

- Finally, the third task gives rise to dimension hierarchies. Previous step filters the dimensional concepts of interest: among the whole set of dimensional concepts identified in the first step, the end-user selects those cubes of his / her interest. However, the user selects data cubes of interest (i.e., a specific data granularity level), but we need to propose interesting aggregation paths to navigate and analyze the cubes. Thus, for each dimensional concept in a base selected, we produce its own dimension hierarchy. Consequently, we aim to identify relevant aggregation paths looking for typical part-whole relationships for each interesting dimensional concept. In this step, AMDO builds graphs giving shape to each dimension hierarchy that the user may tune up to his / her necessities.

AMDO carries out an exhaustive search of potential facts among all the concepts of the domain, like supply-driven methods do. This paradigm has a main benefit with regard to those

approaches which derive the schema from requirements and later, map them onto the data sources (i.e., demand-driven approaches): in many real scenarios, the user may not be aware of all the potential analysis contained in the data sources and, therefore, overlook relevant knowledge. Demand-driven stages do not consider this fact, and assume that requirements are exhaustive. Thus, knowledge derived from the sources not depicted in the requirements is not considered and disregarded. In our approach, we claim to derive all the multidimensional knowledge contained in the ontology, filter results according to quality evidences and eventually, let the user select results of his / her interest (i.e., according to the end-user requirements) among those produced by AMDO. On the one hand, we are conciliating requirements with data available as hybrid approaches do. On the other hand, we believe that, in many scenarios, it is easier to carry out the requirement elicitation process from knowledge proposed by AMDO than carrying out it from scratch.

As counterpart, supply driven approaches tend to generate too many results and mislead the user. In this sense, our approach overcomes this problem by minimizing the amount of data shown to the user (i.e., by means of the concepts of quality function, threshold and the base concept). Specifically, after computing the likeliness of each concept as a fact, AMDO presents a ranked list of potential facts to the user (according to the quality function and threshold selected). For each concept, a value estimating its likeliness is provided. Moreover, if the user would like to, AMDO can show the list of potential measures and dimensional concepts computed for this fact. In the end, the user must select those relevant facts for his / her decision making. Similarly, among dimensional concepts identified for a fact of interest, we filter them according to the base concept. We propose to the user sets of concepts producing data cubes at different data granularity levels. Then, the user selects those cubes that better fulfill his / her necessities and accordingly, we filter the dimensional concepts. Finally, for each fact, AMDO provides (i) a list of (filtered) measures, (ii) a list of (filtered) dimensions (with their corresponding dimension hierarchies shown as a directed graph), and (iii) a list of relevant bases (i.e., potential data cubes of interest derivable from the set of dimensional concepts shown). For further details, an example over a realistic case is discussed while presenting our approach.

4.3.1 Discovering Dimensional Concepts

This section introduces the multidimensional pattern to identify dimensional concepts from DL ontologies. Note that, in this chapter, we propose two different algorithms to compute this pattern (i.e., an ad hoc reasoning algorithm, and another using generic DL reasoners). Both are properly described in Section 4.4.

According to [C1], a dimensional concept is related to a fact by a one-to-many relationship (i.e., a *complete fd*); that is, every instance of factual data is related to one, and just one, of its instances. Hence, we can express our pattern to look for dimensional concepts as follows:

$$F \sqsubseteq = 1r.D, \;\; where \;\; r \equiv (r_1 \circ \ldots \circ r_n)$$

Note that this pattern is expressed in Description Logic (DL) [BCM$^+$03], where *r* and *D* are variables, and *F* the ontology concept we are trying to identify as a potential fact. As discussed in the introduction, in the general case, we assume OWL DL (a W3C recommendation) as our

input ontology language. Accordingly, we use OWL notation and thus, we consider a *class* to be a unary predicate (i.e., *D* and *F*), and a *property* (i.e., *r*) as a binary predicate expressing a relationship between two classes . Briefly, the $\sqsubseteq$ symbol stands for *subsumption*, the basic inference in DL. Subsumption is the problem of checking if the *subsumer* (in our assertion, *F*) is considered more general than the *subsumee* ($= 1r.D$). That is, if the subsumee can always be considered a subset of the subsumer. $\equiv$ stands for a logic *equivalence* and can be defined as a specific kind of subsumption, that is: $subsumer \sqsubseteq subsumee$ and $subsumee \sqsubseteq subsumer$. $\circ$ stands for property *composition* (i.e., $\{a, c\} \in r \circ s$ iff $\exists b$ such that $\{a, b\} \in r$ and $\{b, c\} \in s$). Finally, $= 1$ stands for a specific *number restriction* where, in our case, the number of individuals belonging to class *D* related to a given individual of the class *F*, through the property *r*, must be exactly one. Thus, we are looking for classes (*D*) such that every instance of a given fact (*F*) is related, directly or by property composition (*r*), to, at least and at most, one of its instances. For each ontology class *F* we look for classes that may play the dimensional concept role by evaluating the pattern presented above; where the dimensional concept is defined by the class *D* (from here on, the *ending* concept) and a composite property *r* (from here on, the *property path* or simply, the *path*).

For example, consider the conceptual schema in Figure 4.2. `Branch` is a dimensional concept of `rental agreement`, as every instance of `rental agreement` is related, to at least and at most, one instance of `branch`. Thus, `rental agreement` would play the *F* role; `branch` the *D* role and `dropOffBranch` the *r* role. Note, however, that *r* can be a composite property and thus, `country` is a dimensional concept of `rental agreement` as well, because every instance of `rental agreement` is related, at least and at most, to one instance of `country` by means of `dropOffBranch` $\circ$ `locatedAt`.

In our approach, we not only consider classes to play a dimensional concept role (i.e., the role of *D*), but also *datatypes*. Hence, a datatype may play a measure role (as it would seem more natural to think and we will discuss later), but also the role of an analysis dimensional concept. Handling facts and dimensions uniformly is not new. In fact, it was introduced by Agrawal et al. [AGS97] and since then, it has also been considered in many other design methods. Consequently, in our example, `basicPrice` and `bestPrice` will be considered potential dimensional concepts of `rental agreement`. Importantly, note that a dimensional concept and a measure derived from the same datatype must be semantically related in the output multidimensional schema (for example, by the "equivalence" construct in OWL or by an "association" relationship in UML).

Definition 1. *A dimensional concept is defined by an* ending concept *and a* path of properties *(i.e., a composite property). From a multidimensional point of view, the path must be considered because it adds relevant semantics. Two classes related by means of* n *different to-one paths (i.e., a complete fd) must give rise to* n *different perspectives of analysis, as all these paths will potentially identify different sets of instances in the ending concept.*

For example, consider Figure 4.2. There, `rental agreement` has two to-one relationships to `branch` (i.e., `pickUpBranch` and `dropOffBranch`). Thus, {`branch`, `pickUpBranch`} and {`branch`, `dropOffBranch`} must be considered as two different points of view from where analyze a `rental agreement`, and the semantics of each dimensional concept identified is provided by the combined semantics of the path and the ending concept.

4.3.1.1 Practical Consideration

Several current design methods consider a dimensional concept to be a functional dependency (see, for example, [GR09, BvE99, HLV00, MK00, JHP04, GRG05]). Thus, they do not require the dimensional concept to be a complete functional dependency. From our point of view, we strongly recommend to enforce the theoretical pattern presented as much as possible. Relaxing them, indeed, may entail the identification of meaningless dimensions or give rise to sparser multidimensional spaces, which may mislead the user.

Nevertheless, we could consider this practical consideration (i.e., fact instances not related to any instance of a dimensional concept) and, like current approaches do, automatically create a dummy instance (for example, named `others`) related to fact instances not related to the dimensional concept. Then, our pattern to look for dimensional concepts would look like as follows:

$$F \sqsubseteq \leq 1r.D, \;\; where \;\; r \equiv (r_1 \circ \ldots \circ r_n)$$

This multidimensional pattern, however, cannot be used over arbitrary OWL DL ontologies. Indeed, the mandatory participation is needed in arbitrary ontologies to avoid discovering meaningless functional dependencies. Importantly, in an ontology, *properties are not necessarily typed*, i.e., they do not necessarily have a specified class as domain and a specified class as range. Therefore, we cannot establish, in the general case, that a property relates one class to another class. As a consequence, considering the pattern introduced in this section, every *functional untyped property* would potentially allow to infer that two arbitrary classes are functionally dependent on each other, provided that the property relates one instance to, at most, a single other instance (i.e., that it is functional).

Note, however, that this general assumption does not hold for conceptual schemas. Consider Figure 4.2. In a UML conceptual schema, every property is *strictly typed*. Therefore, OWL DL ontologies derived from conceptual schemas are also strictly typed. Thus, when working from OWL DL ontologies assuming *strict property-typing* (for example, ontologies derived from conceptual schemas) we may relax and successfully compute the alternative pattern presented in this section.

In the AMDO tool we introduced a check-box to allow the user enforce this restriction, or relax it, according to the designer own considerations.

4.3.2 Discovering Measures

We now introduce the multidimensional patterns to identify measures from DL ontologies, and in Section 4.4.2.3 we describe how to compute them.

In this step we look for measures (i.e., factual data). Typically, measures are numerical attributes allowing data aggregation. AMDO considers any summarizable *datatype* (i.e., those allowing data aggregation by its own nature) to be a measure of a given fact *F* if, according to [C3], it preserves a correct data aggregation from *F*; i.e., if they are conceptually related by a one-to-one relationship or according to our notation, by a complete fd whose *inverse* property is also a complete fd). (1) The to-one multiplicity in the measure end enforces that each fact instance is related to just one measure value, and by the mandatory participation we preserve

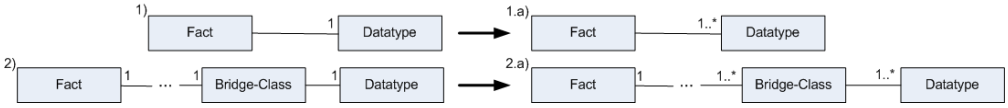


Figure 4.4: AMDO: multidimensional patterns to discover measures

completeness, (2) whereas the to-one multiplicity in the fact end preserves *disjointness* (i.e., a measure is related to one and just one fact). Similar to Def. 1, we must take into account the semantics of the paths between the fact and the datatype when producing measures. Thus:

Definition 2. *A measure is defined by a* datatype *and a* path of properties *(i.e., a composite property). From a multidimensional point of view, the path must be considered because it adds relevant semantics. A fact related to a datatype by means of* n *different to-one paths must give rise to* n *different measures, as all these paths will potentially relate each fact instance with different datatype values.*

It is important to note that our definition of measure is wider than the definition used by previous approaches. Previous approaches identify them among class / entity (in case of working over ER or UML diagrams) numerical attributes or among numerical attributes of relations (in case of working over relational schemas). In our framework (i.e., from an ontology) the measure definition, according to previous approaches, would be equivalent to only consider datatypes directly related to classes. Oppositely, our definition is not restricted to direct datatypes but to any (i.e., any reachable by means of property composition) preserving the one-to-one relationship required. This kind of measures (from here on, *aggregate measures*) have been overlooked in the literature, but identifying them is important to discover meaningful and additional factual data.

Figure 4.4 shows the two type of patterns we look for. The first pattern (see 1) depicts a measure directly related to the fact. This pattern is the equivalent one in DL to those used by previous approaches in the relational model or UML / ER diagrams. Roughly speaking, a datatype does not have an *object identifier* (i.e., an *oid*) and we allow any multiplicity in the fact end without violating *disjointness*. Consider now our example depicted in Figure 4.2. There, the `bestPrice` and `basicPrice` datatypes related to `rental agreement` would be identified by this pattern as potential measures for `rental agreement`. As a particular scenario (see 1.a), we may accept mandatory multivalued datatypes (i.e., each class instance must be related to at least one value and at most, to many of them). In this case, the *many* values related to each fact instance should be aggregated by means of a *compatible* aggregation function (see [C3]) prior to be inserted in the data warehouse once built. AMDO provides the designers with this information to be taken into account during the ETL process (for example, as an appendix of the multidimensional schema produced).

Next pattern (see 2) is used to discover aggregate measures. It depicts a class (from here on, a *bridge-class*) that is both directly related to a datatype, and, by means of a one-to-one path, to a fact. This datatype is also a potential measure for the fact since *disjointness* and *completeness* are guaranteed. In this case, similar to the scenario discussed in the previous pattern, in the path between the fact and the bridge-class, it is enough to ask for a mandatory relationship (see

2.a). Again, all the (*many*) measure values related to each fact instance must be aggregated by a *compatible* aggregation function, prior to be inserted in the data warehouse. Therefore, [C1] is also guaranteed in this pattern.

In our example (see Figure 4.2) and according to this pattern, the `bestPrice` and `basicPrice` of a `rental agreement` per `branch` would be identified as aggregate measures of `branch`. In this case, `branch` play the fact role, `rental agreement` is the bridge-class and `bestPrice` (or `basicPrice`) the datatype related to the bridge-class. Depending on the aggregation function used, we can derive different aggregate measures. For example, if we use the `MIN` operator we would get the best `bestPrice` (or `basicPrice`) offered in a given `rental agreement` per `branch`; using the `AVG` operator we would get the average `bestPrice` (or `basicPrice`) offered in a `rental agreement` per `branch` and so on. The patterns discussed in this section may be formally captured as follows:

- (**Pattern 1**) For each class *C*, we look for summarizable datatypes *directly* related to it. In DL it is equivalent to look for mandatory to-one properties such that their *domain* is *C* and their *range* are datatypes. We can take advantage of basic reasoning to compute this pattern (see Section 4.4.2.5 for further details):

  $$C \sqsubseteq = 1r.dt,$$

  Where *r* is a single property and *dt* any datatype allowing aggregation of data (for example, the OWL *int*).

  - (**Pattern 1a**) For considering mandatory multivalued datatypes we must consider any mandatory property such that its *domain* is *C* and its *range* is a datatype:

    $C \sqsubseteq \exists r.dt$, where analogously, *r* is a single property and *dt* a summarizable datatype.

- (**Pattern 2**) A datatype *dt*, directly related to a class *B* is a potential measure for a class *F* iff *B* is a bridge-class for *F*. Note that, similar to Def. 1, each path between *F* and *dt* will produce a different measure:

  $F \sqsubseteq = 1r.B \wedge B \sqsubseteq = 1r^{-}.F$, where *r* is a composite property and *B* fulfills Pattern 1.

  If so, it means that, by transitivity, we have been able to identify *B* as a dimensional concept of *F* (i.e., *B* is a complete fd of *F*) and viceversa.

  - (**Pattern 2a**) For considering mandatory multivalued datatypes we must consider any mandatory path such that its *domain* is *F* and its *range* is a bridge-class:

    $F \sqsubseteq \exists r.B \wedge B \sqsubseteq = 1r^{-}.F$, where *r* is a composite property and *B* fulfills Pattern 1.a).

    In this case, it means that *F* is a dimensional concept of *B*, and *B* is related to the bridge-class by means of a mandatory path. Thus, by aggregating data we obtain the needed one-to-one relationship.

4.3.2.1 Practical Consideration

Multidimensional patterns to discovering measures can also benefit from the practical considerations discussed in in Section 4.3.1.1. In this case, if we want to emulate previous approaches, the multidimensional patterns could be relaxed by not requiring the bridge-class (or the datatype) to be a complete functional dependency of the fact but just a functional dependency. In other words, relax the mandatory participation in the bridge-class end / datatype of the conceptual relationship (i.e., not forcing each fact to have a numerical value for that measure). In this case, if the user selects this fact as of his / her interest, we need to introduce a specialization of that measure in the data warehouse conceptual schema to preserve completeness. Thus, our new patterns would look like, respectively, as follows:

- (**Pattern 1**): This pattern now, looks for functional dependencies:

  $$C \sqsubseteq \leq 1r.dt, \text{ where } r \text{ is a single property.}$$

  - (**Pattern 1.a**): Note that, now, Pattern 1.a does not have to be considered anymore, since any multiplicity would be allowed in the datatype end of the relationship.

- (**Pattern 2**): This pattern consider as bridge-class those concepts being a functional dependency of F:

  $F \sqsubseteq \leq 1r.B \wedge B \sqsubseteq = 1r^{-}.F$, where r is a composite property and B fulfills Pattern 1.

  - (**Pattern 2.a**): Similar to Pattern 2, it considers any functional dependency of F to play the role of a bridge-class. Furthermore, since Pattern 1.a does not hold anymore, we just ask the bridge-class to be related to a datatype:

    $$B \sqsubseteq = 1r^{-}.F, \text{ where } r \text{ is a composite property.}$$

Importantly, like in Section 4.3.1.1, these alternative patterns can only be consider when working over OWL DL ontologies considering strict property-typing.

4.3.3 Discovering Facts

Once potential dimensional concepts and measures of analysis have been computed for each class, AMDO computes the likeliness of each ontology class as a fact. Results obtained are ordered according to a function f that evaluates how good candidate each class is. Those values not reaching a pre-defined quality threshold are directly pruned and not presented to the user.

Table below summarizes the information presented to the user at the end of this step. There, each class is followed by its number of potential dimensional concepts and measures identified during the process. In our case, results are ordered according to the `FactEstimation` function, which is defined as follows:

$$\texttt{FactEstimation(M, DC)} := \begin{cases} M * 2 + DC, & if \texttt{ M} > 0; \\ 0, & otherwise. \end{cases}$$

Concept	#Dimensional Concepts	#Potential Measures	FactEstimation
LateReturn	78	5	88
DamageCost	81	3	87
Prepared	81	3	87
AssignedCar	80	3	86
PaidWithPointsRental	74	4	82
ClosedRental	74	4	82
EarlyReturn	74	4	82

Table 4.1: AMDO: ranked facts proposed for the EU-Car Rental case study

where M is the number of potential measures, and DC the number of potential dimensional concepts of a given ontology class. As discussed in Section 4.3, AMDO is completely flexible and tuneable regarding the quality function *f*, and designers are able to provide any function which meets his / her requirements better.

Consider now the EU-Car Rental case study shown in Appendix B (and from where Figure 4.2 is extracted). Next table summarizes the classes that AMDO would propose as facts if our threshold regarding the `FactEstimation` function was 80 (note that, according to our quality function semantics, in average, we are asking for facts having two measures and, at least, 5 dimensions with an average of 15 dimensional concepts; i.e., levels and descriptors):

Taking a look at results provided, in general, the classes in the `rental agreement` taxonomy (i.e., `early` or `late rental`, `closed rental` and `club member rental`) are the best candidates to play a fact role, and *subclasses* are better rated than *superclasses*. This result is sound, as we should expect the `rental agreement` to be the keystone concept of a rental service. Moreover, subclasses inherit all the multidimensional knowledge inferred for their superclasses and in the worst case, they will be rated as high as their superclasses. Classes in the list which do not belong to the `rental agreement` taxonomy are just three: `damage cost` (2nd place), `prepared` (3rd place) and `assigned car` (4th place). However, they represent events, which traditionally have been good candidates as facts (see, for example, [BvE99, MK00]). `Damage cost` captures information about cars damaged during the rental agreement; `prepared` captures information of when the car assigned to a rental agreement is prepared and finally, `assigned car` (which is `prepared` superclass) tells us about which car has been assigned to which rental agreement. Objectively, three interesting events to analyze.

At this point, among facts proposed, the user should select those facts of interest to him / her (normally, more than one). Importantly, AMDO can provide additional information to help the user in this process. For example, it can show the measures and dimensional concepts computed for a given fact, or arrange the multidimensional concepts to propose dimension aggregation hierarchies (by pre-computing the patterns introduced in Section 4.3.5).

4.3.4 Discovering Bases

We now discuss the multidimensional pattern to discovering bases, whereas Section 4.5 introduces an algorithm to compute bases by exploiting the ontological knowledge available. Ac-

cording to the base definition in Section 4.2, we denote by base a *minimal* set of dimensional concepts *determining* a fact. Furthermore, dimensions giving rise to a base must be *orthogonal* (i.e., functionally independent). Thus, the base concept provides necessary identification mechanisms for multidimensional objects (i.e., instances). Indeed, its definition is equivalent to the relational *minimal key* concept. Unfortunately, this pattern cannot be expressed by means of DL notation. Indeed, despite the importance of object identification, most DL do not provide identification mechanisms, and only very expressive DL (that are not suitable for real world applications due to their computational complexity) incorporate them [CDGL$^+$08]. For example, the only way to specify identification in OWL DL is by means of one-to-one properties, which are clearly not enough. Furthermore, the fact that most description logics do not consider n-ary relationships makes impossible to assert composite keys for ontologies. In short, there is no way to express that $A \cup B \rightarrow C$ holds (where A, B and C are concepts) because in most ontology languages, roles are binary predicates relating one class to another class.

In our approach, we do not present just another algorithm for computing keys. Importantly, AMDO guides the process at the conceptual level and introduces a set of pruning rules for improving the performance by reducing the number of key (i.e., bases) hypotheses generated, and to be verified with data.

Once the bases algorithm has finished, verified bases obtained are presented to the user according to their *sparsity level*. This quality rule works on the same principle as [C2], which asks dimensions forming a base to be orthogonal. Sparse multidimensional spaces can mislead the user and thus, among the bases obtained, we rank better those less sparse. Similar to the quality rule for ranking facts, we introduce the concept of *sparsity threshold* and accordingly, verified bases below this threshold are discarded and not even presented to the user.

4.3.5 Shaping Dimension Hierarchies

Previous steps have identified, for each fact, the measures and dimensional concepts of interest. However, we still need to shape the dimension hierarchies in order to allow summarizability of data; one of the multidimensionality principles. In this step we present the multidimensional patterns used to shape dimension hierarchies, whereas Section 4.4.2.4 describes how to compute them. Dimension hierarchies must guarantee a correct summarizability of data (see [C3]). In this step we look for to-one relationships (also known as *roll-up* or *whole-part* relationships, or according to our notation, complete fds) giving rise to hierarchies allowing a correct data aggregation: the to-one multiplicity preserves *disjointness* of aggregated data, and by the mandatory participation we also preserve its *completeness*.

From the set of dimensional concepts filtered by previous section, a directed graph following all the to-one relationship paths is depicted. At this point, two important remarks must be done. On the one hand, note that some graphs will overlap. Consider two concepts A, B such that B is a dimensional concept of A. Clearly, the graph created from B will be subsumed by the graph created from A. In these cases were a graph is subsumed by another one (i.e., it is completely overlapped), the *subsumee graph* is disregarded and not considered during the process. Intuitively, we only work with *maximal* graphs, as the rest can be derived from them straightforward. On the other hand, note that, at this moment, we cannot differentiate the role played by each graph node (either as a level or as a descriptor) within the dimension hierarchy. Two

specific patterns are introduced to distinguish levels from descriptors:

- (**Levels**): If a class (C_3) is placed in more than one graph, we consider it to be a level (since it seems interesting to show data at this granularity level). Following with DL notation we could formalize this pattern as:

$$\exists C_1, C_2, \exists r_1, r_2 \mid (\exists r_1 \sqsubseteq C_1) \wedge (\exists r_1^- \sqsubseteq C_3) \wedge (\exists r_2 \sqsubseteq C_2) \wedge (\exists r_2^- \sqsubseteq C_3) \wedge C_1 \neq C_2 \wedge r_1 \neq r_2$$

  Where C_1 and C_2 are classes and r_1 and r_2 are disjoint paths formed by graph edges. Intuitively, we are looking for classes in two different graphs. These classes will give rise to two different levels in each graph. Nevertheless, as they refer to the same concept, it is mandatory to semantically relate both levels by means of a conceptual relationship in the output schema.

- (**Descriptors**): If two classes C_1 and C_2 are related by means of a one-to-one relationship (i.e., two inverse complete fds) in a graph:

$$\exists r_1 \mid (C_1 \sqsubseteq = 1 r_1 . C_2) \sqcap (C_2 \sqsubseteq = 1 r_1^- . C_1)$$

  This relationship can represent a (i) semantic relationship between two different dimensions if the *ending* class (note that the graph is directed) was identified as a level by the previous pattern, or (ii) as an attribute level (i.e., a descriptor) otherwise. In the first case, it means that the ending class provides a relevant data granularity (i.e., it was identified as a level) and thus, we also consider its counterpart to be of interest. Since a one-to-one relationship does not make much sense within the same dimension hierarchy, they are considered to represent the same granularity level in two different dimensions that, consequently, must be related semantically. In the second case, we consider it to be a descriptor of the *initial* concept, since they have not been identified as interesting analysis levels. Similar to previous pattern, semantic relationships identified must be explicitly asserted in the output schema.

Directed graphs built in this step are presented to the user as dimension hierarchies, altogether with those semantic relationships between dimensions identified. Note, however, that these two patterns are not exhaustive and, in some cases, we have not been able to identify each graph node as a level or a descriptor. Nevertheless, this is sound, as it is up to a design decision to spot each class as an attribute of an existing level or as a new level; giving rise to implicit or explicit dimension hierarchies in the resulting schema. Consequently, this differentiation should be made by the user, if she / he is interested in aggregating data at this level, when stating his/her requirements. Finally, the user may also be interested in reshaping the analysis dimensions derived from datatypes. AMDO does not propose any hierarchy for these dimensions automatically, but we allow the user to use predefined functions (for example, get_day, get_month or get_year in case of dates) or aggregate data to generate value ranges in case of numerical datatypes (for example, from 0 to 20, 21 to 50, 51 to 99 in case of ages), to create ad hoc dimension hierarchies.

For example, suppose that the user chooses `rental agreement` as a fact of his / her interest (see Section 4.3.3). For this subject of analysis, AMDO would generate the dimension

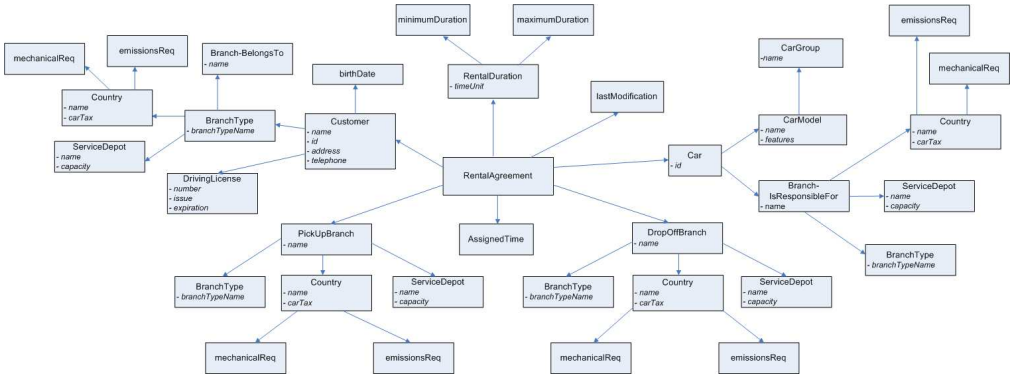


Figure 4.5: AMDO: multidimensional schema proposed for the `rental agreement` fact

hierarchies shown in Figure 4.5. There, each arrow starting from `rental agreement` depicts a dimension hierarchy. Concepts identified as levels by AMDO are depicted as a class, whereas concepts identified as descriptors are depicted in *italics* in the level they belong to. In total we generate nine maximal directed graphs (from `customer`, `lastModification`, `assigment`, `dropOffBranch`, `pickUpBranch`, `rentalDuration`, `car`, `bestPrice` and `basicPrice`). Next, the user must tune up dimension hierarchies obtained up to his necessities. For example, by considering some descriptors as interesting aggregation levels (it is the case of `minimumDuration` and `maximumDuration` that were identified as descriptors but we have considered to be interesting levels of aggregation in Figure 4.5) or by dropping dimensions of no interest (in our example, we have disregarded the (single-node) graphs produced for `bestPrice` and `basicPrice`). It is important to remark that levels derived from the same class but placed in different graphs would be related by semantic relationships. For example, `country` or `branch type`, which appear in different hierarchies.

4.4 Computing Functional Dependencies

After presenting the patterns used by AMDO to identify the multidimensional concepts from a domain ontology, we know discuss how to compute them. Importantly, note that patterns presented to discover dimensional concepts, measures and dimension hierarchies are based on discovering complete fds. For example, dimensional concepts (see Section 4.3.1) and dimension hierarchies (see Section 4.3.5) patterns directly ask for complete fds between classes. About the patterns for discovering measures, Pattern 1 (see Section 4.3.2) looks for complete fds between classes and datatype. Pattern 2 looks for a bridge-concept B fulfilling Pattern 1 such that, the fact is a dimensional concept of B and viceversa (i.e., B is a complete fd of F by means of a path p, whose inverse, namely p^-, also depicts a complete fd between F and B). Consequently, in this section we propose two different algorithms to compute complete fds. First, according to our general assumption, we consider the input ontology to be expressed in OWL DL. As discussed

in Section 4.4.2, this algorithm only benefits partially from generic reasoning algorithms. For this reason, and to take advantage of the well-known reasoning services provided by DL languages, we propose a second algorithm fully computable by a generic DL reasoner. To do so, we restrict the expressivity of the input DL, as discussed in Section 4.4.3.

As an exception, note that measure patterns (namely, Pattern 1 and Pattern 2) can be relaxed and just ask for a mandatory (i.e., complete relation) in the bridge-class / datatype end. However, we will discuss later how to compute them.

4.4.1 Computing Functional Dependencies Over DL Ontologies

In this section, we discuss how the functional dependency concept maps onto ontologies. Specifically, we discuss how to discover functional dependencies by relying on the assertions in a DL ontology. First, we recall some basic definitions regarding functional dependencies in the standard relational model (see, e.g., [AHV95]). Consider a relation schema R, i.e., a relation symbol with an associated set of attributes, each denoting one component of R. A *functional dependency* (fd) over R has the form $R : X \rightarrow Y$, where X and Y are sets of attributes of R. We say that a relation r for R *satisfies* such a dependency if for each pair t_1, t_2 of tuples in r such that $\pi_X(t_1) = \pi_X(t_2)$, we have $\pi_Y(t_1) = \pi_Y(t_2)$ (where, as usual, $\pi_X(t)$ denotes the projection of tuple t on the attributes in X).

It is well known (for example, see [AHV95]) that the following set of inference rules is sound and complete for implication of fds over a relation schema R (below, X, Y, and Z are sets of attributes of R, and juxtaposition of two sets stands for their union):

- If $Y \subseteq X$, then $R : X \rightarrow Y$ (*reflexivity*).
- If $R : X \rightarrow Y$, then $R : XZ \rightarrow YZ$ (*augmentation*).
- If $R : X \rightarrow Y$ and $R : Y \rightarrow Z$, then $R : X \rightarrow Z$ (*transitivity*).

In other words, all fds derived from a set $\mathcal{F}$ of fds over R, i.e., the $\mathcal{F}$*-closure*, can be computed by starting from $\mathcal{F}$ and exhaustively applying the above inference rules.

We would like now to carry over the conceptual level the standard notion of fd defined at the logical level. To this aim, we introduce the formal notion of fd over an ontology. We observe that previous work has already considered fds in the context of ontologies, see e.g., [CDGL01, TW05, TW08]. In these works, mimicking the notion in the relational model, a fd ensures that, if two objects that are instances of some concept share the same values for a set of attributes (or of attribute chains), then they share also the value of an additional attribute (or attribute chain), namely the attribute (chain) that functionally depends on the former attributes (chains). Instead, for our purposes, a fd should capture the intuition that the instances of one concept *functionally depend* on the instances of another concept. In other words, given two concepts C_1 and C_2, we are interested in establishing whether each instance of C_1 allows one to determine a unique instance of C_2. We will denote this by $C_1 \rightarrow C_2$. Several observations are in order:

(i) The dependency between the two concepts C_1 and C_2 needs to be established explicitly, and this can be done by means of some role that relates C_1 to C_2[2].

[2]Note that in the relational model, attributes that functionally depend on other attributes are implicitly related through the relation schema to which the attributes belong.

(ii) Since each instance of C_1 should determine a unique instance of C_2, and such a dependency is established through a role, we need to require such a role to be functional.

(iii) If we want to ensure a property analogous to transitivity (i.e., if $C_1 \rightarrow C_2$ and $C_2 \rightarrow C_3$, then also $C_1 \rightarrow C_3$), we need to allow the dependency to be established not only by atomic roles, but also by composite roles (i.e., role chains).

(iv) In an ontology, roles are not necessarily typed, i.e., they do not necessarily have a specified concept as domain and a specified concept as range. Therefore, one cannot establish in general that a role relates one concept to another concept. As a consequence, every untyped role would potentially allow one to establish that two arbitrary concepts are functionally dependent on each other, provided that the role relates one object to a single other object, i.e., that it is functional. This is clearly unsatisfactory, and therefore we need to enforce some stricter condition for a functional dependency $C_1 \rightarrow C_2$ to hold. Specifically, we will require not only that the role is functional, but also that the instances of C_1 mandatorily participate to the role, and that the role necessarily relates them to an instance of C_2.

Importantly, note that the fd notion over DL ontologies is equivalent to that of complete functional dependency introduced in Section 4.2.

4.4.2 Using Specific Reasoning Algorithms

In this section we consider that AMDO's input ontology is expressed in OWL DL. Unfortunately, we may not take advantage of generic DL reasoning algorithms for directly computing patterns introduced at once, as most common reasoning services are not decidable when considering composite properties [BCM$^+$03]. Indeed, composite properties are not even expressible in OWL DL. For this reason, in this section we discuss each pattern separately. We start discussing how to compute the pattern to discover dimensional concepts (see Section 4.3.2):

- First, for a given class $\mathcal{F}$, we look for its *direct* dimensional concepts.
- Next, we compute the transitive closure of dimensional concepts according to the following *transitive rule*:

  Definition 3. *If* $\{\mathrm{A}, \mathrm{r}\}$ *(being* A *a class and* r *a property) is a dimensional concept of* B*, and* $\{\mathrm{C}, \mathrm{r1}\}$ *is a dimensional concept of* A*, then* $\{\mathrm{C}, \mathrm{r} \circ \mathrm{r1}\}$ *is a dimensional concept of* B *as well.*

The first step can be completely computed using generic algorithms provided by DL reasoners and the second step requires an ad hoc algorithm that will also partially benefit from these algorithms.

4.4.2.1 Computing Direct Dimensional Concepts

Computing direct dimensional concepts (i.e, complete fds) is equivalent to consider *r* as a single property instead of a composite property in the pattern introduced in previous section:

$$F \sqsubseteq = 1r.D,$$

Where *r* is a single property. This pattern can be computed by basic reasoning (see Section 4.4.2.5 for further details about basic reasoning in DL) and for each class *F* we keep track of pairs $\{D, \{r_1, ..., r_n\}\}$ (where every r_i is a property between *F* and *D*), which define its potential dimensional concepts. According to Def. 1, every r_i identified between F and D will give rise to a different multidimensional concept. Using generic reasoning means that any assertion stated in the ontology (by using OWL DL constructs) is automatically considered. For example, subsumption of classes, subsumption of properties, cardinality restrictions, functional (or inverse functional) properties, etc.

Finally, note that if the practical consideration introduced in Section 4.3.1.1 must be considered then, the pattern to look for in this step would be:

$$F \sqsubseteq \leq 1r.D,$$

Where *r* is a single property, which can be also computed by basic reasoning.

4.4.2.2 Computing the Transitivity Closure of Dimensional Concepts

In this section we present an ad hoc algorithm to compute the transitive closure of dimensional concepts. Despite this algorithm cannot be fully computed by using generic reasoning services, we can take advantage of subsumption to propagate this knowledge through class taxonomies, as we will show later.
Our algorithm aims to compute the transitive closure of the asserted complete fds. With this aim, we build a matrix M of *N*×*N* elements (where *N* is the number of classes in the ontology) such that each row depicts a class and its potential dimensional concepts:

$$\forall\{D, \{r, ..., r_n\}\} \in M < F >: \begin{cases} F \sqsubseteq = 1r.D, \\ \cdots \\ F \sqsubseteq = 1r_n.D \end{cases}$$

Where M is the *N*×*N* matrix, *F* and *D* are classes, *r*, ..., r_n are composite properties. $M < F >$ is an operator over M that retrieves a list of classes related to *F* by, at least, a complete fd (i.e., its list of dimensional concepts). Each class in this list is represented as $\{D, \{r, ..., r_n\}\}$ where *D* is the class (or datatype) itself and each r_i is a to-one path (i.e., a complete fds) depicted as a composite property. Therefore, we may derive as many dimensional concepts from *D* as different paths it has. Roughly speaking, each cell $C_{(F,D)}$ of M contains a list of composite properties ($\{r, ..., r_n\}$) such that each instance of *F* is related at least and at most to one instance of *D*.

In this section we aim to build the final state of this matrix, and we achieve it by means of the next algorithm:

Since M is a sparse matrix, the function *create matrix* implements it as a *vector* of *lists* (see Figure 4.6, step 1). Thus, every position in the vector represents a class and its list of potential dimensional concepts (see the $to{-}one\_rels$ *typedef* declaration). Lists are created and initialized to the empty list in step 2. Step 3 finds and breaks trivial deadlocks. The need of this step will be justified later in this section.

Step 4 corresponds to the pattern presented in Section 4.4.2.1. Each potential dimensional concept identified is added to the proper list in the vector. Figure 4.7.1 (which represents ma-

typedef list <properties> path

typedef tuple < concept, **list**<path> > paths_to_concept

typedef tuple < concept, **list**<paths_to_concept> > to-one_rels

function*create_matrix* **returns** Matrix

1. **vector**< list<to-one_rels> > M;
2. initialize(M);
3. compute_trivial_deadlocks(M, ontology);
4. first_iteration(M, ontology);
5. propagate_path(M, converge);
6. **return** M;

Figure 4.6: AMDO: an algorithm to compute matrix M

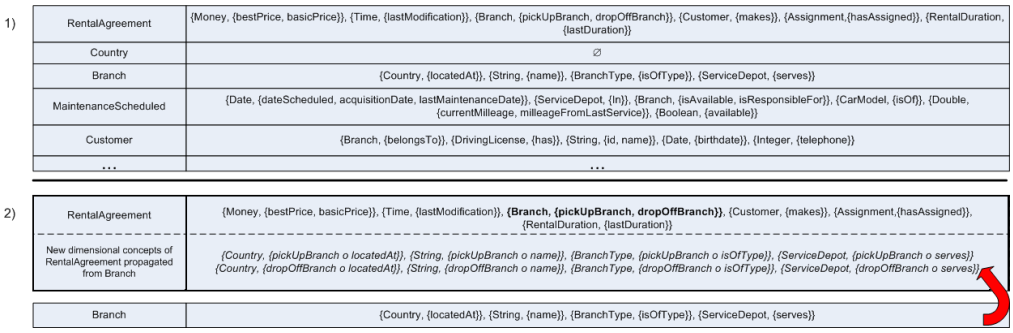


Figure 4.7: AMDO: exemplification of to-one paths propagation by transitivity

trix M) shows results obtained after step 4 for some of the concepts in Figure 4.2. For example, the `maintenance scheduled` class has a to-one relationship to `date` (through the `dateScheduled`, `acquisitionDate` and `lastMaintenanceDate` properties), `service depot` (through the `In` property), `branch` (through the `isAvailable` and `isResponsibleFor` properties), `car model` (through the `isOf` property), `boolean` (through the `available` property) and `double` (through the `currentMilleage` and `milleageFromLastService` properties). Note that, according to Def. 1, those classes (or datatypes) related to `maintenance schedule` by several different paths (e.g., `date`) will generate as many dimensional concepts as different paths between both classes. For example, `date` will produce three different dimensional concepts: {`date`, `dateScheduled`}, {`date`, `acquisitionDate`} and {`date`, `lastMaintenanceDate`}.

Step 5 propagates the dimensional concepts identified in the previous step according to the

transitive rule (see Def. 3). We use the *propagate_path* function (see Figure 4.8) for this purpose.

void *propagate_path* (Matrix ↕M)

7. **list** <paths_to_concept> ending_concepts;
8. **foreach** *C* **in** M **do**
   (a) **if** $M_c < C > = \varnothing$ **then**
      i. $C.closed$:= true;
9. **do** {
10. bool conceptsClosed := false;
11. **foreach** *C* **not closed in** M **do**
   (a) reachable_concepts := $M_c < C >$;
   (b) **foreach** *D* **in** reachable_concepts **such that** !$(M < C, D >).treated$ **do**
      i. **if** $D.closed$ **then**
         A. **foreach** r **in** $M_p < C, D >$ **do**
         B. listEls := listEls ∪ $(r \circ M < D >)$;
         C. $M < C >$:= $M < C >$ ∪ listEls;
         D. $M < C, D >.treated$:= true;
   (c) **if** all_closed(reachable_concepts) **then**
      i. **list** <concept> parents := compute_direct_superconcepts(C, M);
      ii. **if** all_closed(parents) **then**
         A. **foreach** P **in** parents **do**
         B. $M < C >$:= $M < C > \cup M < P >$;
         C. $C.closed$:= true;
         D. conceptsClosed := true;
12. if(!conceptsClosed)
   (a) break_deadlocks(M);
13. } **while** concepts_not_closed(M) > 0

Figure 4.8: AMDO: an algorithm to compute the transitive closure of dimensional concepts

This function implements a smart algorithm to compute the transitive closure of the asserted complete fds (i.e., the final state of M). Essentially, the list of dimensional concepts of each class is propagated only once during this process, when we know that it cannot vary. To do so, dimensional concepts are propagated from *the end* of the complete fd paths (from here on, *leaf classes*) to *the beginning*, according to the definition of *closed class*:

Definition 4. *We say a* class C is closed *or that a given class C closes in the* i^{th} *iteration of our algorithm* Closed(C,i)*, if all its dimensional concepts have been computed in* i *or a previous iteration. In other words, if a class C closes in a certain iteration* i*, no other dimensional concept will be identified for C in any iteration* j *such that* i < j*. In our notation, we define* Closed(C,i) *as a recursive function:*

$$\textit{Closed(C,i)} := \forall\{D, \{r, ..., r_n\}\} \in M < C >: Closed(D, j) \wedge j < i,$$

Our algorithm only *propagates* closed classes (see the *closed* method in the algorithm). If a given class *C* closes in the i^{th} iteration of the algorithm, we propagate its dimensional concepts in the $i+1^{th}$ iteration. Once propagated, it is never considered again thanks to the *treated* method. Note that this method does not hold at a class level but at dimensional concept level (i.e., regarding *C*: $M < C, D >.treated$). Thus, our algorithm aims to identify closed classes and propagate them just once. Propagating knowledge is done in two different ways:

- Let *C* and *D* be two classes such that *D* is in the dimensional concepts list of *C* (see step 11b). Then, the dimensional concepts list of *D* is propagated to *C* by transitivity according to Def. 3 (see step 11(b)iA):

$$\forall D \in M_c < C >, \forall D_i \in M_c < D >: \{D_i, \{M_p < C, D > \circ M_p < D, D_i >\}\} \in M_p < C, D_i >,$$

  Where $M_c < C >$ and $M_p < C, D >$ are two operators over matrix M. The first one retrieves the list of classes in the dimensional concept list of a given class *C* (i.e., it is equivalent to the operator $M < C >$ but overlooking the path lists), and the second one retrieves the list of paths between two classes *C* and *D* such that *D* is in the dimensional concept list of *C* (i.e., it retrieves the path information between *C* and *D* in $M < C >$). D_i are the set of classes in the dimensional concepts list of a concept *D* (i.e., $\forall D_i,\ D_i \in M_c < D >$) and $\{M < C, D > \circ M < D, D_i >\}$ represents the concatenation of each path in $M < C, D >$ with each path in $M < D, D_i >$ (for the sake of readability, this formalization is depicted with an slight abuse of notation in step 11(b)iB of the algorithm). Roughly speaking, we are concatenating each to-one path from *C* to *D*, with each to-one path from *D* to D_i (see step 11(b)i).

- Let *P* be a class such that *P* is a parent (i.e., a direct superclass) of *C*. Then, all the dimensional concepts of *P* must be inherited by *C* (i.e., $M < C >$:= $M < C > \cup M < P >$, see step 11(c)iiB). In our algorithm, this kind of propagation is done when all the dimensional concepts of *C* have closed (see step 11c). Then, we can take advantage of DL basic reasoning (see Section 4.4.2.5 for further details) to compute the list of superclasses to propagate. Moreover, we do not propagate them until they have closed to avoid propagating more than once (see step 11(c)ii).

Finally, closed classes are detected as follows:

- First iteration: Leaf classes and datatypes (as discussed in Section 4.4.2.1, our algorithm considers the datatypes as potential dimensional concepts and in this sense, they can be considered leaf classes) close in this iteration (see step 8).

- Second iteration: Classes that *were closed* in the previous iteration are now propagated. In this case, propagating them is immediate, as their lists of dimensional concepts are empty. Now, according to our definition of closed class, any class whose dimensional concepts have closed (and therefore, already propagated), closes in this step.

- N^{th} Iteration: A given class *C* will close in this iteration if the last class to close in its list of dimensional concepts has already closed in the $(n\text{-}1)^{th}$ iteration (see step 11c). Therefore, all the dimensional concepts of *C* have already been computed and we can now propagate *C* in the next iteration (see step 11(c)iiC).

Following our example, Figure 4.7.1 shows direct dimensional concepts identified for each ontology class and Figure 4.7.2 depicts how we propagate them by transitivity. For example, `rental agreement` is related by two to-one relationships to `branch` (**bolded** in the figure), and `branch` is related by to-one relationships to `country`, `string`, `branch type` and `service depot`. Hence, the latter are also considered dimensional concepts of `rental agreement` according to the transitivity rule (see the arrow in Figure 4.7.2). Moreover, note that the path list of these newly identified dimensional concepts have been properly updated when adding them to the list of `rental agreement`. For example, we can get from `rental agreement` to `branch` by two different paths (i.e., `pickUpBranch` and `dropOffBranch`) and from `branch` to `country` by the `locatedAt` property. Therefore, from `rental agreement` we can get to `country` through the composition of `pickUpBranch` and `locatedAt`, and `dropOffBranch` and `locatedAt`. Analogously for the rest of dimensional concepts.

At a given iteration, when detecting closed classes, it is important to detect potential *deadlocks* due to complete fd cycles between classes. When a cycle is detected (in our algorithm, when no class closes in the current iteration; see step 12) it is broken propagating just once among them (and therefore, sharing all their potential dimensional concepts). Moreover, this situation will be notified to the user to let him / her know that each recurrent propagation within the cycle may add new interesting semantics (i.e., new analysis dimensional concepts) that could be considered. The most common and also easiest case to detect are one-to-one and reflexive relationships. To facilitate the process, AMDO treats these two basic cases immediately after being identified (see step 3 of Figure 4.6). Detecting trivial deadlocks can be computed by generic reasoning (see Section 4.4.2.5 for further details) and we may use any of the current algorithms presented in the graph theory to detect general cycles. For example, a *depth-first-search* (DFS) remembering previous visited nodes would fit properly. For example, `customer` and `driving license` through the one-to-one `has` property is computed in the first iteration. At this point, we are able to compute its list of dimensional concepts breaking up the deadlock (i.e., do not apply the transitive rule over it) and notifying to the user that, in case s/he would be interested, it is possible to derive new dimensional concepts just following the cycle semantics.

Complexity of the Algorithm The computational cost of this algorithm has $\Theta(N \times c^l)$ as an upper bound; where N is the number of classes in the ontology; c the maximum to-one *connectivity* (i.e., direct to-one relationships from a class) and l the maximum chain of to-one properties. However, this upper bound is theoretical and hardly achievable in practice since real ontologies neither have all classes with maximum to-one connectivity nor all to-one paths are of maximum length. Moreover, along the process, classes computed in previous iterations are not considered in the forthcoming ones.

In practice, the computational complexity raised by AMDO is polynomial for most ontologies. For example, consider the EU-Car Rental ontology (see Figure 4.2). The whole EU-Car Rental ontology has 65 classes and 170 properties (or relationships) of which 94 properties are between classes (30 of them are subsumption assertions) and 76 are properties among classes and datatypes. The maximum to-one connectivity (i.e., c) is 21 (raised by `LateReturn`). In the worst case (i.e., assuming the practical consideration introduced in Section 4.4.3.4), the longest to-one path has length 5. Consequently, the theoretical upper bound for this simulation would be

$\Theta(65 \times 21^5)$.

However, using the AMDO tool, the execution of the algorithm was immediate (less than one second in a regular desktop computer). The algorithm converges in just 5 iterations: closing 2 classes before starting (i.e., besides datatypes, there are two classes with empty list of dimensional concepts), 12 classes in the first iteration, 13 in the second one, 19 in the third one, 15 in the fourth one and 4 in the last one. In each iteration, only some classes are propagated and those previously propagated are not computed again, so, a better estimation of the answer time would be:

$$\sum_{i=1}^{l} N_i \times c_i$$

Where N_i is the number of classes not yet closed (i.e., that still have to be considered) in that iteration, c_i the maximum functional connectivity in that iteration and *l* the number of iterations (i.e., the size of the bigger to-one path in the ontology). In our example, it would raise:

$$\sum_{i=1}^{5} N_i \times c_i = (63 \times 21) + (51 \times 24) + (38 \times 54) + (19 \times 87) + (4 \times 109)$$

Result got, drastically smaller than the theoretical upper bound, is still an upper bound of the answer time of AMDO. Notice that we are considering the maximum connectivity for each class in each iteration, something completely false. However, we want to underline some important things depicted in this formula: on the one hand, we may appreciate that the value of N_i is strictly decreasing and, on the other hand, the value of c_i is never exponential. In fact, in the last iteration, its value is 109, far away from 21^5. All in all, despite the EU-Car Rental ontology size, AMDO behaves well and the answer time is good enough to develop an interactive tool.

Soundness and Completeness of the Algorithm Our algorithm is clearly sound, since it computes direct to-one relationships and propagates them according to the transitivity rule presented in Section 4.4.2.2.

Our algorithm is complete if we can assure that it converges; that is, if it would fully explore each to-one path (starting from the *end*, by identifying leaf classes, and going through the paths up to the *beginning*). We can say so if we can assure that if in a given iteration the vector M is not updated then, in any of the following iterations it will not be updated either. It can be guaranteed because:

- We detect and break deadlocks and,
- in the worst case, if *P* is the maximum number of to-one properties chained in the ontology, in each iteration the *propagate_path* function (see step 5 of Figure 4.6) will propagate, at least, one property. Indeed, the *invariant* of the main loop of the algorithm (see step 9 of Figure 4.8) guarantees that the length of each to-one path explored up to current iteration is *strictly increasing* and at most, in *P* iterations we would have explored (and propagated) all chained to-one properties in the ontology. Thus, step 5 will not be able to propagate any other property in next iterations.

– Note, however, that we need to show that, in OWL DL, a to-one path (i.e., a composite property) is compound of to-one properties. In other words, that a path compound of, at least, a to-many property cannot be asserted to raise a to-one relationship as a whole. This holds because of the *DL tree model property* [Var96], which most DL languages, such as $\mathcal{SHOIQ}(\mathcal{D})$ upon which OWL DL is based, guarantee.

4.4.2.3 Computing Measures

In Section 4.3.2 we introduced 2 different patterns with two variants. Pattern 1 and Pattern 1.a, can be computed by generic reasoning (see Section 4.4.2.5 for further details). Similar to the idea introduced in Section 4.4.2.1, for each class *F* we keep track of pairs $\{dt, \{r_1, ..., r_n\}\}$ (where every r_i is a property between *F* and *dt*). According to Def. 2, every r_i identified will give rise to a different measure.

Pattern 2 can be directly computed by matrix M (see Section 4.4.2.2):

$$\text{Measure}(F,\{dt, r_1, ..., r_n\}) := \exists B, \exists r_1, ..., r_n \mid \forall r_i, 1 \leq i \leq n,\ B \sqsubseteq \exists r_i.dt \wedge (F \in M_c < B > \ \wedge B \in M_c < F >)$$

Note that it is expressed using matrix M notation, and it means that, by transitivity, we have been able to identify *B* as a dimensional concept of *F* (i.e., as a complete fd) and viceversa.

Finally, note that Pattern 2.a does not look for complete fds, and therefore, it is not computable from matrix M. Thus, to compute Pattern 2.a, we need to introduce a new matrix, namely matrix M_1 (of *N*×*N* elements), where *N* is the number of classes in the ontology. This matrix represents, for each class *F*, the list of classes we may get to by means of *mandatory paths*. Thus, analogously to the definition of matrix M, each row of M_1 can be defined as follows:

$$\forall \{D, \{r, ..., r_n\}\} \in M_1 < F >: \begin{cases} F \sqsubseteq \exists r.D, \\ \cdots \\ F \sqsubseteq \exists r_n.D \end{cases}$$

Where M_1 is the *N*×*N* matrix, *F* and *D* are classes, *r* and r_n are composite properties and $M_1 < F >$ an operator over M_1 that retrieves a list of classes related to *F* by, at least, a mandatory (i.e., `1..N`) path. Like in the definition of matrix M, each class in this list is represented as $\{D, \{r, ..., r_n\}\}$ where *D* is the class (or datatype) itself and each r_i is a mandatory path depicted as a composite property. Now, we can compute this pattern as follows:

$$\text{Measure}(F,\{dt, r_1, ..., r_n\}) := \exists B, \exists r_1, ..., r_n \mid \forall r_i, 1 \leq i \leq n, B \sqsubseteq \exists r_i.dt \wedge (\text{F} \in M_c < B > \ \wedge B \in M_{1c} < F >)$$

Where $M_{1c} < B >$ is the equivalent operator to $M_c < B >$ of M_1. Matrix M_1 can be computed with an algorithm analogous to the one presented in Section 4.4.2.2 (see Figure 4.6) to compute M, but instead of looking for direct to-one relationships in step 4, we will look for direct mandatory relationships:

$$F \sqsubseteq \ \exists r.D,$$

Where *r* is a single property. The rest of the algorithm (i.e., propagating this knowledge) remains the same. The addition of matrix M_1 do not modify the overall computation complexity. Indeed,

the computational cost of M_1 is analogous to that of M and it has $\Theta(N \times c^l)$ as an upper bound; where N is the number of classes in the ontology; c the maximum mandatory *connectivity* (i.e., mandatory relationships of a class) and l the maximum chain of mandatory properties in the ontology. Similarly, the same practical considerations discussed in Section 4.4.2.2, as well as considerations about the soundness and completeness apply for this algorithm.

Importantly, note that, if the practical consideration introduced in Section 4.3.2.1 is assumed, it would entail that we do not need to compute matrix M_1. By using matrix M we will be able to compute if a class *F* can use a given class *B* as bridge-class (i.e., $F \in M_c < B >$), and it would be enough to compute the pattern, as any multiplicity would be allowed in the bridge-class end of the relationship. Once computed, we know data must be aggregated by a *compatible* aggregation function, prior to be inserted in the data warehouse, if $B \notin M_c < F >$.

4.4.2.4 Computing Dimension Hierarchies

The *maximal* directed graphs needed to shape the dimension hierarchies (see Section 4.3.5) are built by means of complete fds and thus, they can be created by navigating matrix M (see Section 4.3.1).

For every class identified as a dimensional concept, we create (and link) a graph node for each dimensional concept in its list, and we repeat the process for each node created. For example, consider Figure 4.7. If `branch` is identified as a dimensional concept, we create a graph node for each one of its four dimensional concepts (i.e., `country`, `string`, `branchType` and `servideDepot`). Every directed arch between `branch` and these nodes must be labeled with the property relating them. For example, the `branch` (the root node) is related to the `country` node by an edge labeled `locatedAt`. Next, we repeat the process for each of these nodes.
Note that a smart implementation to build the graphs would directly produce maximal graphs. From every path of complete fds in matrix M, we apply the algorithm described above from the selected dimensional concept that is closer to the *beginning* of the path. Note that we follow an approach similar to that in Section 4.4.2.2. Fds entail a certain order in the path (i.e., *A* determines *B* and *B* determines *C*; we say, thus, that *A* is placed at the beginning of the path and *C* at the end) and we exploit this feature to directly compute maximal graphs.

Importantly, the properties of the algorithm used to compute matrix M guarantee that this process will eventually end (we identified deadlocks and then, we know when a loop may arise, and we can guarantee that eventually, we will reach leaf classes and stop the process). Furthermore, the computational cost of this step is negligible since matrix M is already calculated and we only need to navigate and explore it.

4.4.2.5 Discussion

The specific reasoning algorithm presented in this section computes AMDO's multidimensional patterns by partially benefiting from generic DL reasoning techniques provided by DL reasoners.

Although we need to compute the fd closure by an ad hoc algorithm, most of our patterns can still be reduced to basic reasoning tasks that any commercial reasoner available in the market could answer by its querying services. In our implementation we have used FaCT++ [Hor98]. This reasoner supports OWL DL except for nomimals, but nominals fall out of our needs and do

not affect our reasoning tasks. FaCT++ provides basic tasks such as discover *class taxonomies* (i.e., given a class find all its subclasses, superclasses, ancestors or successors), *property taxonomies* (analogous to class taxonomies reasoning but over properties) and *subsumption*[3] (given two OWL DL assertions say if one is subsumed by the other). Any of the reasoning tasks mentioned in this chapter can be reduced to this three query services. Using DL reasoning have considerably reduced the complexity of our task and have facilitated the whole automation of AMDO. Indeed, most of the work done in AMDO have been reduced to reasoning over FaCT++. For example, consider the EU-Car Rental ontology used as example in this chapter (see Figure 4.2). With AMDO we have been able to identify 2069 dimensional concepts (note that it does not mean that the produced multidimensional schema contains 2069 dimensional concepts but that the number of dimensional concepts identified regarding all the ontology classes is 2069 -most of them disregarded when choosing facts of interest and filtered by selecting bases of interest; see Section 4.3-). 1375 out of this 2069 dimensional concepts were identified using reasoning (i.e., querying the EU-Car Rental TBox using FaCT++) while 694 out of 2069 were identified by our ad hoc algorithm.

4.4.3 Using Generic Reasoning Algorithms

To match our approach to traditional reasoning over ontology languages and better exploit their reasoning capabilities, in this section we introduce an approach for discovering functional dependencies from a domain ontology. Unlike our previous algorithm, we do use generic reasoning algorithms so that no ad hoc techniques and tools are needed but just a generic reasoner such as FaCT++ [Hor98] or Racer [HM01]. In this section, we make use of *DL-Lite*$_{\mathcal{A}}$, a DL of the *DL-Lite* family [CDGL^{+}07, CDGL^{+}06], which is particularly well suited for conceptual modeling due its ideal trade-off between expressive power and computational properties. The reader may find *DL-Lite*$_{\mathcal{A}}$ in Appendix A.

Importantly, this section uses *DL-Lite* notation and thus, we do not talk about OWL DL *classes*, *properties* and *datatypes* but about *DL-Lite concepts*, *roles* and *attributes*. Briefly, *DL-Lite*$_{\mathcal{A}}$ distinguishes concepts, denoting sets of objects, from *value-domains*, denoting sets of values, and roles, denoting binary relations between objects, from *attributes*, denoting binary relations between objects and values. More precisely, concepts, roles, value-domains, and attributes in *DL-Lite*$_{\mathcal{A}}$ are formed starting from atomic elements (where the distinction between *basic* and *arbitrary* elements is relevant). For a detailed view of how basic and arbitrary concepts are formed, we address the reader to appendix A.

4.4.3.1 *DL-Lite*$_{\mathcal{A}}$ Expressivity

In this section we show that, although *DL-Lite*$_{\mathcal{A}}$ is less expressible than OWL DL, the expressivity provided is enough in many real scenarios. Indeed, its ideal trade-off between expressive power and computational properties make *DL-Lite*$_{\mathcal{A}}$ a promising ontology language to be used in practice. *DL-Lite*$_{\mathcal{A}}$ is specifically tailored to capture basic ontology languages, conceptual data

[3]Note, however, that class taxonomies and properties taxonomies are computed by a specific case of subsumption (the third reasoning task enumerated), but they are typically differentiated in commercial products.

models (e.g., ER), and object-oriented formalisms (e.g., basic UML class diagrams) while keeping the complexity of reasoning low. In particular, all forms of inference over a *DL-Lite* ontology (e.g., satisfiability, logical implication, and CQ answering) can be done in polynomial time in the size of the TBox, whereas ontology satisfiability, instance checking, and answering conjunctive queries can all be done in LogSpace with respect to data complexity [CDGL$^+$07, PLC$^+$08]. To illustrate its expressivity power, in this section we show how a *DL-Lite*$_\mathcal{A}$ TBox can capture a UML conceptual diagram (in short, UML-CD).

Consider the UML-CD in Figure 4.2. Intuitively, each UML class in the diagram is represented by an atomic concept in the TBox, each UML (binary)[4] association by an atomic role, and each UML attribute by an atomic attribute (we assume here that the name of the class is part of the attribute name). We describe how suitable TBox assertions capture the constraints imposed on the domain by the UML-CD:

- A generalization between two classes is represented by means of an inclusion assertion between the corresponding concepts, e.g., $\mathsf{Reservation} \sqsubseteq \mathsf{RentalAgreement}$.
- To represent the domain (i.e., first component) and range (i.e., second component) of an association P, we use $\exists P$ and $\exists P^-$, respectively. E.g., to represent that the domain of the LocatedAt association is contained in Branch we use $\exists\mathsf{LocatedAt} \sqsubseteq \mathsf{Branch}$, and to represent that the range is contained in Country, we use $\exists\mathsf{LocatedAt}^- \sqsubseteq \mathsf{Country}$.
- To represent domain and range of an attribute U, we use $\delta(U)$ and $\rho(U)$, respectively. E.g., to represent that bestPrice is an attribute with domain RentalAgreement and range Money, we use $\delta(\mathsf{bestPrice}) \sqsubseteq \mathsf{RentalAgreement}$ and $\rho(\mathsf{bestPrice}) \sqsubseteq \mathsf{Money}$.
- To capture mandatory participation in an association (i.e., a min. multiplicity 1), we use e.g., $\mathsf{Branch} \sqsubseteq \exists\mathsf{IsOfType}$ or $\mathsf{Customer} \sqsubseteq \exists\mathsf{Makes}^-$.
- To capture functionality of an association (i.e., a max. multiplicity 1), we use e.g., $(\mathsf{funct\ LocatedAt})$ or $(\mathsf{funct\ Makes}^-)$.
- Finally, to capture disjointness between classes, we use negation on concepts, as e.g., in $\mathsf{Car} \sqsubseteq \neg\mathsf{Branch}$.

4.4.3.2 Functional Dependencies over *DL-Lite*$_\mathcal{A}$ Ontologies

The observations introduced in Section 4.4.1, lead us to the following definition of when a fd between two *DL-Lite*$_\mathcal{A}$ concepts holds in a given interpretation. We make use of the notion of *role chain* $Q_1 \circ \cdots \circ Q_n$ of basic roles, interpreted as the composition of the binary relations corresponding to the roles. Formally, for an interpretation $\mathcal{I}$, we have that

$$(Q_1 \circ \cdots \circ Q_n)^\mathcal{I} \;=\; Q_1^\mathcal{I} \circ \cdots \circ Q_n^\mathcal{I}.$$

We can then apply concept and role constructs to role chains (instead of basic roles), and the semantics naturally extends the one for the case of basic roles. When we talk about a role chain

[4] Associations of arity greater than 2 can be handled through *reification* [CLN98, CDGL$^+$09].

$Q_1 \circ \cdots \circ Q_n$ *over* a TBox $\mathcal{T}$, we intend that for each $i \in \{1, \ldots, n\}$, at least one of Q_i or $\mathsf{Inv}(Q_i)$ appears in $\mathcal{T}$. Similarly, a basic concept *over* $\mathcal{T}$ is any basic concept that can be constructed from atomic concepts and roles in $\mathcal{T}$.

Definition 5. *Given a DL-Lite$_\mathcal{A}$ TBox $\mathcal{T}$ and two concepts C_1 and C_2 over $\mathcal{T}$, the expression $C_1 \rightarrow C_2$ is called a* functional dependency *(over $\mathcal{T}$). Given an interpretation $\mathcal{I}$ of $\mathcal{T}$, we say that $C_1 \rightarrow C_2$* is satisfied in $\mathcal{I}$*, denoted $\mathcal{I} \models C_1 \rightarrow C_2$, if there is a role chain $S = Q_1 \circ \cdots \circ Q_n$ over $\mathcal{T}$, with $n > 0$ and such that for each object $o_1 \in C_1^\mathcal{I}$ there is exactly one object $o_2 \in C_2^\mathcal{I}$ such that $(o_1, o_2) \in S^\mathcal{I}$.*

Intuitively, the definition requires that each instance of C_1 determines a unique instance of C_2 by means of some chain of roles in $\mathcal{T}$. Note that the inverse of such a role chain corresponds to a path in the path-based identification constraints in [CDGL$^+$08]. Importantly, note that the notion of fd in an arbitrary *DL-Lite$_\mathcal{A}$* ontology corresponds to the complete fd concept introduced previously in Section 4.2, upon which most AMDO multidimensional patterns are based on.

From the above definition, it is immediate to verify that the following properties hold for fds over $\mathcal{T}$ involving concepts C_1, C_2, C_3, and for every interpretation $\mathcal{I}$:

Asserted:	If $\mathcal{I} \models (\mathsf{funct}\ Q)$, then $\mathcal{I} \models \exists Q \rightarrow \exists \mathsf{Inv}(Q)$.	(4.1)
Transitivity:	If $\mathcal{I} \models C_1 \rightarrow C_2$ and $\mathcal{I} \models C_2 \rightarrow C_3$, then $\mathcal{I} \models C_1 \rightarrow C_3$.	(4.2)
Left-inclusion:	If $\mathcal{I} \models C_1 \rightarrow C_2$ and $C_3^\mathcal{I} \subseteq C_1^\mathcal{I}$, then $\mathcal{I} \models C_3 \rightarrow C_2$.	(4.3)
Right-inclusion:	If $\mathcal{I} \models C_1 \rightarrow C_2$ and $C_2^\mathcal{I} \subseteq C_3^\mathcal{I}$, then $\mathcal{I} \models C_1 \rightarrow C_3$.	(4.4)

We are now interested in determining when an fd is logically implied by the assertions in the TBox, i.e., the fd is necessarily satisfied in all models of the TBox.

Definition 6. *Given a DL-Lite$_\mathcal{A}$ TBox $\mathcal{T}$, we say that a fd $C_1 \rightarrow C_2$ over $\mathcal{T}$ is logically implied by $\mathcal{T}$, denoted $\mathcal{T} \models C_1 \rightarrow C_2$, if $C_1 \rightarrow C_2$ is satisfied in every model of $\mathcal{T}$.*

In the following, we will restrict the attention to functional dependencies between basic concepts only, since in *DL-Lite$_\mathcal{A}$* negation is used only to assert disjointness. Exploiting the restrictions in the expressive power of *DL-Lite$_\mathcal{A}$*, we can show the following property.

Proposition 1. *Given a DL-Lite$_\mathcal{A}$ TBox $\mathcal{T}$ and two basic concepts B_1, B_2 over $\mathcal{T}$, we have that $\mathcal{T} \models B_1 \rightarrow B_2$ if and only if there is a role chain $S = Q_1 \circ \cdots \circ Q_n$ over $\mathcal{T}$, where $n > 0$, such that (i) $\mathcal{T} \models (\mathsf{funct}\ Q_i)$, for $i \in \{1, \ldots, n\}$, (ii) $\mathcal{T} \models B_1 \sqsubseteq \exists S$, and (iii) $\mathcal{T} \models \exists S^- \sqsubseteq B_2$.*

Proof. The "if" direction is a direct consequence of Definitions 5 and 6.

For the "only-if" direction, we observe that properties (ii) and (iii) follow from the canonical model property of the DL of the *DL-Lite* family [CDGL$^+$07], and in particular of *DL-Lite$_\mathcal{A}$* [PLC$^+$08]. Instead, for property (i), we exploit the tree-model property of *DL-Lite$_\mathcal{A}$*. This property is shared by most DL [BCM$^+$03], and states that if a TBox admits a model, then it admits one that has the structure of a tree (where the nodes of the tree are the elements of the interpretation domain, and the edges are determined by the role instances). In a tree-model, it is ruled out that from a given object o_1 there are two *different* paths labeled with the same roles that lead to the same object o_2. Hence, a chain of roles is forced to be functional in the TBox $\mathcal{T}$ only if also all of the component roles are forced to be functional. □

We can now exploit Proposition 1 to obtain a simple technique that derives pairs of concepts B_1, B_2 such that $\mathcal{T} \models B_1 \rightarrow B_2$. The technique is based on turning the properties (4.1)–(4.4) above into the following inference rules, which derive new fds from existing ones for a given TBox $\mathcal{T}$ and for basic concepts B_1, B_2, and B_3 over $\mathcal{T}$:

Asserted:	If $\mathcal{T} \models (\mathsf{funct}\ Q)$, then $\mathcal{T} \models \exists Q \rightarrow \exists \mathsf{Inv}(Q)$.	(4.5)
Transitivity:	If $\mathcal{T} \models B_1 \rightarrow B_2$ and $\mathcal{T} \models B_2 \rightarrow B_3$, then $\mathcal{T} \models B_1 \rightarrow B_3$.	(4.6)
Left-inclusion:	If $\mathcal{T} \models B_1 \rightarrow B_2$ and $\mathcal{T} \models B_3 \sqsubseteq B_1$, then $\mathcal{T} \models B_3 \rightarrow B_2$.	(4.7)
Right-inclusion:	If $\mathcal{T} \models B_1 \rightarrow B_2$ and $\mathcal{T} \models B_2 \sqsubseteq B_3$, then $\mathcal{T} \models B_1 \rightarrow B_3$.	(4.8)

We consider these rules to be applied exhaustively to all basic concepts over $\mathcal{T}$. Soundness of the rules follows directly from the corresponding properties above, while completeness is a consequence of Proposition 1. Moreover, since the number of basic concepts over $\mathcal{T}$ is finite, rule application clearly terminates.

We observe that the "left-inclusion" and "right-inclusion" rules propagate fds according to the TBox inclusion assertions, and as such they provide an interaction between functional and inclusion dependencies. In general, implication is undecidable when combining functional and inclusion dependencies [AHV95], our setting is much simpler, since we only consider unary inclusion[5] and functional dependencies [CKV90]. Note also that there is no counterpart of the augmentation rule for fd implication in the relational setting, since we deal only with unary functional dependencies.

Functional Dependencies Considering Datatypes In AMDO, we also need to discover functional dependencies involving datatypes at the right-hand side (i.e., which datatypes are functionally identified by an ontology concept). In *DL-Lite* datatypes are represented by means of value-domains and attribute expressions (see Section A for further details). Accordingly, we extend the functional dependency definition as follows:

Definition 7. *In DL-Lite, a functional dependency:* C $\rightarrow$ F *(where* C *is either an atomic or basic concept and* F *a general value-domain) holds if exists an atomic or basic concept* D *and a general concept attribute* U_D *such that* C *functionally determines* D *and* D *functionally determines* F *by means of* U_D.

Intuitively, *C* will functionally determine *F* if *C* functionally determines the concept (i.e., *D*) such that *F* is a mono-valued attribute of *D*. The semantics of this definition can be defined as follows. An interpretation *I* is a *model* of a functional dependency $C \rightarrow F$ if $\exists D,\ \exists U_D$, such that $C \rightarrow D$ and U_D is a function and $\delta(U_D) \sqsubseteq D$ and $\rho(U_D) \sqsubseteq F$.

4.4.3.3 Discovering FDs in *DL-Lite*$_\mathcal{A}$

In this section, we propose an algorithm to discover all the fd's logically implied by a *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$, and which exploits the reasoning capabilities of a *DL-Lite*$_\mathcal{A}$ reasoner. Our algorithm

[5]Note that, in *DL-Lite*, role inclusions are restricted so as not to interact with functionality.

starts from the asserted fds (see inference rule (4.5)), and then computes the closure of the asserted fds w.r.t. the remaining rules. We recall that the asserted fds are simply $\exists Q \rightarrow \exists \mathsf{Inv}(Q)$, for each functional role Q in the TBox.

The closure of the asserted fd's is computed as follows. First, we identify the sets $\mathcal{B}_d$ and $\mathcal{B}_r$ of all basic concepts that appear respectively in the domain and range of a functional basic role. To do so, we scan all functional basic roles, and for each such role Q and each basic concept B over $\mathcal{T}$, if $\mathcal{T} \models B \sqsubseteq \exists Q$ then we add B to $\mathcal{B}_d$, and if $\mathcal{T} \models B \sqsubseteq \exists \mathsf{Inv}(Q)$ then we add B to $\mathcal{B}_r$.

Then, for each pair of basic concepts $B_d \in \mathcal{B}_d$ and $B_r \in \mathcal{B}_r$, we need to check whether $\mathcal{T} \models B_d \sqsubseteq \exists S.B_r$, for some chain S of functional basic roles[6]. To perform such a check, we have to face the difficulty that in principle we have to try all possible lengths n of the chain S, and all the possible ways of composing it by means of functional basic roles. To tackle the latter issue, we introduce a new atomic role U in $\mathcal{T}$, and for each basic role Q such that $(\mathsf{funct}\ Q)$ is in $\mathcal{T}$, we add to $\mathcal{T}$ the assertion $Q \sqsubseteq U$[7]. Hence, U acts as a super-role of all functional basic roles in $\mathcal{T}$ and it is sufficient to consider S as a chain of U n times with itself, for suitable values of n. We then iterate over n until we have tried a sufficiently large value (see below).

In *DL-Lite*$_\mathcal{A}$ we cannot directly check the logical implication $\mathcal{T} \models B_d \sqsubseteq \exists S.B_r$, with $S = U \circ \cdots \circ U$ (for some fixed length n of the chain). However, we can easily encode such a check into the problem of computing the certain answers to the CQ

$$Q^n_{B_d,B_r}() \leftarrow B_d(a), U(a,x_1), U(x_1,x_2), \ldots, U(x_{n-1},x_n), B_r(x_n)$$

over the ABox constituted only by the assertion $B_d(a)$. Indeed, since the only fact in the ABox is one involving B_d, it is not possible to satisfy the atoms $U(x,x')$ and $B_r(x_n)$ with facts in the ABox. Hence, the only case in which the answer to the query could anyway be positive, is when the whole body of $Q^n_{B_d,B_r}$ can be rewritten to just $B_d(a)$ [CDGL$^+$07]. And this is precisely the case when $\mathcal{T} \models B_d \sqsubseteq \exists S.B_r$. Notice that we are taking advantage of the query rewriting technique for *DL-Lite*$_\mathcal{A}$, which exploits the knowledge contained in the TBox of the ontology to actually compute the right-inclusion and left-inclusion inference rules with the *DL-Lite*$_\mathcal{A}$ reasoner.

The question that we still need to address is which is the maximum bound for the length n of the role chain S. If the ontology does not contain functional cycles, we should stop when no new answer is retrieved. However, it is not uncommon to find functional cycles in a real world ontology. In this case, we should stop looking for functional dependencies originating at a concept B_d when (i) for a given length no results are provided, or (ii) no new concepts are proposed with regard to previous iterations. Intuitively, the reason is that in *DL-Lite* all the roles involved in a functional path must be functional as well, and hence at each step we must get, at least, one new concept of the longest path. Otherwise we are looping in a cycle.

More precisely, let B_d be the concept from where we start looking for functional dependencies and $\mathcal{D}_i$ the set of concepts that we have already identified up to iteration i. Let B_r be a concept functionally dependent on B_d and not yet identified.

- If B_d functionally identifies B_r, then there must be a role chain S' that connects $\mathcal{D}_i$ and

[6]The concept $\exists S.B_r$ is called a qualified existential and is interpreted as $\{o \mid \exists o'.(o,o') \in S^\mathcal{I} \wedge o' \in B_r^\mathcal{I}\}$. It is not a *DL-Lite*$_\mathcal{A}$ concept.

[7]Note that this is compatible with the conditions in the *DL-Lite*$_\mathcal{A}$ ontology definition introduced in Appendix A.

B_r. Let $\mathcal{D}_?$ be the concepts along S'. Note that $\mathcal{D}_i$ and $\mathcal{D}_?$ are disjoint, since the $\mathcal{D}_i$ contains concepts already visited, while $\mathcal{D}_?$ does not.

- At least, one concept of $\mathcal{D}_i$ and one concept of $\mathcal{D}_?$ must be directly related. Let's call them B_i and $B_?$, respectively.
- If along the $i+1$-th iteration we do not identify any new concept, then, $B_? \in \mathcal{D}_i$, which contradicts our initial assumption that $\mathcal{D}_i$ and $\mathcal{D}_?$ are disjoint.

Computational Complexity An upper bound for the maximum number of queries we will have to pose is $\Theta(n \cdot |\mathcal{B}_d| \cdot |\mathcal{B}_r|)$, where n is the length of the maximum chain of functional roles. However, this is an upper bound not reachable in practice because those concepts that do not get any new solution for paths of length i, are not queried in the next iterations. In most cases (considering ontologies in real applications), most of the concepts will end with n being rather small (indeed, the longest functional path in the EU-Car Rental ontology has size 5). Furthermore, if in a previous iteration we have shown that $\mathcal{T} \models B_d \rightarrow B_r$, then, this pair will not be checked again in next iterations.

We note that the computational complexity of the rewriting algorithm of *DL-Lite*$_\mathcal{A}$ is exponential in the size of the query. However, this turns out to be manageable for real ontologies, given that the number of times we have to concatenate role U is relatively small.

4.4.3.4 Practical Consideration

The fds as introduced in Section 4.4.3.2 are conceived for arbitrary *DL-Lite*$_\mathcal{A}$ ontologies. But similar to the discussions introduced in Sections 4.3.1.1 and 4.3.2.1, we can compute functional dependencies without a mandatory participation.

Nevertheless, there are some interesting additional considerations to be made regarding *DL-Lite*$_\mathcal{A}$ ontologies derived from conceptual schemas. Consider the UML diagram depicted in Figure 4.2 and let $\mathcal{T}_{eu}$ be the corresponding *DL-Lite*$_\mathcal{A}$ TBox. Specifically, the hasAssigned association results in the following assertions:

$$\begin{array}{rcll} \exists \mathsf{hasAssigned} & \sqsubseteq & \mathsf{RentalAgreement} & \\ \exists \mathsf{hasAssigned}^- & \sqsubseteq & \mathsf{Assignment} & (\mathsf{funct\ hasAssigned}) \\ \mathsf{Assigment} & \sqsubseteq & \exists \mathsf{hasAssigned}^- & (\mathsf{funct\ hasAssigned}^-) \end{array}$$

According to Definition 6, we have that $\mathcal{T}_{eu} \models Assignment \rightarrow RentalAgreement$, (but $\mathcal{T}_{eu} \not\models RentalAgreement \rightarrow Assignment$), since RentalAgreement does not have a mandatory participation in hasAssertion. As discussed in Section 4.4.3.2, the mandatory participation is needed in arbitrary *DL-Lite*$_\mathcal{A}$ ontologies to avoid discovering meaningless functional dependencies. For example, consider the following TBox $\mathcal{T}$:

$$\exists P_1 \sqsubseteq A_1, \quad \exists P_1^- \sqsubseteq A_2, \quad (\mathsf{funct}\ P_1), \quad \exists P_2 \sqsubseteq A_3 \quad \exists P_2^- \sqsubseteq A_4 \quad (\mathsf{funct}\ P_2)$$

Without requiring the mandatory participation we would have that $\mathcal{T} \models A_1 \rightarrow A_4$. Indeed, both P_1 and P_2 are functional, and therefore, in every model of $\mathcal{T}$, every instance of A_1 is connected to at most one instance of A_4 via the role chain $P_1 \circ P_2$.

However this scenario cannot happen in ontologies derived from conceptual schemas. In an UML-CD (or ER schema) two classes are supposed to be disjoint unless they are related by a generalization relationship and furthermore, strict role-typing is assumed (i.e., exactly the opposite assumptions to those in arbitrary *DL-Lite*$_\mathcal{A}$ ontologies). Hence, when translating UML-CDs to *DL-Lite*$_\mathcal{A}$ it makes sense to identify functional dependencies from non mandatory relationships. With this aim, we redefine the functional property definition presented in Section 4.4.3.2 for *DL-Lite*$_\mathcal{A}$ ontologies derived from conceptual schemas:

Definition 8. *Given a DL-Lite*$_\mathcal{A}$ *TBox* $\mathcal{T}$*, an atomic role* P *in* $\mathcal{T}$ *is* strict role-typed in $\mathcal{T}$ *if there is a single atomic concept* A_1 *such that* $\exists P \sqsubseteq A_1$ *is in* $\mathcal{T}$*, and a single atomic concept* A_2 *such* $\exists P^- \sqsubseteq A_2$ *is in* $\mathcal{T}$*. The concepts* A_1 *and* A_2 *are called respectively the* domain *and* range *of* P*. A DL-Lite*$_\mathcal{A}$ *TBox* $\mathcal{T}_c$ *is called DL-Lite*$_\mathcal{A}$ conceptual schema *if each atomic role* P *is strict role-typed in* $\mathcal{T}$ *and for each pair of atomic concepts* A_1*,* A_2*, either* A_1 *and* A_2 *are disjoint (i.e.,* $\mathcal{T}_c \models A_1 \sqsubseteq \neg A_2$*) or* A_1 *and* A_2 *participate in the same concept taxonomy (i.e., there is an atomic concept* A *such that* $\mathcal{T}_c \models A_1 \sqsubseteq A$ *and* $\mathcal{T}_c \models A_2 \sqsubseteq A$*).*[8]

Definition 9. *Given a DL-Lite*$_\mathcal{A}$ *conceptual schema* $\mathcal{T}_c$*, two basic concepts* B_1 *and* B_2 *over* $\mathcal{T}_c$*, and an interpretation* $\mathcal{I}$ *of* $\mathcal{T}$*, we say that* $\mathcal{I} \models B_1 \rightarrow B_2$ *if there is a chain* $S = Q_1 \circ \cdots \circ Q_n$*, with* $n > 0$*, of roles that are strict role-typed in* $\mathcal{T}$*, where* B_1 *is the domain of* Q_1*,* B_2 *is the range of* Q_n*, and such that for each object* $o_1 \in \exists Q_1^\mathcal{I}$ *there is exactly one object* $o_2 \in C_2^\mathcal{I}$ *such that* $(o_1, o_2) \in S^\mathcal{I}$*.*

Roughly speaking, we may relax the mandatory participation of B_1 in S thanks to the implicit constraints we may find in a *DL-Lite* conceptual schema.

The semantics of Definition 9 can be defined as follows. An interpretation *I* is a *model* of a functional dependency $C \rightarrow D$ if $\exists C_0, ..., C_n$, $\exists R \; where \; R \equiv (R_1 \circ \ldots \circ R_n)$ with $n \geq 1$, such that $R^\mathcal{I}$ is a function and $(\exists R)^\mathcal{I} \sqsubseteq C^\mathcal{I}$ and $(\exists R^-)^\mathcal{I} \sqsubseteq D^\mathcal{I}$ and $\forall R_i \; (\exists Ri)^\mathcal{I} \sqsubseteq C_{i-1}^\mathcal{I}$ and $(\exists Ri^-)^\mathcal{I} \sqsubseteq C_i^\mathcal{I}$ with $1 \leq i \leq n$-1.

We can take advantage of the algorithm presented in Section 4.4.3.3 to discover functional dependencies over *DL-Lite*$_\mathcal{A}$ conceptual schemas by adding the following two assertions for each functional role P with domain A_1 and range A_2:

$$A_1 \sqsubseteq \exists P \qquad\qquad A_2 \sqsubseteq \exists P^-$$

Indeed, we are adding a mandatory participation for the role to its domain and range. In terms of UML-CDs, we are modifying the cardinality of the relationship and making it mandatory. With this trick we fulfill Definition 5 of fd and despite this change, the semantics with regard to fd's will not change and the results we get are sound. Notice that these assertions are only needed while discovering functional dependencies and they have to be retracted once the algorithm has finished.

This trick cannot be applied for arbitrary *DL-Lite* ontologies. In an arbitrary *DL-Lite*$_\mathcal{A}$ ontology disjointness of concepts cannot be assumed and therefore, adding the domain and range assertion we would modify the semantics of the model also with respect to fds. As a consequence, we could identify false fds.

[8] Note that the concept A may coincide with A_1 or A_2.

4.4.3.5 Computing AMDO Multidimensional Patterns

In previous sections we have shown how to compute complete fds from a *DL-Lite*$_\mathcal{A}$ ontology. Now, we discuss how this knowledge must be applied to compute the multidimensional patterns presented in Section 4.3.

Computing Dimensional Concepts In our current framework, it is immediate to compute the list of dimensional concepts for each ontology concept. According to Section 4.3.1, *B* is a dimensional concept of *A* if *B* is a complete fd of *A*.

In an arbitrary *DL-Lite*$_\mathcal{A}$ ontology, note that the fd notion is equivalent to that of complete fd and therefore, this pattern is computed by the algorithm presented in Section 4.4.3.3. In case of a *DL-Lite*$_\mathcal{A}$ conceptual schema (see Section 4.4.3.4), we have two possible scenarios. If the practical consideration discussed in Section 4.3.1.1 is assumed then, the output of the algorithm introduced in Section 4.4.3.4 must be considered; otherwise, the pattern must be computed by the algorithm presented in Section 4.4.3.3.

Finally, note that these considerations also hold for computing the dimensional hierarchies (see Section 4.3.5).

Computing Measures We introduced two different patterns, with a slightly variant each, to compute measures. Similar to our previous algorithm. Pattern 1 and 1a may be computed by querying a generic reasoner.

Consider now an arbitrary *DL-Lite*$_\mathcal{A}$ ontology. Pattern 2 must be computed by the algorithm introduced in Section 4.4.3.3. Once the functional dependencies have been computed, we identify concepts that may play the bridge-concept role (i.e., those fulfilling Pattern 1). Then, an attribute a, directly related to a concept B is a potential measure of a given concept F if $F \rightarrow B$ and $B \rightarrow F$. Note, however, that Pattern 2a cannot be fully computed by any of the algorithms introduced. We can compute if $B \rightarrow F$ (where B and F are concepts and B fulfills Pattern 1a) by the algorithm introduced in Section 4.4.3.3, but the first assertion (i.e., $F \sqsubseteq \exists r.B$) does not express a functional dependency and thus, not computable by our algorithms. Nevertheless, note that we can still compute it by an analogous algorithm to that presented in Section 4.4.3.3:

First, we identify the sets $\mathcal{B}_d$ and $\mathcal{B}_r$ of all basic concepts that appear respectively in the domain and range of all the ontology roles. Then, we introduce a new atomic role U in $\mathcal{T}$, and for each basic role Q in $\mathcal{T}$, we add to $\mathcal{T}$ the assertion $Q \sqsubseteq U$. Finally, we need to compute the certain answers to the CQ

$$Q^n_{B_d,B_r}() \leftarrow B_d(a), \exists U(a,x_1), \exists U(x_1,x_2), \ldots, \exists U(x_{n-1},x_n), B_r(x_n)$$

over the ABox constituted only by the assertion $B_d(a)$. Where the rest of considerations made in Section 4.4.3.3 still hold. Importantly, note that the number of roles tried in the query rewriting technique of *DL-Lite*$_\mathcal{A}$ is considerably bigger for this case.

If we consider a *DL-Lite*$_\mathcal{A}$ conceptual schema (see Section 4.4.3.4) and the practical consideration made in Section 4.3.2.1, Pattern 2 is computed by the algorithm introduced in Section 4.4.3.4. Then, an attribute a, directly related to a concept B is a potential measure of a given concept F if $F \rightarrow B$ (by considering roles enforced to be mandatory) and $B \rightarrow F$ (by not considering them). Finally, in this case, Pattern 2a can be easily computed by the algorithm in Section

4.4.3.3. Thus, an attribute a, directly related to a concept B is a potential measure of a given concept F if $B \rightarrow F$.

4.4.3.6 Discussion

We consider now the full EU Car-Rental case study (see Appendix B) of which we computed the closure of functional dependencies by applying the algorithm presented in Section 4.4.3.3. This case study has 65 concepts and 170 relationships (30 of which are subsumption assertions between classes).

A total of 2069 functional dependencies were found. The total computation time was 2.332 seconds[9] from which, 0.080 seconds were used by the reasoner to classify the ontology, 0.006 seconds were required to query for candidate domains and ranges for the functional paths (i.e., the $\mathcal{B}_d$ and $\mathcal{B}_r$ sets presented in Section 4.4.3.3), and the remaining time (2.246 seconds) was used to compute the functional dependencies.

To run these tests we used the FaCT++ reasoner. We note that FaCT++ doesn't support answering conjunctive queries, As a workaround, we had to devise a subsumption verification query which was $true$ if and only if the CQ of Section 4.4.3.3 is non-empty, and $false$ otherwise. The query which complies with this specification is $(B_1 \sqsubseteq \exists U.\exists U.\cdots.B_2)?$ were B_1 corresponds to the current domain to be tested, the number of nested U's corresponds to the number of U atoms in the original CQ and B_2 corresponds to the range of the functional path to be tested.

According to Def. 1 and Def. 2, we also computed the functional paths (i.e., the composition of roles) that verified the query shown above. In order to do this, we sent additional queries to the reasoner, whenever we had verified the existence of a path of length n. In these queries we replaced the n'th occurrence of role U in the qualified existential chain with each of the sub-roles of U. In this case, 41.039 seconds were spent pinning-down the specific roles which triggered the existence of these paths.

4.5 Computing Bases

In this section we propose an approach for discovering meaningful bases (i.e., minimal keys) from domain ontologies. Currently, several works for discovering (minimal) keys from relational sources[10] are available. These approaches could be used, for example, to discover bases in the MDBE approach introduced in Chapter 3. In this section, we discuss a novel approach benefiting from the conceptual knowledge available. Previous experiences show that working at the data level is computationally expensive in the general case. Surprisingly, despite this fact, none of the current approaches consider (and exploit) the conceptual knowledge available for this purpose. Thus, following with the spirit of this thesis, we propose to guide the key discovering process at the conceptual level. Furthermore, we also introduce a set of pruning rules for improving this task performance by reducing the number of key hypotheses generated and to be verified with data. To our knowledge, this is the first approach introduced in the literature proposing to lead

[9]The computer used in this test was equipped with an Intel Core 2 Duo 1.33 GHz processor and 1'99 GB of RAM.

[10]Note that the key concept has been traditionally linked to the database theory.

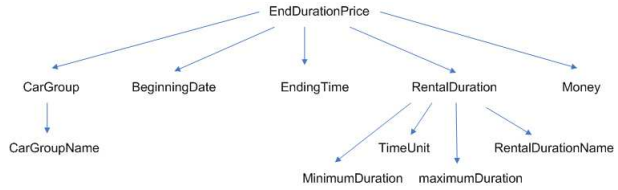


Figure 4.9: AMDO: the FD-tree computed for the `EndDurationPrice` concept

this task at the conceptual level. As result, it is able to generate less hypotheses to be validated with data and therefore, it performs better than current approaches.

4.5.1 Foundations

Importantly, our approach discovers bases guided by the domain knowledge captured in the ontology. First, we formally discuss the implications of the key (i.e., base) concept at the conceptual level. In this section we make use of a generic conceptual notation, but by *concepts* we refer to an ontology concept (classes in OWL notation) or a datatype, and by *relationships* to ontology roles (or properties in OWL notation). We denote concepts by uppercase letters from the beginning of the alphabet (such as *A* and *B*) and sets of concepts by uppercase letters from the end of the alphabet (such as *Y* and *Z*).

Definition 10. *We say that a set of concepts* Z *is a base of a concept* A*, if there is an injective function from* A *to* Z *(i.e., a mandatory one-to-one relationship).*

Note that we do not ask for a mandatory participation of *Z* in *A* (i.e., a bijective function), since some values of the base could not have a correspondence into the identified concept. This definition is equivalent to other concepts previously introduced that we could consider equivalent. For example, the one-to-one relationships introduced in [Che76] or the *reference mode* (mandatory one-to-one relationships) in ORM [HM08]. Note that this is sound, since, as previously discussed in Section 4.3.4, the multidimensional base concept is equivalent to the traditional concept of key and, indeed, these two concepts play the key role in the ER and ORM languages, respectively.

Def. 10 entails that both, the concept identified functionally depends on the base, and the base functionally depends on the concept, giving rise to the following proposition:

Proposition 2. *A set of concepts* Z *is a base of a given concept* A *if and only if* Z *uniquely or functionally determines the values of* A *(i.e.,* $Z \rightarrow A$*) and* A *functionally determines the values of* Z *(i.e.,* $A \rightarrow Z$*).*

This is sound with previous work presented in the literature to discover keys. Previous approaches work at the data level and a key is defined as a specific kind of functional dependency (i.e., a *minimal* set of attributes that uniquely identify the whole *tuple*). Moreover, according to the relational model assumptions, each relation row is supposed to represent a different instance

[Cod90], giving rise as a whole to a one-to-one relationship (furthermore, since a candidate key does not allow *NULL* values, it is also mandatory).

The first step in our approach requires to compute the asserted functional dependencies (in short, fd's) in the domain ontology. To do so, we may take advantage of any of the algorithms introduced in Section 4.4. Eventually, for each domain concept we get a *directed tree of functional dependencies* like the one shown in Figure 4.9. This example refers to the `endDurationPrice` concept of the EU-Car Rental ontology (see appendix B). It represents the final price charged for the car renting to the customer. As shown in the figure, it has 10 fd's: the `car group` (i.e., kind of car rented) and the `car group name`, the `beginning` and `ending` date of the rental agreement, the final price (i.e., `money`) and the rental agreement `duration` (which consists of the `rental duration name` and a `time unit` used to express the `minimum` and `maximum duration` allowed for that rental). From here on, we use *FD-tree* to denote this tree; *root concepts* to denote those concepts in the first level of the tree (in our example: `car group`, `beginning` and `ending` date, `duration` and `money`), and *FD-concepts* to denote the rest of concepts in the tree. We also make use of typical tree notation, and we talk about depth levels, ancestors and descendants.

Next, once fd's asserted in the ontology have been computed, for every ontology concept A, we aim to find the set of concepts Z such that Z is a key of A. According to Prop. 2, Z is a key of A if and only if $A \rightarrow Z$. Thus, we only need to generate combinations of concepts among those functionally identified by A. For instance, in our example, it means that all the possible keys of `endDurationPrice` are combinations of concepts between its fd's (i.e., those represented in the figure). Note the benefits of this proposition. Traditionally, when looking for keys, the searching space is formed by all the attribute combinations up to size N, where N is the number of attributes in the database (i.e., 2^N combinations), but by using ontological knowledge we reduce the searching space to 2^P, where P is the number of concepts functionally dependent on A.

4.5.1.1 Necessary Conditions

A naive approach for discovering bases would entail generating all the combinations in our searching space (i.e., 2^P combinations) and sample data to verify them. However, despite we have reduced considerably the searching space, we may have computational problems for concepts having many fd's, since the searching space is still exponential. For example, in a middle-sized ontology like the EU-Car Rental we obtained concepts with more than 80 functional dependencies (see the second column of Table 4.1 discussed in Section 4.3.3). Furthermore, querying the data may be expensive for large tables. For this reason, we further exploit the conceptual knowledge we have before verifying key hypotheses with data. Specifically, we take advantage of the well-known fd theory.

Given a set of fd's F, a *minimal cover* [AHV95] of F is a set F' of fd's such that:

- (i) Each dependency in F' has the form $Z \rightarrow C$, where C is a concept,
- (ii) $F' \equiv F$,
- (iii) no proper subset of F' implies F and
- (iv) for each dependency $Z \rightarrow C$ in F' there is no $W \subset Z$ such that $F \models W \rightarrow C$.

In our approach, for every ontology concept A, we define F as the set of fd's of the kind $Z \rightarrow A$, (where Z is compound of concepts in the FD-tree of A). Essentially, we look for a minimal cover of F, because we aim to minimize the number of queries posed to the database (i.e., it is the minimum set of fd's to be verified as bases with data). The rest of fd's in F can be generated and verified from F' in polynomial time by the *Armstrong axioms* [RG03] (i.e., a fd of the kind $Z \rightarrow A$ holds if and only if fd $\in F'^{+}$). Nevertheless, as discussed later, we will not be interested in this kind of fd's either, since they are not minimal and thus, they are not bases.

In our approach, (i) is guaranteed by Prop. 2. As discussed in previous section, the fd's we may find in an ontology are of the kind $A \rightarrow B$, (where A and B are concepts), and in our algorithm we generate combinations of concepts in the left-end of the fd (i.e., left-end multi-attribute fd's). Regarding (ii), F will be equivalent to the set of fd's determining A that we can infer from knowledge contained in the ontology (from where we compute the initial knowledge that guides the search) and data (from where verify combinations proposed). Thus, if a key cannot be inferred from the ontology and verified with data, we will not be able to identify it. Section 4.5.3 guarantees that our algorithm is complete with regard to knowledge captured in the ontology and data. Finally, (iii) and (iv) guarantee that the set of fd's in F' is minimal. Note that these two conditions are desirable for our purpose, as they enforce the *minimality* property of bases. *Consequently, fd's in the minimal cover are the only base candidates to be considered.* In our approach, these two items lead to two necessary conditions a fd must fulfill prior to be verified as a base with data. Before introducing them, we recall the *Armstrong axioms*, introduced in Section 4.4.1, used to infer all the fd's that can be computed from a given set F of fd's (i.e., the F closure or F^{+}) [AHV95], since we will need them to proof our propositions:

- (reflexivity) If $Y \subseteq X$, then $X \rightarrow Y$,
- (augmentation) If $X \rightarrow Y$, then $X \cup Z \rightarrow Y \cup Z$
- (transitivity) If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Here, we also introduce the pseudo-transitivity rule [AHV95] (that can be easily derived from the three inference rules discussed above), since we will need it later:

- (pseudo-transitivity) If $X \rightarrow Y$ and $YW \rightarrow Z$, then $XW \rightarrow Z$.

Proposition 3. *Let* Z *and* W *be sets of concepts functionally dependent on a given concept* A. *If* $Z \subseteq W$, *and* Z *is a key of* A *then,* W, *despite functionally determining* A, *it is not minimal and then, it does not belong to the minimal cover; there exists a subset of* W *(i.e.,* Z*) functionally determining* A.

We will not present a proof for this proposition since it is directly formulated from item (iv) in the minimal cover definition. Intuitively, this proposition enforces the minimality property of a base. Said in other words we must look for fd's ($Z \rightarrow A$) such that every concept in the left-end (i.e., in Z) is necessary (i.e., if we drop any concept, Z does not functionally determine A anymore). For example, according to the example introduced in Figure 4.9, if {`rentalDuration, beginningDate`} is known to be a base then, despite {`rental-Duration, beginningDate, money`} determines `endDurationPrice`, it is not a base, since it is not minimal.

Proposition 4. *Let* Z *and* W *be two sets of concepts functionally dependent on a given concept* A. *If* ZW *is a minimal key of* A *then,* Z *and* W *are orthogonal. That is, it does not exist two sets of concepts* Z_1 *and* W_1 *such that* $Z_1 \subseteq$ Z *and* $W_1 \subseteq$ W *and (*$Z_1 \rightarrow W_1$ *or* $W_1 \rightarrow Z_1$*).*

Proof. Let use W_2 to denote W - W_1 (respectively $Z_2 = Z$ - Z_1). If $WZ \rightarrow A$ and $Z_1 \rightarrow W_1$ (resp. $W_1 \rightarrow Z_1$) then $ZW_2 \rightarrow A$ (resp. $WZ_2 \rightarrow A$) holds (by the pseudo-transitivity rule). Thus, given a set of fd's F' such that $\{ZW \rightarrow A,\ ZW_2 \rightarrow A$ (resp. $WZ_2 \rightarrow A)\} \in F$', F' is not a minimal cover, since $Z_1 \rightarrow W_1$ (resp. $W_1 \rightarrow Z_1$) holds. We can show it by contradiction.

Let F' be a minimal cover of F and $\{ZW \rightarrow A,\ ZW_2 \rightarrow A$ (resp. $WZ_2 \rightarrow A)\} \in F$'. If $Z_1 \rightarrow W_1$ (resp. $W_1 \rightarrow Z_1$) holds, then, there is a proper subset F'' of F' (i.e., F'' = F' - $\{ZW \rightarrow A\}$) such that F'' implies F, which violates item (iii) in the minimal cover definition. Indeed, we can get $ZW \rightarrow A$ from $ZW_2 \rightarrow A$ (resp. $WZ_2 \rightarrow A$) by the augmentation rule (i.e., adding W_1 (resp. Z_1) to both sides), and then by the reflexivity and transitivity rules. □

Intuitively, this property says that the combination ZW must not be considered if Z and W are not orthogonal. Otherwise, this set of fd's will not be minimal as demanded in the minimal cover definition. In our example, it means that {`rentalDuration, minimumDuration`} must not be considered as a potential base, since `rentalDuration` → `minimumDuration`. Finally, we introduce a third necessary condition derived from the fd's theory:

Proposition 5. *Let* W *be a set of concepts functionally dependent on a concept* A*, and* C *a concept such that* C ∈ W*. Let* $\mathcal{I}$ *be the set of intermediate concepts giving rise to the many-to-one path between* A *and* C*. If* W *functionally determines* A*, for each* $\{C_i\} \in \mathcal{I}$*,* $(W - \{C\}) \cup \{C_i\}$*, namely the intermediate sets of* W*, functionally determines* A*.*

Proof. It can be seen by means of the pseudo-transitivity rule. Since each intermediate concept determines C then, if $W \rightarrow A$, for each $\{C_i\} \in \mathcal{I}$, $(W - \{C\}) \cup \{C_i\} \rightarrow A$. □

Intuitively, we are taking advantage of the pseudo-transitivity rule to foresee if a given combination can functionally determine A. For example, if {`endingDate, minimumDuration`} is known to be a base then, {`endingDate, rentalDuration`} must be a base as well, since `rentalDuration` → `minimumDuration`.

In our approach, we only generate and verify with data those combinations that fulfill the three necessary conditions introduced above. In short, let A be a concept and Z a set of concepts. We check whether $Z \rightarrow A$ if and only if $A \rightarrow Z$ (i.e., for generating Z we only consider combinations of concepts in the FD-tree of A). Furthermore, among all the potential combinations we may generate to verify $Z \rightarrow A$, we only look for those that would belong to the minimal cover of the fd's determining A, by applying Props. 3 and 4 (note that these propositions guarantee minimality; the first one regarding subsets, and the second one regarding fd's). We also use Prop. 5 for pruning based on the pseudo-transitivity axiom. If a certain combination does not guarantee any of these conditions we may foresee it will not be a base and thus, it does not have to be generated and verified. Only those combinations satisfying all the conditions are *feasible base* and have to be verified with data. In this way, we reduce drastically the searching space and the number of combinations to be tested against the database (i.e., minimizing the number of queries posed to the RDBMS).

4.5.1.2 Searching Space

Our searching space can be characterized as a directed graph like the one shown in the left-end of figure 4.10. Two combinations of dimensional concepts in the searching space can be related by two different kinds of edges:

Subset edges (SS-edges) link two nodes of size *i* and *i+1* (where *i* is an integer between 1 and P), such that the first is a subset of the second one. For example, the edge between {rentalDuration} and {rentalDuration, carGroup}, and we say that {rentalDuration, carGroup} is a *SS-descendant* of {rentalDuration} (denoted by {rentalDuration} $\subset$ {rentalDuration, carGroup}). Note that these arrows (in the figure, dashed arrows) link combinations in consecutive depth levels. Therefore, if a 3-sized combination (in the third depth level), such as {minimumDuration, rentalDuration, carGroup} were depicted in the figure, it would be related with a dashed arrow to {minimumDuration, maximumDuration}, {minimumDuration, carGroup}, and {maximumDuration, carGroup} but not to {minimumDuration}, {maximumDuration}, or {carGroup}, since they are not placed in consecutive depth levels. Note, however, that they are related by transitivity.

Functional Dependency edges (FD-edges) are derived from the input FD-tree. They link two nodes of the same size such that there is one concept in the first one functionally determining one concept in the second one. For example, {rentalDuration, carGroup} is related to {rentalDuration, carGroupName}, since carGroup $\rightarrow$ carGroupName (denoted by {rentalDuration, carGroup} $\prec$ {rentalDuration, carGroupName}). Note that these (regular) edges relate combinations where only one concept changes. For example, {rentalDuration, carGroup} is not related to {maximumDuration, carGroupName} despite carGroup $\rightarrow$ carGroupName and rentalDuration $\rightarrow$ maximumDuration, since we must substitute two concepts of {rentalDuration, carGroup} to obtain {maximumDuration, carGroupName}. Indeed, {rentalDuration, carGroup} is related to {maximumDuration, carGroupName} by pseudo-transitivity, since, following the nomenclature of Prop. 5, there are two intermediate sets (i.e., {rentalDuration, carGroupName} and {maximumDuration, carGroup}; see Figure 4.10) between them.

In Figure 4.10, we show a piece of the searching space for endDurationPrice. It is based on the FD-tree between its dimensional concepts. Note that, for the sake of simplicity, we use a partial FD-tree of only five dimensional concepts (right-end of figure 4.10). Moreover, we only draw the first two depth levels of the searching space (left-end of figure 4.10). Furthermore, talking about depth levels is meaningless in most graphs, but we can still talk about graph depth levels according to how we have defined the edges. On level *i* we depict those combinations of size *i*. Between consecutive depth levels we find SS-edges, and FD-edges between combinations in the same depth level. Moreover, SS and FD-edges introduce a *partial order* in the searching space (i.e., $\subset$ for subsets and $\prec$ for fd's). Finally, note that SS-edges will be those explored (and pruned if needed) by Prop. 4, whereas FD-edges will be explored (and pruned) by Props. 5 and 6.

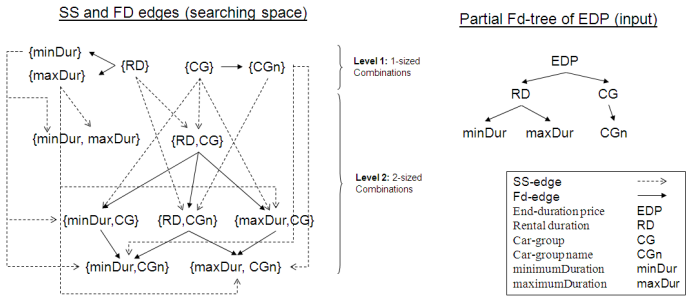


Figure 4.10: AMDO: the searching space for the `endDurationPrice` concept, and a piece of its FD-tree

4.5.2 An Algorithm for Discovering Bases

Our algorithm for discovering bases takes advantage of previous generated and tested combinations to decide which alternatives must be explored, according to the necessary conditions introduced in Section 4.5.1. This algorithm is devised according to three sets:

- *Feasible bases*: In the i^{th} iteration, this set represents those combinations of size *i*, satisfying the three necessary conditions introduced in Section 4.5.1.
- *Candidate sets*: In the i^{th} iteration, this set contains those *feasible bases* refuted as bases with data.
- *Bases*: In the i^{th} iteration, this set contains those *feasible bases*, up to size *i*, verified as bases with data.

Our algorithm has two inputs: the concept we are looking bases for (for example, the `endDurationPrice`), and its FD-tree (see Figure 4.9). The algorithm starts considering each root concept as a *feasible base* (see Figure 4.11, step 3). This is sound, since root concepts are *unary sets* fulfilling the necessary conditions by definition (and thus, we can directly consider them *feasible bases*). Every combination in the *feasible bases* is verified (see step 4ci and Section 4.5.2.1) to see if, according to data, it is indeed a base (if it is, this combination is added to the *base sets*; see step 4ciA) or, if it is not (then, it is added to the *candidate sets*; see step 4ciiA).

Note that we only explore the FD-descendants of a combination *Z* if *Z* determines *A* (see step 4ci and Section 4.5.2.1). It is a direct application of Prop. 5: the FD-descendants of a given combination *Z* cannot determine *A* if *Z* does not determine *A*. From here on, we denote by the *Intermediate Set Rule* (ISR) this application of Prop. 5 in our algorithm. A combination is generated by ISR, in the *gen_comb_by_FD* function, if its direct FD-ancestors are known to determine *A* (the reason why this function needs the *base sets* as parameter). Importantly, ISR only needs to check the direct FD-ancestors to decide if a given FD-descendant must be generated or not. This holds because FD-ancestors are generated either in the *gen_comb_by_FD* function (by ISR) or in *gen_comb_by_SS*, and both functions guarantee that new combinations generated

```
function seek_bases (Concept A, Fd-Tree M) returns Set<Base>

1. Set<Concept> Comb; Ordered Set<Comb> Candidates_Sets, Feasible_Bases;
2. int i:=1; Set<Comb> Bases := {};
3. Feasible_Bases := Get_Root_Concepts(A,M);
4. while(Feasible_Bases != ∅)
   (a) Candidates_Sets := {};
   (b) Comb := Get_First_Combination(Feasible_Bases);
   (c) while(Comb != ∅)
       i. if(Determines(Comb,A)) then
          A. Bases += Comb;
          B. if (Has_fd-Descendants(Comb,M)) then
             Feasible_Bases += Gen_Comb_by_FD(Comb, Bases, M);
       ii. else
          A. Candidates_Sets += Comb;
       iii. Feasible_Bases -= Comb;
       iv. Comb := Get_Next_Combination(Feasible_Bases);
   (d) i++;
   (e) Feasible_Bases := Gen_Comb_by_SS(i, Bases, Candidates_Sets, M);
5. return Bases;
```

Figure 4.11: AMDO: an algorithm for discovering bases

do fulfill the three necessary conditions (see Section 4.5.3 for further details on the former, and Section 4.5.2.2 for further details on the latter).

For example, if {`beginningDate, endingDate, rentalDuration`} is a base, then by means of ISR we must add {`beginningDate, endingDate, rentalDurationName`}, {`beginningDate, endingDate, minimumDuration`}, {`beginningDate, endingDate, maximumDuration`} and {`beginningDate, endingDate, timePeriod`} to the *feasible bases*. Given a verified base, the number of new combinations added to the *feasible bases* by applying ISR, is, in the worst case, linear regarding the number of direct FD-descendants the base has (see the properties of FD-edges in Section 4.5.1.2).

Since combinations generated fulfill the necessary conditions, we directly add them to the *feasible bases* set. Thus, each new combination is eventually verified as a base. If it is a base then, we iteratively apply ISR, and we continue exploring its FD-descendants. Interestingly, note that a ISR-generated combination, which is refuted as a base (i.e., the *determines* function returns false and therefore, it is queued in the *candidate sets*), may give rise to bases of bigger size than i when combined with other SS-descendants in step 4e. Following our example, suppose that {`beginningDate, endingDate, rentalDurationName`} happens to be a base, but not the rest of FD-descendants generated. In this case, according to ISR, this combination could generate new FD-descendants, which would be queued in the *feasible bases* set (however, this is not the case, as none of the concepts in this combinations have FD-descendants).

function *Gen_Comb_by_SS* (int i, **Set**<Comb> Bases, **Ordered Set**<Comb> Candidates_Sets, FdTree M) **returns** Set<Comb>

1. **Set**<Comb> Combinations := {};
2. **For**(**int** j = 0; j < **sizeof**(Candidates_Sets); j++)
   (a) CS1 := get_candidate_set(Candidates_Sets, j);
   (b) **For**(**int** z = j+1; z < **sizeof**(Candidates_Sets); z++)
      i. CS2 := get_candidate_set(Candidates_Sets, z);
      ii. **if** (Have_(i-1)_Concepts_In_Common(CS1,CS2) AND (i != 2 OR orthogonal(CS1,CS2)))
         A. **if**(Find_Subsets(CS1,CS2, Bases, Candidates_Sets, z)) **then**
         B. Combinations += Eliminate_Duplicates(CS1 ∪ CS2);
3. return Combinations;

Figure 4.12: AMDO: an algorithm to compute SS-descendants

Regarding the rest of combinations, they would be queued in the *candidate sets* and eventually, they could generate *(i+1)*-sized sets (for example, {`beginningDate, endingDate, minimumDuration, carGroup`} from {`beginningDate, endingDate, minimumDuration`}).

When the current algorithm iteration is done (i.e., all the *i*-sized combinations have been treated; see step 4c), function *gen_comb_by_SS* generates *feasible bases* of size *i+1* from the *i*-sized *candidate sets* (see step 4e and Section 4.5.2.2 for further details). The algorithm iterates until we are not able to generate *feasible bases* of size *i+1*.

4.5.2.1 The *determines* Function

This function is called when the three necessary conditions are guaranteed (i.e., we have identified a *feasible base*). Then, we verify if this combination determines *A* by querying data. Prior to query the instances, we first introduce a final pruning rule:

Proposition 6. *Let* Z *be a feasible base. We say that* Z *is yet a* feasible *base, if it is able to identify all instances of* A. *In other words, if the cardinality of* A *is lesser (or equal to) than the product of the cardinalities of the concepts in* Z *(i.e.,* $\prod_{Z_i \in Z} |Z_i| \geq |A|$*)*

Note that this pruning rule discards combinations by just querying the RDBMS catalog, as follows (expressed in Oracle syntax):

```
SELECT NUM_ROWS FROM USER_TABLES WHERE TABLE_NAME = t;
SELECT NUM_DISTINCT FROM USER_TABS_COLS WHERE TABLE_NAME = t AND COLUMN_NAME = c;
```

Where *t* is the name of a table and *c* of a column. If the ontology concept maps to a relational table then by means of the first query we get the cardinality of *t*, and if the ontology concept maps to a relational attribute by means of the second query we get the number of different values it has. Those combinations satisfying this rule are still candidates to be a base, and we verify it by the following query (in Oracle syntax):

function *Find_Subsets* (**Comb** CS1, **Comb** CS2, **Set**<Comb> Bases, **Ordered Set**<Comb> Candidates_Sets, int z) **returns** Boolean

1. **Set**<Comb> SubSets := Generate_Subsets(CS1, CS2);
2. **For**(**int** i = 0; i < **sizeof**(Bases); i++)
   (a) BaseAux := get_base(Bases, i);
   (b) **if**(BaseAux **in** SubSets) **then**
      i. **return false**;
3. **For**(**int** w = z+1; w < **sizeof**(Candidates_Sets); w++)
   (a) CSAux := get_candidate_set(Candidates_Sets, w);
   (b) **if**(CSAux **in** SubSets) **then**
      i. SubSets -= {CSAux};
4. **foreach**(SubSet **in** SubSets) **do**
   (a) **if**(all_root_concepts(Subset))
      i. **return false;**
5. **return true;**

Figure 4.13: AMDO: an algorithm for generating (i+1)-sized combinations

```
SELECT "base" FROM DUAL WHERE NOT EXISTS(SELECT attrSet FROM tables WHERE joinConds GROUP
BY
attrSet HAVING COUNT(*) > 1)
```

Where `DUAL` is the *dummy* table in Oracle and *attrSet* are the attributes forming the *feasible base* to be verified, *tables* the list of tables containing that attributes and *joinConds* the join clauses needed to join tables involved in the query. If we are able to find two rows with the same values for the base hypothesis then, this combination, according to data, is not a base. Notice that we use a `NOT EXISTS` expression so that if we find a counter example for this combination then, the RDMBS engine could stop the query.

4.5.2.2 Generating Combinations of Size (i+1): The *gen_comb_by_SS* Function

Once the *i*-sized combinations have been verified (i.e., either proved to be bases, and thus added to the *bases set*, or refuted, and thus, added to the *candidate sets*), the *gen_comb_by_SS* function (see Figure 4.12) generates (*i+1*)-sized combinations from the *i*-sized *candidate sets* obtained in the previous iteration of the algorithm. This function looks for pairs of sets having (*i*-1) concepts in common (see Figure 4.12, step 2bii). To do so, it scans the *candidate sets* in such an order that it does not generate twice the same pair (see the configuration of the two main loops; steps 2 and 2b). From every pair identified, it produces a *i+1*-sized combination (note that the combinations paired share *i-1* concepts and thus, produce a *(i+1)*-sized combination).

Importantly, this function only generates combinations fulfilling the three necessary conditions, as follows:

- Prop. 4 is guaranteed in step 2(b)ii. A 2-sized set is generated if concepts combined are orthogonal (i.e., if they are not present in the FD-tree of each other). As shown in Section 4.5.3, if 2-sized sets are orthogonal then, Props. 3 and 4 guarantee that no orthogonal sets of size greater than 2 are generated. Thus, it is enough to check this proposition for 2-sized sets. In our example, after the algorithm first iteration, the *gen_comb_by_SS* function is called for $i = 2$ (note that we first increase i and then, call this function; see step 4d). If the *candidate sets* contain {`rentalDuration`} and {`minimumDuration`} they would not be combined to form {`rentalDuration, minimumDuration`}, since since `rentalDuration` → `minimumDuration`.

- Prop. 3 is guaranteed in step 2(b)iiA by the *find_subsets* function (described in Figure 4.13). This function generates all the i-sized subsets of the current $(i+1)$-sized set treated (note that it can be done in linear time: for a $i+1$-sized combination, we must generate $i+1$ subsets overlooking each one of the concept in the $i+1$ combination), and verify them not to be bases as follows:

  - (1) If any of the subsets of the $(i+1)$-sized set is in the *base sets* then, it must not be considered a (i+1)-sized *feasible base* (see step 2), since it is not minimal.
  - (2) If all the i-sized subsets are in the *candidate sets* then, we guarantee that the $(i+1)$-sized set is minimal (see Figure 4.13, step 3) and thus, a feasible base.
  - (3) Alternatively, due to our pruning rules, it may happen that a subset is neither in the *candidate sets* nor in the *base sets*. In this case, we have two possible scenarios: on the one hand, (3.1) if the subset is only compound of root concepts, the (i+1)-sized combination must be refuted, since our algorithm is exhaustive regarding root concepts. Thus, we can assure that this subset is a SS-descendant of a base (see step 4a). On the other hand, (3.2) in any other case, this subset is a FD-descendant of a i-sized *feasible base* refuted with data (see step 5) and therefore, fulfilling the three necessary conditions. We justify this decision in Section 4.5.3.

- Finally, Prop. 5 is guaranteed by the *candidate sets* definition, since they have been refuted as bases.

For example, suppose that the *gen_comb_by_SS* function combines {`beginningDate, minimumDuration`} and {`beginningDate, money`} to produce {`beginningDate, minimumDuration, money`}. This scenario could happen in the second iteration of the algorithm, when $i = 3$, and we combine 2-sized sets having in common one concept to eventually produce 3-sized sets. Since i != 2, Prop. 4 does not apply, and we just need to focus on function *find_subsets*, which generates the 2-sized subsets of {`beginningDate, minimumDuration, money`}: {`beginningDate, minimumDuration`}, {`minimumDuration, money`} and {`beginningDate, money`}. According to (1), if any of them is a base then, the 3-sized set is not generated (since it is not minimal). Oppositely, according to (2), if all of them are in the *candidate sets*, we can guarantee that the 3-sized set is minimal and it is generated. Consider now that, according to (3), the {`beginningDate, money`} and {`beginningDate, minimumDuration`} subsets are neither in the *candidate sets* nor in

the *base sets*. In the first case, since it is compound of root concepts, it denotes that the 3-sized set is not minimal (since our algorithm is exhaustive for root concepts and thus, it means that either `beginningDate` or `money` is a base). In the second case, since {`beginningDate, minimumDuration`} is a FD-descendant of {`beginningDate, rentalDuration`} it does not invalidate the 3-sized set. The reason is that {`beginningDate, minimumDuration`} has been compulsory generated by the *gen_comb_by_FD* function (had it been generated by the *gen_comb_by_SS* function, this subset would have been included in the *base sets* or *candidate sets* and thus, considered by either (1) or (2)). Thus, it means that {`beginningDate, rentalDuration`} is a *feasible base* refuted with data. Consequently, since it was refuted as a base, according to Prop. 5, we could foresee that {`beginningDate, minimumDuration`} would not be a base and then, it was not even generated. However, this set fulfills the three necessary conditions and therefore, it does not invalidate the 3-sized set.

4.5.3 Algorithm Correctness

Our algorithm is sound and complete with regard to the domain ontology and data, as it generates sets from knowledge captured in the ontology and verifies them with the data. In this section we show that we generate all the sets of the searching space that fulfill the three necessary conditions, and that we do not generate any set that do not fulfill them. Thus, our proofs of soundness and completeness rely on the three necessary conditions discussed and justified in Section 4.5.1. Note that, without considering our prune rules, our algorithm would generate all the possible combinations of the searching space. For this reason, we just need to show that our pruning rules do not miss any base nor generate combinations not being base. In other words, in our algorithm we generate all the combinations of the searching space that (i) all its subsets are not bases (i.e., all its SS-ancestors are not bases), (ii) its subsets are orthogonal (i.e., we cannot find two SS-ancestors such that one is a FD-descendant of the other) and (iii) its *intermediate sets* are, indeed, bases (i.e., all its FD-ancestors are bases). If we are able to generate all the sets meeting the necessary conditions and no other sets, it is easy to see that by verifying them with data (see Section 4.5.2.1) our algorithm is still sound and complete.

For the sake of understandability, we divide our proof in two parts. The first part shows that our algorithm is sound and complete for *root combinations* (i.e., those only consisting of root concepts). Then, we show that it is also sound and complete for *FD-combinations* (i.e., those that contain, at least one FD-concept).

4.5.3.1 Soundness & Completeness for Root Combinations

In this section we show that we generate all the root combinations satisfying the three necessary conditions (completeness) and no other combination (soundness). We show it by induction on the size of the sets:

- **First Iteration:**

  `completeness` In the first iteration of the algorithm, every root concept is considered a *feasible base*, as they are the smaller set to be verified as a base (see Figure 4.11, step 3).

`soundness` Since all of them are 1-sized sets, they trivially satisfy Props. 3, 4.

- **Second Iteration:**

  `soundness` 2-sized root combinations are generated by combining all the 1-sized *candidate sets* found in the previous iteration. By definition, those 1-sized sets in the *candidate sets* have been refuted as bases (i.e., fulfilling Prop. 3), and only those 2-sized sets consisting of orthogonal concepts are generated (i.e., fulfilling Prop. 4. See Figure 4.12, step 2(b)ii).

  `completeness` We generate all the 2-sized combinations between root concepts except for (i) those consisting of a root concept being a base, and (ii) those consisting of two root concepts such that one is a FD-descendant of the other.

- **Induction Step:**

  If we generate all the n-sized root combinations fulfilling the three necessary conditions and only these combinations (induction hypothesis), we will generate all the (n+1)-sized root combinations that fulfill the three necessary conditions and no other sets.

  `soundness` If all the n-sized *feasible bases* are generated, those verified as bases were added to the *base sets* and those refuted as bases to the *candidate sets*. According to Figure 4.13 (steps 3 and 4), Prop. 3 is fulfilled because root combinations are generated if all their subsets are in the *candidate sets* (according to the induction hypothesis all the n-sized *candidate sets* are generated) and discarded otherwise. Prop. 4 is guaranteed because 2-sized sets consisting of non-orthogonal concepts were not generated. It can be shown by contradiction. Supposed that a (n+1)-sized set Z, that contains two concepts A and B such that $A \rightarrow B$ is generated by our algorithm. Thus, all the n-sized subsets of Z are in the n-sized *candidate sets* (i.e., there is at least one n-sized set W such that $W \subset Z$ and A, $B \subset W$). But this contradicts the induction hypothesis, as W is not a n-sized *feasible base* and it is not present in the n-sized *candidate sets*. Intuitively, since $\{A, B\}$ was not generated, all its SS-descendants were not generated either (i.e., $\{A, B\}$ will not be in the 2-sized *candidate sets* and according to Prop. 3, every 3-sized set containing it will not be generated, neither those 4-sized sets containing the 3-sized sets and so on up to W that will not be generated and thus neither Z).

  `completeness` All the (n+1)-sized root combinations are generated except for those that, at least, one of their n-sized subsets is not in the *candidate sets*. It may happen because (i) this subset is a base (all the n-sized bases are found by hypothesis) or (ii) it contains a subset that is a base (i.e., it is not a n-sized *feasible base*) and thus not generated by hypothesis.

Finally, note that we have not justified Prop. 5. This is because root combinations do not have intermediate sets by definition. □

4.5.3.2 Soundness & Completeness for FD-combinations

In this section we show that we generate all the FD-combinations satisfying the three necessary conditions (completeness) and no other combination (soundness). Like in previous section, we show it by induction:

- **First Iteration:**

  `soundness` 1-sized FD-combinations are generated applying ISR to combinations verified as bases and thus, fulfilling Prop. 5. This new generated combinations are of size 1 and they also fulfill Props. 3 and 4.

  `completeness` All the 1-sized FD-combinations are generated because ISR is iteratively applied (i.e., it follows the FD-edges until it finds FD-descendants that are not a base). Therefore, those FD-descendants not explored will not be a *feasible base* since they do not fulfill Prop. 5.

- **Second Iteration:**

  `soundness` 2-sized FD-combinations can be generated from the *gen_comb_by_SS* function or by applying ISR over 2-sized combinations verified as bases. In the first case, they fulfill the necessary conditions by the same proof introduced for 2-sized root combinations. In the second case, ISR guarantees Prop. 5, as explained in the first iteration. Furthermore, combinations generated by ISR also guarantee Props. 3 and 4. Let W, Z be two 2-sized sets of concepts such that $W \prec Z$, and Z was generated by ISR. By definition of the FD-edges (see Section 4.5.1.2) there is a concept B such that $B \subset Z$ and $B \subset W$, and two concepts A, A_1 such that $A \rightarrow A_1$, and $A \subset W$, and $A_1 \subset Z$. Z satisfies Prop.3 because if W is generated, B is not a base (since it is in the *candidate sets*), and if A_1 is a base, then A is a base as well. Thus, A would have been verified as a base in the first iteration, and, by ISR, A_1 as well (so that, A_1 would be in the *base sets* and Z would not be generated). Prop. 4 is guaranteed because 2-sized sets are generated if concepts on it are orthogonal (see Figure 4.12; step 2bii).

  `completeness` We show that our algorithm finds all the 2-sized FD-combinations by contradiction. Let Z be a 2-sized FD-combination not found by our algorithm. If we have not been able to generate Z, at least one of its concepts (i.e., A_1) must not be in the 1-sized *candidate sets*. Therefore, A_1 is a FD-concept (as every root concept will be present in the *candidate sets* or in the *base sets*; see second iteration in previous section). By definition of our searching space, there exists a 2-sized combination W such that W is an intermediate set of Z and it consists of root concepts (see Section 4.5.1.2). In other words, $W \prec Z$ by pseudo-transitivity. Thus, by Prop. 5, if Z is a base then W is also a base. Since our algorithm is complete for root combinations, W would have been generated, and Z as well by ISR.

- **Induction Step:** If we generate all the n-sized FD-combinations fulfilling the three necessary conditions and only these combinations (induction hypotheses), we will generate all the (n+1)-sized FD-combinations that fulfill the three necessary conditions and no other sets.

`soundness` If all the *n*-sized *feasible bases* are generated, those verified as bases were added to the *base sets* and those refuted as bases to the *candidate sets*. (*N*+1)-sized FD-combinations can be generated by applying ISR over (*n*+1)-sized combinations verified as bases, or from the *gen_comb_by_SS* function. In the first case, we can show that the necessary conditions are preserved by a proof analogous to the one introduced for 2-sized sets generated by ISR. In the second case, Prop. 3 is guaranteed because the subsets of the (*n*+1)-sized *feasible bases* are minimal (see Figure 4.13; steps 3 and 5): either we can find them on the *n*-sized *candidate sets* or they are neither in the *candidate sets* nor in the *base sets*. The latter raises when two *n*-sized combinations (where n $\geq$ 2) generated by ISR have been refuted as bases (thus, added to the *candidate sets*) and combined for generating a (*n*+1)-sized set. Let *W* and *Z* be the two *n*-sized sets generated by ISR from *W'* and *Z'* (i.e., *W'* $\prec$ *W* and *Z'* $\prec$ *Z*). Let *A*, *B*, A_1 and B_1 concepts such that $A \rightarrow A_1$, $B \rightarrow B_1$, $A \subset W$, $A_1 \subset W'$, $B \subset Z$ and $B_1 \subset Z'$. If *W*, *Z* can be combined (i.e., $Y = W \cup Z$), they share *n*-1 concepts in common (i.e., there exists a subset *X* s.t. $W = X \cup A_1$, $Z = X \cup B_1$ and $Y = X \cup B_1 \cup A_1$), and the *n*-sized subsets of *Y* where B_1 and A_1 are combined will be missing. These sets, however, can be shown to be minimal because they are of the kind $X' \cup B_1 \cup A_1$, (where $X' \subset X$). By hypothesis, *X* is not a base (otherwise, *W* and *Z* would have not been minimal sets). Prop. 5 is guaranteed because the FD-ancestors of *Y* are of the kind: $X \cup A \cup B_1$ (i.e., $W \cup B_1$) and $X \cup A_1 \cup B$ (i.e., $Z \cup A_1$). Since *W* and *Z* are bases, they fulfill Prop. 5 (note that, in this case, the FD-ancestors are not minimal, but it is sound since they are not generated as *feasible bases* in our algorithm). Finally, Prop. 4 can be shown by contradiction. Let *X* be a (*n*+1)-sized combination containing two concepts *A*, A_1 such that $A \rightarrow A_1$ and $Y = X - \{A\} - \{A_1\}$. Thus, either *A* is a base (and then A_1 was generated by ISR) or $A \cup Y'$ (where $Y' \subset Y$) is a base (and then $A_1 \cup Y'$ was generated by ISR). Thus, the set being a base would be missing and according to Prop. 3, *X* would have not been generated (since it is not minimal).

`completeness` We can easily show that our algorithm is complete for (*n*+1)-sized FD-combinations by means of the completeness proof for root-concepts. If our algorithm is complete for(*n*+1)-sized root-combinations then, ISR guarantees that it is also exhaustive for (*n*+1)-sized FD-combinations. □

4.5.4 Discussion

In this section we introduce results obtained after carrying out our algorithm over the EU-Car Rental case study (see appendix B). This ontology has 65 concepts and 170 relationships. For each concept, we computed its FD-tree (see Section 4.4) and later, by means of the algorithm introduced in section 4.5.2, its *feasible bases*. It is important to remark that, in this simulation, we have not verified the bases with data. The reason is that the EU-Car Rental case study just provides a conceptualization of the domain, but not sample data. Nevertheless, we simulated the cardinalities we could expect for each ontology concept and consequently, the *determines* function can apply Prop. 6 (note that we have simulated the catalog by providing each concept

cardinality), but it does not query the sources.

As result, each concept has an average of 31'83 concepts in the FD-tree (i.e., in our algorithm we have an average searching space of $\sim 2^{32}$ combinations). About the FD-tree generated, an average of 6'67 are root concepts (i.e., the average value of combinations we start with, in the first iteration of our algorithm). After launching our algorithm, we obtained an average of 156'53 *feasible bases* per concept (i.e., we will query the database $\sim$157 times, in average; in front of the 3.817.550.246 times, if we would have generated all the combinations in the searching space (i.e., a naive approach). In general, considering all the queries posed to the database for all the concepts, we have a total of 10.018 queries, of which a 15% are answered by querying the catalog (see section 4.5.2.1) and the rest, by querying data (note that, as previously explained, in our simulation we did not queried data). Importantly, this is the minimum number of queries we must pose in order to find all the bases of the domain (see section 3.3.1). Furthermore, note that about the 30% of *n*-sized *feasible bases* generated, are generated by combining (*n*-1)-sized *candidate sets* whereas the rest are generated by ISR. The execution time of our algorithm is insignificant in front of the cost of querying data. Indeed, all the *feasible bases* for all the concepts were generated in less than 10 seconds[11] and in general, our algorithm behaves much better than traditional approaches.

For example, consider again the `endDurationPrice` concept discussed in this section. This concept produces 29 *feasible bases*, verified up to Prop. 6, with the catalog we simulated:

```
{EndingTime, BeginningDate}
{EndingTime, Money}
{BeginningDate, Money}
{Money, RentalDuration}
{Money, RD-MaximumDuration}
{Money, RD-MinimumDuration}
{Money, RD-Name}
{EndingTime, CarGroup, RentalDuration}
{BeginningDate, CarGroup, RentalDuration}
{Money, CarGroup, RD-TimeUnit}
{EndingTime, CG-Name, RentalDuration}
{EndingTime, CarGroup, RD-Name}
{BeginningDate, CG-Name, RentalDuration}
{BeginningDate, CarGroup, RD-Name}
{Money, CG-Name, RD-TimeUnit}
{EndingTime, CG-Name, RD-Name}
{BeginningDate, CG-Name, RD-Name}
{EndingTime, CarGroup, RD-MaximumDuration, RD-MinimumDuration}
{EndingTime, CarGroup, RD-MaximumDuration, RD-TimeUnit}
{EndingTime, CarGroup, RD-MinimumDuration, RD-TimeUnit}
{BeginningDate, CarGroup, RD-MaximumDuration, RD-MinimumDuration}
{BeginningDate, CarGroup, RD-MaximumDuration, RD-TimeUnit}
```

[11] The computer used in this test was equipped with an Intel Core 2 Duo 1.33 GHz processor, 1'99 GB of RAM.

{BeginningDate, CarGroup, RD-MinimumDuration, RD-TimeUnit}
{EndingTime, CG-Name, RD-MaximumDuration, RD-MinimumDuration}
{EndingTime, CG-Name, RD-MaximumDuration, RD-TimeUnit}
{EndingTime, CG-Name, RD-MinimumDuration, RD-TimeUnit}
{BeginningDate, CG-Name, RD-MaximumDuration, RD-MinimumDuration}
{BeginningDate, CG-Name, RD-MaximumDuration, RD-TimeUnit}
{BeginningDate, CG-Name, RD-MinimumDuration, RD-TimeUnit}

In an arbitrary database, it seems clear that most of these proposals would be discarded when querying data. For example, given that most rental agreements span over a fixed number of days, and that many customers tend to benefit from predefined offers (such as: rent a familiar car for two days and save up to 20%), {Money, RentalDuration} and the likes do not seem good candidates. Oppositely, proposals based on dates seem rather accurate. For example, {EndingTime, BeginningDate} or {BeginningDate, Money} could be interesting bases, depending on how the dates were stored. Interestingly, had we queried data, the refuted combinations would have been placed in the *candidate sets* and therefore, they could have produced bases of bigger size.

As a remark, alternative key concepts (such as rentalDuration and rentalDuration-Name for a rentalDuration, or carGroup and carGroupName for a carGroup) produce exactly the same bases (e.g., {Money, RentalDuration} and {Money, RD-Name}), and it would be up to the user to decide which is the most interesting for him / her. Indeed, the user selects those data cubes of interest, by selecting which bases are relevant for him / her. For example, the endDurationPrice concept has 10 dimensional concepts (see Figure 4.9). However, if for this fact we are only interested in the {EndingTime, BeginningDate} base, AMDO discards the rest of dimensional concepts and produce a conceptual schema with just these two dimensions. Later, as explained in Section 4.3.5, AMDO proposes aggregation hierarchies for each dimension identified. Relevantly, this is the way AMDO filters dimensional concepts, not to disturb the end-user with too many results and, unlike current supply-driven approaches, filter the results proposed by means of the base concept.

4.6 A Practical Case: The TPC-H

The TPC-H [Tra09] is a decision support benchmark, which have been used in Chapter 3 to exemplify how our first approach, i.e., MDBE, works. Although in this section we just focus on how AMDO applies for TPC-H, our final objective is clear; as shown in Chapter 5, these studies over the TPC-H lead to a comprehensive framework for analyzing and discussing the differences between both approaches. Thus, we adapt the TPC-H schema to AMDO. This benchmark introduces a (logical) relational schema and, in principle, it would not fit the assumptions of AMDO, which works at the conceptual level. However, the TPC-H relational schema introduced is well-formed and, for this reason, it makes sense to derive an OWL DL ontology from it, by means of reengineering[12].

[12]The OWL DL ontology derived from the TPC-H relational schema can be downloaded from www.lsi.upc.edu/∼oromero/TPC-H.owl

Concept	#Dimensional Concepts	#Potential Measures	FactEstimation
Lineitem	77	8	93
Orders	27	13	53
Customer	17	17	51
Partsupp	32	4	40
Supplier	16	8	32
Part	9	8	25
Nation	8	4	12
Region	3	4	11

Table 4.2: AMDO: ranked facts proposed for the TPC-H case study

Analogous to the MDBE analysis over the TPC-H case study, we focus on five interesting aspects: the specificity of requirements needed, the expressiveness and quality required in the sources, the degree of automation achieved, the computational complexity of the algorithm and the quality of results obtained (i.e., the output correctness and the extra knowledge obtained in the output thanks to the novel contributions of AMDO). Note that our study is exhaustive regarding the four axis discussed in Section 1.7, and we also provide a study of the performance (and thus, feasibility, of our proposal).

4.6.1 Requirements Specificity

AMDO is a sequential hybrid approach for deriving the multidimensional schema and therefore, it considers both requirements and data sources as first-class citizens. In this approach, the analysis of the data sources leads the process. Thus, it means that the process starts without considering the end-user requirements (in the TPC-H case study, represented by the 22 business queries introduced in the benchmark). After a full supply-driven approach, AMDO filters the results obtained according to objective evidences, and presents them to the user. In this moment, the user states his / her requirements by selecting the multidimensional concepts of interest. Specifically, AMDO starts by analyzing potential facts (see Section 4.3.3). In the supply-driven stage for the TPC-H case study, AMDO identifies 8 possible facts (we address the reader to Figure 3.2 in page 57 to recall the TPC-H relational schema): `lineitem`, `orders`, `partsupp`, `part`, `customer`, `supplier`, `nation` and `region`.

We estimate a concept to be a fact by means of the `factEstimation` quality function (see Section 4.3.3 for further details), which is defined as:

$$\texttt{FactEstimation}(\text{M, DC}) := M * 2 + DC$$

where M is the number of potential measures, and DC the number of potential dimensional concepts of a given ontology concept. Since we consider the TPC-H business queries as the end-user requirements, at this point, we should expect that the end-user chose `lineitem`, `orders` and `supplier` as facts of interest. In other words, the end-user would choose 3 out of the 4 better ranked fact candidates proposed by AMDO.

Next step in AMDO produces bases of interest for each fact identified. Like in the EU-Car Rental case study (see Section 4.5.4), in this simulation, we have not verified the base hypotheses generated with data, since the TPC-H case study does not provide sample data. Although the TPC-H benchmark provides rules for populating an arbitrary TPC-H relational database, these rules are oriented to estimate each concept cardinality, but not to simulate a real set of data (i.e., with a set of *implicit constraints or business rules*, which held in data and could be exploited by the base algorithm). For example, consider the p_mfgr attribute. According to the TPC-H definition, it must be populated with text appended with a digit, like in the [*"Manufacturer",M*] pattern, where *M* is a random value in [1,5]. Similarly, p_brand must be populated with text appended with a digit, like the [*"Brand",MN*] pattern, where *N* is a random value in [1,5], and *M* is defined while generating p_mfgr[13]. Clearly, this kind of rules are not enough for our purpose. On the contrary, constraints of the kind *a* customer *cannot place two* orders *during the same day*, or *a given* supplier *only supplies* customers *of his own* country, would be of our interest. Roughly speaking, by means of populating rules such as the ones introduced in the TPC-H benchmark, we mostly generate a random set of values, which will not capture the potential business rules of our domain. Indeed, meaningful candidate keys are hardly generated *by chance*, but they are a consequence of implicit or explicit business rules that hold in our domain and, consequently, in data.

Nevertheless, since we can estimate each attribute cardinality by means of the populating rules introduced, we have simulated the catalog of an arbitrary TPC-H database. Thus, the *determines* function does not verify the hypotheses with data, but can apply Prop. 6. Considering the orders fact, AMDO generates 289 *feasible bases* for it (the biggest ones compound of 4 concepts). Then, among the bases proposed, the user should choose those of his / her interest, but browsing 289 bases could be a tough task. Importantly, note that in this simulation we are just considering hypotheses and not verified bases. Indeed, some of the hypotheses generated (intuitively, most of them, since they are mostly of 2 or 3-sized combinations), would be rejected by the *determines* function if they were verified with data. In this case, they could have given rise to other combinations of bigger size (see Prop. 3). Having a look to the semantics involved in the orders table (see Figure 3.2 in page 57), it is rather intuitive to see that most (if not all) of the 2 or 3-sized hypotheses will be refuted as bases (for example, {o_custkey, o_comment}, {o_custkey, o_orderstatus, o_shippriority} or {o_comment, o_orderpriority, o_clerk}). Oppositely, we may just expect a bunch of verified bases compound of, at least, 5 or 6 attributes (note that, by virtue of Prop. 5, if an intermediate set is not a potential base then, its FD-descendants are not considered and thus, until a combination of orders attributes are not verified as a base, orders FD-descendants -i.e., customer, nation and region- will not be considered and combined). For this reason, in a execution of our algorithm over real-world data, our algorithm would produce a significantly smaller set of bases among which the user must choose those of his / her interest. .

In the last step, for every concept involved in a selected base, we shape its own dimension hierarchy by means of part-whole relationships. Note that this step takes advantage of the functional dependencies computed in the first step and, for example, the o_custkey concept (one of the dimensional concepts that we could expect to appear in at least, one selected orders base)

[13] We address the reader to [Tra09] for further details.

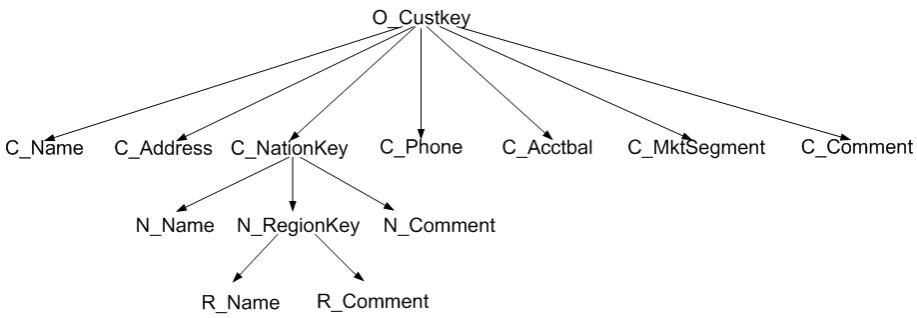


Figure 4.14: AMDO: an exemplification of of a directed graph

would produce the graph depicted in Figure 4.14. At this point, it is up to the end-user to tune-up the hierarchy according to his / her necessities. For example, according to the TPC-H business queries, the `clerk` attribute from `orders`, and `n_comment` and `r_comment` from `nation` and `region` should be discarded, since they are not used in any of the TPC-H queries.

All in all, AMDO minimizes the user interaction, and we just demand his / her participation to select the multidimensional concepts of interest among those not filtered by AMDO. Interestingly, AMDO does not generate large chunks of results, but filters them according to objective evidences. Furthermore, AMDO can provide additional knowledge to the user when selecting knowledge of his / her interest. For example, for every fact we can show the dimensional concepts and measures identified, or even pre-calculate bases or dimension hierarchies to facilitate his / her decision.

4.6.2 Data Source Expressiveness

AMDO works at the conceptual level and therefore, it requires semantically rich inputs. Indeed, the supply-driven stage performed in AMDO relies on the quality of the domain picture captured in the input ontology. Specifically, AMDO asks for:

- A ontology language expressive enough to capture all the domain assertions of interest. The expressivity needed may vary depending on the organization domain, since some conceptualizations require more expressive ontology languages to capture their concepts and relationships. In any case, assertions regarding the relationship multiplicities are crucial for computing AMDO multidimensional patterns. In our example, although the TPC-H conceptualization can be captured using *DL-Lite*$_{\mathcal{A}}$ (and therefore, benefiting from our generic reasoning algorithm for computing functional dependencies -see Section 4.4.3-), we have created an ontology in OWL DL notation. The reason is that AMDO disposes of a tool to automatically generate OWL DL ontologies, which facilitated this pre-process.

- For computing bases, AMDO requires a mapping among the ontology concepts and the data sources. For instance, preserving the concept names in the implementation would be

enough. Anyway, if this information is not available, we can infer this mapping through any of the approaches presented in the literature for automatic schema matching. As presented in [RB01], there are plenty of possibilities to choose from. For example, linguistic approaches such as name matching, may be used as a pre-process in our method. In systems following the Ontology-Based Data Access (OBDA) paradigm [PLC+08] or from the Semantic Web area, this mapping is supposed to exist. For the TPC-H case study, we have developed our own OWL DL ontology from the relational schema provided in [Tra09]. Therefore, this mapping is available since we preserved the relational names.

- Tightly linked to the previous item, we also need to access the data sources to eventually verify the base hypotheses generated by AMDO with data. In the TPC-H case study, sample data with which to verify the feasible bases proposed is missing. To overcome partially this drawback, we have simulated a database catalog by following the TPC-H populating constraints.

4.6.3 Automation

AMDO's automation degree is rather high. Indeed, the end-user is only involved in the design task to state his / her multidimensional requirements:

- After the first stage, AMDO ranks ontology concepts likely to play a factual role, according to a quality function. It is up to the user to choose the *factEstimation* function provided or introduce an ad hoc formula. Next, the user must set a threshold upon which the filtering task is based on. Finally, he / she must select the facts of his / her interest. In the TPC-H case study, if we had set the *factEstimation* threshold at 30, AMDO would have proposed 5 potential facts (see Table 4.2). Eventually, the user would choose the three facts of his / her interest, as discussed in Section 4.6.1.

- At the end of the second step, verified bases would be ranked according to their sparsity level and then, presented to the user. Thus, the end-user is responsible for modifying the threshold provided up to his / her needs. In the TPC-H case study, this threshold does not apply, since we have not been able to verify base hypotheses with data. Nevertheless, according to the discussion introduced in Section 4.6.1 and considering the TPC-H queries, the end-user would not need to choose more than a ten of bases for the three facts of interest.

- Finally, the third step generates the dimension hierarchies, which are presented to the user and eventually, tuned-up to meet the end-user requirements. In the TPC-H simulation, the user just needs to discard those attributes in the directed graphs that are not used in any of the queries. Specifically, 21 out of the 61 attributes in the ontology should be discarded.

Summing up, AMDO only asks for two input parameters: the quality functions (if we want to modify them) and the thresholds. Besides this, it does not require any other interaction from the user but to state the requirements, which are demanded in a guided and comprehensive way.

4.6.4 Computational Complexity & Performance

We discuss AMDO's feasibility by analyzing the computational complexity of the underlying algorithms. In this section we first discuss of the algorithm for discovering functional dependencies (used to compute the dimensional concepts, measures and dimension hierarchies), and later, we discuss of the algorithm for computing bases. The execution times presented in this section refer to a regular desktop computer[14]:

Computing Functional Dependencies: Since we generated an OWL DL ontology for the TPC-H case study, we computed the functional dependencies by means of our specific reasoning algorithm (see Section 4.4.2). As shown in Table 4.2, AMDO discovers, in overall, 189 dimensional concepts (the sum of the second column in the table) and 66 measures (the sum of the third column). The time needed to compute them is negligible: i.e., less than a second.

Computing Bases: The TPC-H case study identifies three facts: `lineitem`, `orders` and `partsupp`. First, note that the results shown in this section refer to *feasible bases* and thus, to base hypotheses in the minimal cover not yet verified with data. In overall, AMDO discovers 289 feasible bases for `orders`, 728 for `partsupp` and 1026 for `lineitem`, and the time needed to compute them was 15 seconds.

4.6.5 Output Quality

In this section we measure the quality of the output produced by AMDO. We do so by means of the result correctness (by comparing the output obtained with the multidimensional schema proposed in the Star Schema Benchmark), and the additional output inferred regarding both, the Star Schema Benchmark and previous approaches.

4.6.5.1 Output Correctness

We may derive the Star Schema Benchmark from the AMDO output by producing the same conceptual schema as MDBE and then, generating a logical star schema by means of the design decisions discussed in Section 3.5.5.1. To produce the same conceptual schema as MDBE, we just need to select, among results provided by AMDO, those producing the schema depicted in Figure 3.3 (see page 3.3). It can be achieved as follows:

- At the end of the first stage, choosing `lineitem`, `orders` and `partsupp` as our facts of interest. Consider the results provided in Table 4.2. Clearly, `lineitem` is the most interesting event to analyze in the TPC-H schema and, consequently, it is ranked in first position. Objectively, it provides 77 dimensional concepts from which a subject of analysis, containing 8 different measures, can be analyzed from. Similarly, `orders` and `partsupp` provide plenty of multidimensional knowledge, and it is rather logical that they were ranked in the first positions. However, it is surprising the position of `customer`. Objectively, it provides 17 dimensional concepts and 17 different measures

[14]The computer used in this test was equipped with an Intel Core 2 Duo 1.33 GHz processor and 1'99 GB of RAM.

(most of them aggregate measures; see Section 4.3.2 for further details). Indeed, it provides as many measures as dimensional concepts. On the one hand, this is interesting, since we have plenty of relevant indicators to analyze customers from. On the other hand, the amount of dimensional concepts is not that high, and the dimension hierarchies obtained span over three deep levels (`customer`, `nation` and `region`) at most. To some degree, `supplier` and `part` portray a similar scenario, whereas `nation` and `region` are clearly of no interest.

- At the end of the second stage, choosing the bases forming the multidimensional spaces required for each TPC-H query. For example, considering the TPC-H business queries, we must select the minimal set of bases containing all the dimensional concepts of interest; i.e., `o_custkey, o_orderstatus, o_totalprice, o_orderdate, o_shippriority` and `o_comment`. Importantly, note that bases produced by AMDO are *atomic bases* regarding the data sources. Thus, *aggregated bases* (i.e., those implying aggregation of data to be considered a base) at coarser level are not computed by AMDO, since they are not keys at the data granularity level provided by the relational sources. For example, according to the TPC-H queries, the `n_nationkey` attribute by itself is an interesting base for `orders`. However, it will only be a base by aggregating data. To consider it, we should choose any atomic base involving `o_custkey`, from which we would be able to derive, by means of aggregations, this aggregated base.

- Finally, among the graphs produced for each dimensional concept involved in a selected base, we must choose the attributes considered in any of the queries. Specifically, we must discard 21 out of the 61 depicted in the TPC-H schema. For example, the `p_mfgr` and `r_comment` attributes are not demanded by any of the TPC-H queries.

4.6.5.2 Additional Output Inferred

In this section we measure the impact of the main contributions of AMDO on the output. AMDO leads the whole task from a thorough analysis of the data sources. Consequently, its main contribution is its capability to derive objective analysis evidences that, at first sight, could not be immediate to identify. This feature can be exploited to derive a whole multidimensional schema without providing, beforehand, the end-user requirements. Specifically, this feature provides a novel contribution regarding bases. Consider again the `orders` fact. As discussed in Section 4.6.4, AMDO generates 289 *feasible bases*. For our current purpose, let us suppose that they are, indeed, verified bases. Thus, they would be ranked and presented to the user, according to its sparsity level. Having a look to the bases proposed, we note that some of them are rather interesting, despite they are not immediate to see, and likely to be verified with data. Indeed, this is the novel contribution of discovering bases: on the one hand, they help to filter dimensional concepts of no interest; on the other hand, they provide an insightful evidence to decide what can be interesting for us. For example, {`o_orderdate, o_shippriority, c_mktsegment, n_nationkey`}, which analyzes `orders` from the perspective of the *ordering data*, the *shipment priority* chosen, the *market segment* the customer belongs to, and the *customer nationality*. Indeed, note that this base is similar to {`o_custkey, o_orderdate`} but, interestingly, it suggests to identify the `customer` by the `segment market`, the `shipment`

priority required and his / her nation, and it is more likely to be verified with data than {o_custkey, o_orderdate}. A similar case would be {o_orderstatus, o_order-priority, c_address, c_comment}, which analyzes a shipment from the perspective of its *current status* (e.g., delivered, lost, on the way, etc.), its *order priority* (e.g., urgent, high, medium, low, etc.), the *address* it was sent to and *comments* regarding the whole process. Thus, bases provide plenty of additional knowledge for a supply-driven stage, which AMDO exploits to filter the dimensional concepts and facilitate the user task of selecting analysis perspectives of his / her interest. Similarly, last step shapes dimension hierarchies from directed graphs. Again, the user may identify relevant data granularity levels or descriptors of interest from the output proposed by AMDO and eventually, discover unknown analysis perspectives.

Finally, we would like to remark three other relevant features regarding AMDO outputs:

- AMDO is able to derive meaningful aggregation paths to shape the dimension hierarchies. In the TPC-H case study, this feature does not make the difference, since the relational schema is well-formed. But, in the general case, AMDO produces hierarchies exploiting all the conceptual knowledge captured in the input ontology, which is semantically richer than the domain knowledge captured at the logical level.

- The multidimensional patterns introduced for discovering measures consider the novel concept of *aggregated measures*. For example, in the TPC-H case study, by means of Patterns 2 and 2.a introduced in Section 4.3.2, AMDO discovers up to 15 aggregated measures for the customer concept. Note that orders, lineitem, partsupp and part are considered *bridge-classes* for customer, when computing the above mentioned patterns.

- AMDO consider the path semantics when discovering dimensional concepts and measures. Thus, two concepts related by means of *n* different paths satisfying the multidimensional patterns would produce, respectively, *n* different dimensional concept or measures. For example, in the TPC-H case study, this is the case of nation and region regarding the lineitem. Specifically, we can navigate from lineitem to nation (or region) through orders and customer or by means of partsupp and supplier. Each path combined with the ending concept semantics produce a different dimensional concept (i.e., the supplier nation or the customer nation).

4.7 The AMDO Tool

The AMDO tool was devised by using the Java SKD v1.4.2 for implementing the algorithm described in this section, and the Protégé-OWL API for manipulating the input ontology. For this reason, as depicted in Fig. 4.15, the AMDO tool has been integrated with Protégé [fBIR] as a plug-in. This feature facilitates the use of the tool: we can take advantage of Protégé to loading and handle an ontology and then, by using the AMDO tab, launch our approach by a single click.

Using the AMDO tool is easy. As shown in Fig. 4.15, we first need to adapt the algorithm to our needs. As discussed in Section 4.3.1.1 and Section 4.3.2.1, we may relax the multidimensional patterns in the *properties frame*. There, the user can state the multiplicity looked in each pattern. Currently, AMDO is still under development, and it is not yet ready to query databases.

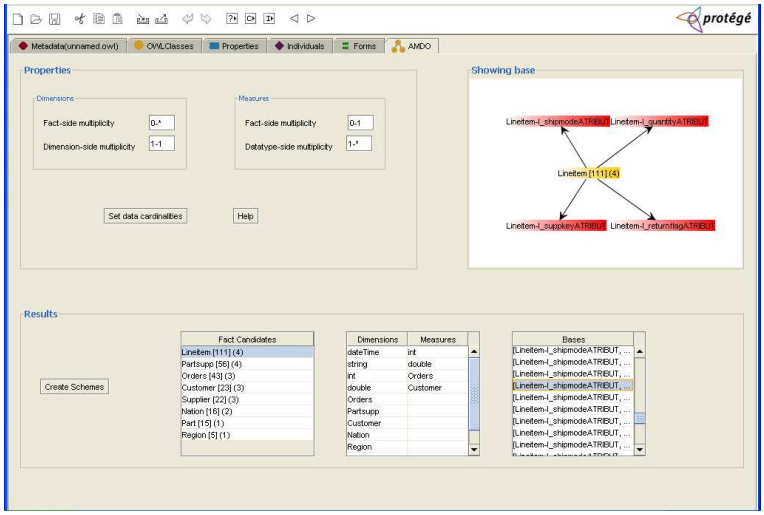


Figure 4.15: The AMDO App. integrated in Protégé

Thus, by now, we provide the *create instance file*, which allows to simulate a database catalog and exploit Prop. 6 in the *determines* function, when discovering bases for a fact.

Next, by clicking on the *create schemes* button, we launch AMDO. The interface works as follows. First, it shows a list of fact candidates (ranked according to the *FactEstimation* function; see Section 4.3.3). The user can select a fact to see which dimensions and measures AMDO proposes for it. Furthermore, AMDO also generates the *feasible bases* for that fact, so the user can see which multidimensional spaces can be formed. When a fact is selected, the *showing base* frame is intended to show the fact, the dimensions and measures selected and the hierarchies produced. However, despite the hierarchies are internally computed, this feature is not yet fully available.

4.8 Conclusions

In this chapter we have presented AMDO: our approach to produce multidimensional schemas from a domain ontology. First, we have presented a set of patterns to identify each multidimensional concept. Importantly, for dimensional concepts, measures and dimension hierarchies, we have introduced two different algorithms to compute these patterns: (i) an ad hoc algorithm exploiting the ontological knowledge contained in expressible DL ontologies, such as OWL DL ontologies, which partially benefits from generic DL reasoners; (ii) and a second algorithm based on *DL-Lite*$_{\mathcal{A}}$, which can be computed by means of generic DL reasoners such as FaCT++. Whereas the first algorithm copes with more expressible DL, the second one restricts the expressivity to the *DL-Lite* family. In both cases, we have discussed the theoretical computational complexity of our algorithms and thanks to the AMDO tool, we have also been able to discuss its behavior

with real world ontologies, in which turn to have a polynomial computational complexity.

Furthermore, we have proposed an algorithm for generating interesting bases (i.e., composite keys) from domain ontologies. In our approach, we take advantage of knowledge captured in the ontology to generate bases hypotheses that are later verified with data. Unlike previous approaches that addressed this task at the data level, we take advantage of ontological knowledge that allows to better depict and prune the searching space. As consequence, our approach does not completely rely on data and it opens new perspectives for data quality processes. We have shown that our algorithm is sound and complete with regard to knowledge captured in the domain ontology and the data, and we have presented the feasibility of our method by means of the statistics raised by the implementation of our algorithm over a case study.

We believe this work to be the first to address the issue of automating the multidimensional design from ontologies. Up to now, traditional approaches were typically carried out manually or were designed to work in an automatic way from relational sources. In our approach, AMDO carries out the data warehouse design process from a domain ontology (i.e., at the conceptual level) which improves the quality of the multidimensional schemas automatically generated. Furthermore, working from ontologies opens new interesting perspectives. For example, we can extend the data warehouse and OLAP concepts to other areas like the Semantic Web. One consequence would be that despite the data warehouse design has been typically guided by data available within the organization, we would be able to integrate external data from the web into our data warehouse to provide additional up-to-date information about our business domain.

Chapter 5

Conclusions and Further Work

" Satisfaction lies in the effort, not in the attainment, full effort is full victory. "

Mohandas Karamchand Gandhi

In this thesis dissertation we have introduced two novel approaches for supporting the data warehouse design task. Previous experiences in this field have shown that the data warehouse multidimensional conceptual schema must be derived from a hybrid approach: i.e., by considering both the end-user requirements and the data sources, as first-class citizens. Currently, several methods (i.e., detailed design approaches) and dissertations (i.e., high level discussions highlighting the necessities in each real scenario) for supporting the data warehouse design task have been introduced in the literature, but none of them provides an integrated and automated solution embracing both aspects. On the one hand, dissertations about how the design task must be adapted to every real-world scenario provide an insightful idea of how to proceed in each case. However, they fail to provide detailed algorithms to undertake this task. On the other hand, detailed methods introduced tend to focus on a narrow-ranged set of scenarios. For example, today, it is assumed that the approach to follow in a scenario where the end-user requirements are clear and well-known is completely different from that in which the end-user requirements are not evident or cannot be easily elicited (for example, this may happen when the users are not aware of the analysis capabilities of their own sources). Similarly, the need to dispose of requirements beforehand is smoothed by the fact of having semantically rich data sources. In lack of that, requirements gain relevance to extract the multidimensional knowledge from the sources.

Importantly, our methods establish a combined and comprehensive framework to decide, according to the inputs provided in each scenario, which is the best approach to follow. In other words, we claim to provide two approaches that, combined, turn up to be exhaustive regarding the scenarios discussed in the literature:

- MDBE follows a classical approach, in which the end-user requirements are well-known beforehand. This approach benefits from the knowledge captured in the data sources, but

guides the design task according to requirements and consequently, it is able to work and handle semantically poorer data sources. In other words, providing high-quality end-user requirements, we can guide the process and overcome the fact of disposing of bad quality (from a semantical point of view) data sources.

- AMDO, as a counterpart, assumes a scenario in which the data sources available are semantically richer. Thus, the approach proposed is guided by a thorough analysis of the data sources, which is properly adapted to shape the output result according to the end-user requirements. In this context, disposing of high-quality data sources we can overcome the fact of lacking of very expressive end-user requirements.

The opposite assumptions of each approach are clearly exemplified with the TPC-H case study used in this document (see Sections 3.5 and 4.6 for a detailed discussion of how every approach applies to this case study). Consider the two main axis of study discussed in Section 1.6:

Requirements Specificity: MDBE integrates the end-user requirements, which lead the whole process, in an automated approach. Therefore, the quality and expressiveness of the input requirements must be high. In the TPC-H case study, this fact is represented by the 22 TPC-H business query. On the contrary, at the beginning of the process, AMDO does not need the end-user requirements to work. Indeed, results provided by the full supply-driven approach (plus its filtering steps) carried out first, are eventually shaped by the end-user decisions a posteriori. In this latter step, the end-user just state his / her requirements by choosing his / her concepts of interest regarding the results provided. In the TPC-H case study, it results in selecting 3 facts out of 5, some oout of tens, and shape the end-user graphs provided for each dimensional concept involved in a selected base.

Data Source Expresiveness: In this case, MDBE is able to even handle denormalized sources and produce high-quality results. In the TPC-H case study, MDBE has enough by querying the logical schema, even if it is not well-formed. On the contrary, AMDO requires high-quality input sources, and it needs to exploit the conceptual information captured in the TPC-H ontology, a mapping between the conceptual schema and the sources (in our case, provided by our reengineering task performed to create the TPC-H ontology) and access the data sources at the instance level (to verify the bases with data).

In short, the requirements / data sources duality introduced in the literature is indeed, carried over the MDBE / AMDO dichotomy. By means of Figures 1.3 and 1.4 (in pages 14 and 15 respectively), this duality means that each approach is placed in an opposite end of the space formed by the analysis axis: MDBE at the end of the *y* axis and AMDO at the end of the *x* axis. MDBE assumes high-quality requirements, whereas AMDO assumes high-quality data sources. Roughly speaking and by means of a slightly language abuse, we say that MDBE is *maximal* regarding requirements, and *minimal* regarding the data sources, whereas AMDO is *maximal* regarding the data sources, and *minimal* regarding requirements.

Furthermore, both approaches are automated at their most, and they provide high-quality outputs. We can exemplify these assertions with the TPC-H case study:

Automation: Although both approaches have opposite assumptions, the user interaction during the automated processes is minimal. In MDBE, the end-user is responsible for verifying the output produced in case of relaxing [C5], [C6] or [C7]. Furthermore, he / she may need to shape the dimension hierarchies when identifying dimensional concepts from denormalized data sources. Note, however, that both assumptions cannot be automated working from relational sources and thus, it is compulsory that the user validates the results provided. About AMDO, the user is only responsible for providing the quality functions and thresholds for filtering results obtained in the supply-driven stage.

Quality Output: By means of the TPC-H case study, we have shown that both approaches, despite working from opposite assumptions, are able to produce the same multidimensional schema: On the one hand, MDBE directly focus on the requirements and, by analyzing the data sources, it provides schemas that can give answer to the multidimensional queries used as inputs. On the other hand, AMDO looks for all the multidimensional knowledge captured in the sources, filters it according to quality indicators and eventually, if we share the same end-user requirements, it derives an equivalent schema to that produced by MDBE. This is sound and relevant, since, for the same scenario (i.e., same data sources and same end-user requirements) we expect to obtain the same result regardless of the approach chosen. Indeed, the only difference between both approaches is *how* we want to obtain the final result (i.e., which is our current scenario). Importantly, we must note that each approach is exhaustive regarding requirements (in case of working with MDBE) or the data sources (when working with AMDO) provided. For example, MDBE can compute *derived measures* or *concept specializations* not explicitly captured in the data sources, but present in the end-user requirements. Similarly, this kind of measures and specializations can only be identified by AMDO if they are captured in the input ontology. Analogously, this also happens regarding *semantic relationships* between multidimensional concepts, *aggregate measures* and *aggregate bases*. The case of the *factless facts*, however, is slightly different. MDBE identifies them by means of requirements, but in AMDO, the *quality function* used to identify facts would be crucial. Finally, there is just one matter of difference in the output produced by both approaches. In any case, AMDO will always provide meaningful aggregation paths (i.e., dimension hierarchies) regarding the knowledge captured in the ontology. However, MDBE is not able to shape dimension hierarchies when facing denormalized sources. In this case, the user is responsible for manually shape them.

Summing up, both approaches automate the design task as much as possible. Note that only the stages devoted to gather / state the end-user requirements are not automated. This is sound by the inherent nature of this task (i.e., gather high-level requirements and formalize them). Furthermore, they can derive the same multidimensional schema by considering the same end-user requirements (e.g., TPC-H business queries) and the same data sources (e.g., the TPC-H relational schema / ontology). Regarding their feasibility, the computational complexity of MDBE is considerably smaller than that of AMDO. However, this is sound. MDBE leads the data source analysis by means of requirements and thus, it is feasible even for large databases. On the contrary, AMDO starts with a supply-driven approach, which thoroughly analyzes the data sources. For this reason, AMDO computational complexity is directly affected (specially

the base algorithm) by the size of the input ontology and the data sources. Nevertheless, the tests carried out for AMDO raised good answer times.

Finally, note that the combination of our approaches establish a work framework covering all the scenarios discussed in the literature: from cases in which the end-user requirements are clear and available beforehand, to those in which they are not; and from cases providing semantically rich data source conceptualizations, to those poorly capturing the domain semantics. Thus, any intermediate scenario is covered by combining both approaches. Furthermore, AMDO and MDBE denote the *minimal* set of assumptions allowing to fully automate the process. For example, if we do not dispose of end-user requirements beforehand nor of a reliable domain conceptualization, we will be forced to manually carry out some of the processes in the design task. In other words, to automate the process and obtain high-quality results we need either a fair picture of the business domain, a clear idea of the multidimensional requirements or an intermediate situation in which the lack of a rich domain conceptualization is (partially) replaced with providing some of the multidimensional requirements, or viceversa.

5.1 Further Work

The two novel methods introduced in this thesis dissertation, provide a formal and comprehensive framework from which extend our current work. Basically, we focus on four main scenarios:

The data warehouse evolution schema: One of the issues gaining more relevance in the literature, is the maintenance and evolution of the conceptual schema of the data warehouses [RALT06]. In this sense, we do believe that both approaches can contribute to develop conceptual schemas and keep track of their evolution. On the one hand, MDBE establishes a framework that can be used incrementally. By launching new queries, the user can see the impact on the final conceptual schema. Thus, the MDBE tool may allow to easily delete / add new queries and keep track of the conceptual schema at each point. On the other hand, AMDO benefits from the reasoning services provided by DL languages and thus, we can extend our approach to consider temporal Description Logics, which would facilitate the data warehouse schema evolution.

Integrating the ETL process and the conceptual schema task designs: Automating the conceptual schema data warehouse design allows to identify many details relevant for the extraction, transformation and loading of data into the data warehouse. In this sense, notions like aggregate measures or bases can help in the ETL design process. In case of using AMDO, we may benefit from the reasoning services provided by DL.

In this work, we have identified different criteria to be considered when implementing the conceptual schema derived at the logical level. One key question to answer when performing this mapping is: which Cells we should materialize? OLAP functionality is based on fast analysis and thus, views and data materialization providing a better performance are a must. The framework established by MDBE provides relevant criteria to be considered for automating this process. For example, according to the requirements provided for producing the conceptual schema, or by means of the criteria introduced in Section 3.2 to avoid the potential translation problems when mapping the multidimensional algebra to SQL.

The analysis of the mapping between the multidimensional algebra and SQL queries also opens new interesting perspectives. We say that a SQL sentence is a cube-query if it captures a multidimensional navigation path (see Section 3.2). Therefore, once the conceptual schema is generated, we can identify which navigation path produces each SQL query used as input. Discovering these navigation paths provide a novel approach for query recommendations for OLAP tools.

Appendices

Appendix A

A Tractable Description Logic: *DL-Lite*$_{\mathcal{A}}$

In Description Logics (DLs), objects with common properties are grouped into *concepts*, and the properties are represented through *roles*, denoting binary relations over the domain of interest. Complex concepts and roles are built inductively by starting from atomic ones (i.e., simple concept and role names) and applying a set of constructs. Different from traditional DLs, and following what is done in other conceptual modeling formalisms such as UML class diagrams [Grob], *DL-Lite*$_{\mathcal{A}}$ distinguishes between (abstract) objects and (data) values. Hence, it distinguishes concepts, denoting sets of objects, from *value-domains*, denoting sets of values, and roles, denoting binary relations between objects, from *attributes*, denoting binary relations between objects and values. More precisely, concepts, roles, value-domains, and attributes in *DL-Lite*$_{\mathcal{A}}$ are formed starting from atomic elements according to the following syntax (where the distinction between *basic* and *arbitrary* elements is relevant in what follows):

	atomic	basic	arbitrary
concept	A	$B \longrightarrow A \mid \exists Q \mid \delta(U)$	$C \longrightarrow B \mid \neg B$
role	P	$Q \longrightarrow P \mid P^-$	$R \longrightarrow Q \mid \neg Q$
value-domain		$E \longrightarrow \rho(U)$	$F \longrightarrow \top_D \mid T_1 \mid \cdots \mid T_n$
attribute	U	$V \longrightarrow U$	$W \longrightarrow V \mid \neg V$

Above, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values; $\rho(U)$ denotes the *range* of U, i.e., the set of values that U relates to objects; $\top_D$ is the universal value-domain; $T_1, \ldots, T_n$ are n pairwise disjoint unbounded value-domains, corresponding to data types, such as `string`, `integer`, etc. In the following, let $\mathsf{Inv}(Q) = P^-$ when $Q = P$, and $\mathsf{Inv}(Q) = P$ when $Q = P^-$. Impo

In *DL-Lite*$_{\mathcal{A}}$, knowledge about the domain is represented by means of an *ontology* (or knowledge base), consisting of a TBox, encoding intensional knowledge, and an ABox, encoding extensional knowledge on specific objects. Specifically, a *DL-Lite*$_{\mathcal{A}}$ TBox is constituted by a set of assertions of the form:

$$B \sqsubseteq C, \quad Q \sqsubseteq R, \quad E \sqsubseteq F, \quad V \sqsubseteq W, \quad (\mathsf{funct}\ Q), \quad (\mathsf{funct}\ U),$$

which respectively denote an inclusion between a basic and an arbitrary concept, role, value-

domain, and attribute, and functionality on a role and on an attribute[1]. As for the ABox, we introduce two disjoint alphabets, Γ_O of *object constants* denoting objects, and Γ_V of *value constants* denoting data values. A *DL-Lite*$_\mathcal{A}$ ABox is a finite set of *membership assertions* of the form (where $a, b \in \Gamma_O$ and $c \in \Gamma_V$):

$$A(a), \qquad P(a, b), \qquad U(a, c).$$

Definition. *A DL-Lite*$_\mathcal{A}$ *ontology* $\mathcal{O}$ *is a pair* $\langle \mathcal{T}, \mathcal{A} \rangle$*, where* $\mathcal{T}$ *is a DL-Lite*$_\mathcal{A}$ *TBox,* $\mathcal{A}$ *is a DL-Lite*$_\mathcal{A}$ *ABox, and the following conditions are satisfied:*

(1) for each atomic role P*, if either* $(\mathsf{funct}\ P)$ *or* $(\mathsf{funct}\ P^-)$ *occur in* $\mathcal{T}$*, then* $\mathcal{T}$ *does not contain assertions of the form* $Q \sqsubseteq P$ *or* $Q \sqsubseteq P^-$ *(for* Q *a basic role);*

(2) for each atomic attribute U*, if* $(\mathsf{funct}\ U)$ *occurs in* $\mathcal{T}$*, then* $\mathcal{T}$ *does not contain assertions of the form* $V \sqsubseteq U$ *(for* V *an atomic attribute).*

Intuitively, these two conditions say that, in a *DL-Lite*$_\mathcal{A}$ TBox, roles and attributes occurring in functionality assertions cannot be specialized. These conditions are crucial for the tractability of reasoning [PLC$^+$08].

The semantics of *DL-Lite*$_\mathcal{A}$ is given in terms of FOL interpretations. An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of a first order structure over the interpretation domain $\Delta^\mathcal{I}$ that is the disjoint union of $\Delta^\mathcal{I}_O$ and $\Delta^\mathcal{I}_V$, and of an *interpretation function* $\cdot^\mathcal{I}$ such that $a^\mathcal{I} \in \Delta^\mathcal{I}_O$ for all $a \in \Gamma_O$, $c^\mathcal{I} \in \Delta^\mathcal{I}_V$ for all $c \in \Gamma_V$, and such that the following conditions are satisfied (below, $o, o' \in \Delta^\mathcal{I}_O$, and $v \in \Delta^\mathcal{I}_V$):

$$\begin{array}{rcl@{\qquad}rcl}
A^\mathcal{I} & \subseteq & \Delta^\mathcal{I}_O & P^\mathcal{I} & \subseteq & \Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_O \\
(\exists Q)^\mathcal{I} & = & \{o \mid \exists o'.\, (o, o') \in Q^\mathcal{I}\} & (P^-)^\mathcal{I} & = & \{(o, o') \mid (o', o) \in P^\mathcal{I}\} \\
(\delta(U))^\mathcal{I} & = & \{o \mid \exists v.\, (o, v) \in U^\mathcal{I}\} & (\neg Q)^\mathcal{I} & = & (\Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_O) \setminus Q^\mathcal{I} \\
(\neg B)^\mathcal{I} & = & \Delta^\mathcal{I}_O \setminus B^\mathcal{I} & & & \\
T_i^\mathcal{I} & \subseteq & \Delta^\mathcal{I}_V & U^\mathcal{I} & \subseteq & \Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_V \\
\top_D^\mathcal{I} & = & \Delta^\mathcal{I}_V & (\neg V)^\mathcal{I} & = & (\Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_V) \setminus V^\mathcal{I} \\
(\rho(U))^\mathcal{I} & = & \{v \mid \exists o.\, (o, v) \in U^\mathcal{I}\} & & &
\end{array}$$

We assume that the *unique name assumption* holds, i.e., different (object and value) constants are interpreted as different domain elements.

We define now when an interpretation $\mathcal{I}$ satisfies a TBox or ABox assertion. Specifically, $\mathcal{I}$ *satisfies*:

- $\alpha_1 \sqsubseteq \alpha_2$, if $\alpha_1^\mathcal{I} \subseteq \alpha_2^\mathcal{I}$;
- $(\mathsf{funct}\ \beta)$, where β is either P, P^-, or U, if $(o, e_1) \in \beta^\mathcal{I}$ and $(o, e_2) \in \beta^\mathcal{I}$ implies $e_1 = e_2$, for each $o \in \Delta^\mathcal{I}_O$, and e_1, e_2 in either $\Delta^\mathcal{I}_O$ or $\Delta^\mathcal{I}_V$;
- $A(a)$ if $a^\mathcal{I} \in A^\mathcal{I}$, $P(a, a')$ if $(a^\mathcal{I}, a'^\mathcal{I}) \in P^\mathcal{I}$, and $U(a, c)$ if $(a^\mathcal{I}, c^\mathcal{I}) \in U^\mathcal{I}$.

[1] Note that an arbitrary element cannot be placed at the left side of a subsumption assertion. This must be guaranteed to preserve the complexity properties of *DL-Lite*$_\mathcal{A}$.

A *model* of an ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$) is an interpretation $\mathcal{I}$ that satisfies all assertions in $\mathcal{O}$ (resp., $\mathcal{T}$). An ontology $\mathcal{O}$ (resp., TBox $\mathcal{T}$) is *satisfiable* if it has at least one model, and $\mathcal{O}$ (resp., $\mathcal{T}$) *logically implies* an assertion α, denoted $\mathcal{O} \models \alpha$ (resp., $\mathcal{T} \models \alpha$), if α is satisfied in all models of $\mathcal{O}$ (resp., $\mathcal{T}$).

A conjunctive query (CQ) q over an ontology $\mathcal{O}$ is an expression of the form $q(\vec{x}) \leftarrow body(\vec{x}, \vec{y})$, where $\vec{x}$ are the so-called *distinguished variables*, $\vec{y}$ are the *non-distinguished* variables, and $body(\vec{x}, \vec{y})$ is a set of atoms of the form $A(x_o)$, $P(x_o, x'_o)$, $T_i(x_v)$, or $U(x_o, x_v)$, where x_o, x'_o are variables in $\vec{x}$ or $\vec{y}$ or constants in Γ_O, and x_v is a variable in $\vec{x}$ or $\vec{y}$ or a constant in Γ_V. Notice that CQs corresponds to SELECT-PROJECT-JOIN SQL queries, and hence are the queries most commonly posed to relational DBs. The query $q(\vec{x}) \leftarrow body(\vec{x}, \vec{y})$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\vec{e} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ such that, when we assign $\vec{e}$ to the variables $\vec{x}$, the first-order logic formula $\exists \vec{y}.\varphi(\vec{x}, \vec{y})$, where $\varphi(\vec{x}, \vec{y})$ is the conjunction of atoms in $body(\vec{x}, \vec{y})$, evaluates to true in $\mathcal{I}$. The reasoning service we are interested in is *(conjunctive) query answering*: given an ontology $\mathcal{O}$ and a query q over $\mathcal{O}$, return the *certain answers* to q over $\mathcal{O}$, i.e., all tuples $\vec{t}$ of elements of $\Gamma_V \cup \Gamma_O$ such that $\vec{t}^{\mathcal{I}} \in q^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$, denoted $\mathcal{K} \models q(\vec{t})$.

As shown in [CDGL$^+$07, PLC$^+$08], all forms of inference over a *DL-Lite* ontology (e.g., satisfiability, logical implication, and CQ answering) can be done in polynomial time in the size of the TBox, and in AC_0[2] in the size of the ABox (i.e., w.r.t. data complexity [Var82]). In particular, to compute the certain answers of a CQ q, we can: (i) by using the assertions in the TBox, rewrite q to a union Q of CQs (which is directly translatable to an SQL query), and (ii) evaluate Q over the database corresponding to the ABox assertions. In this way, all forms of inference in *DL-Lite*$_{\mathcal{A}}$ can be carried out by exploiting standard commercial relational DB technology for manipulating the data (i.e., the ABox).

[2] AC_0 is the complexity class that corresponds to the complexity in the size of the data of evaluating a first-order (i.e., SQL) query over a relational database (see, e.g., [AHV95]).

Appendix B

EU-Car Rentals

This appendix presents the EU-Car Rentals case study. For the sake of understandability, we present here the UML class diagram, which is divided in thematic areas: Branch, Rental Agreement, Rental Agreement subclasses and Cars, Discounts and Enumerations (a detailed discussion about this case study can be found in [FQO03]). We also provide a OWL DL ontology capturing the same conceptual domain in the http://www.lsi.upc.edu/∼ oromero/EUCarRental.owl URL address.

Branch

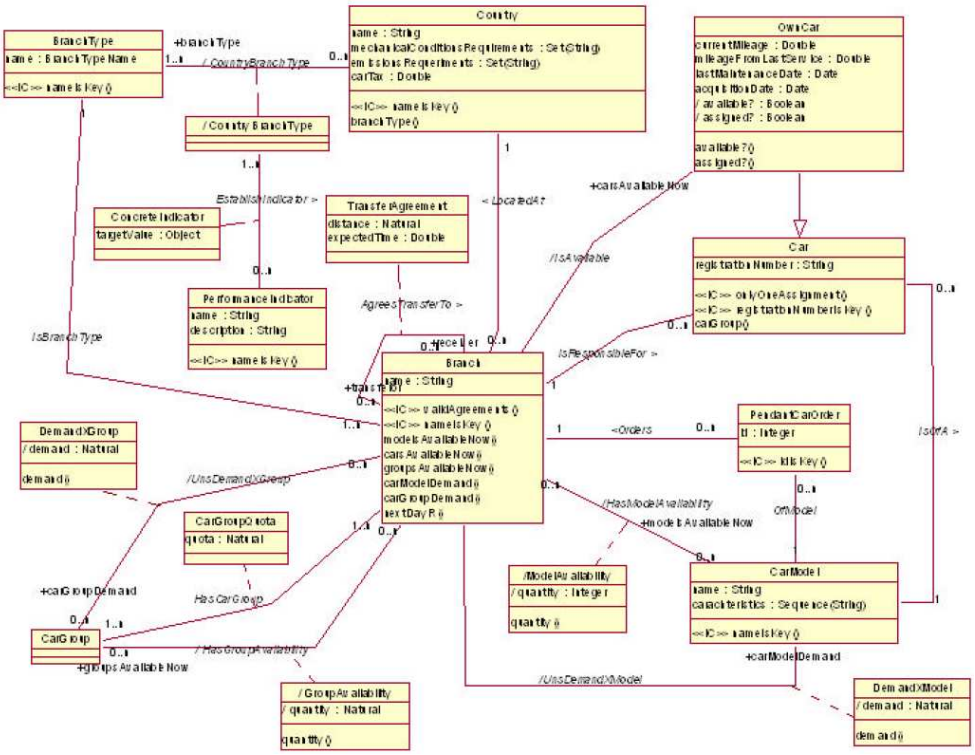


Figure B.1: EU-Car Rentals class diagram: brand

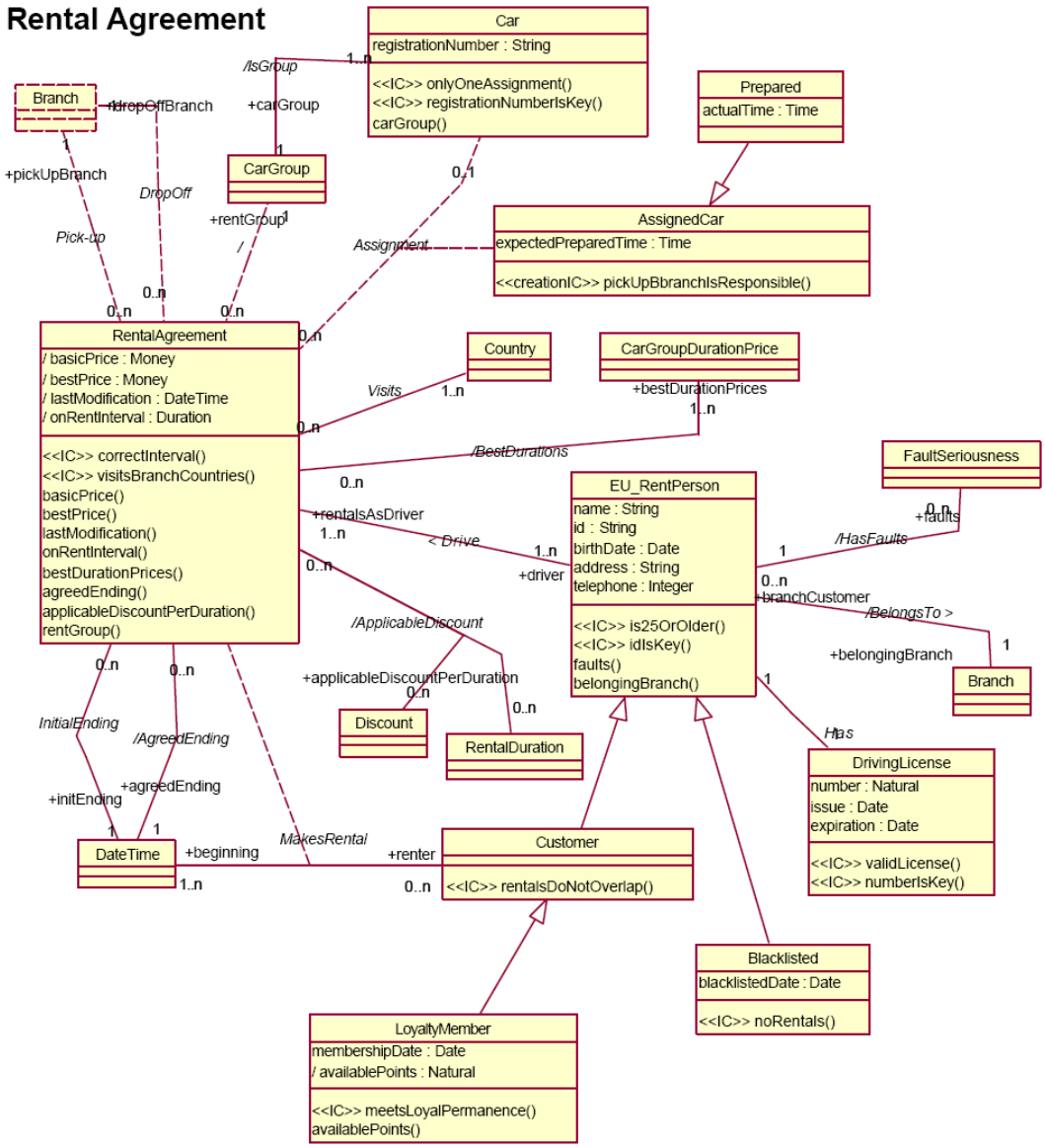
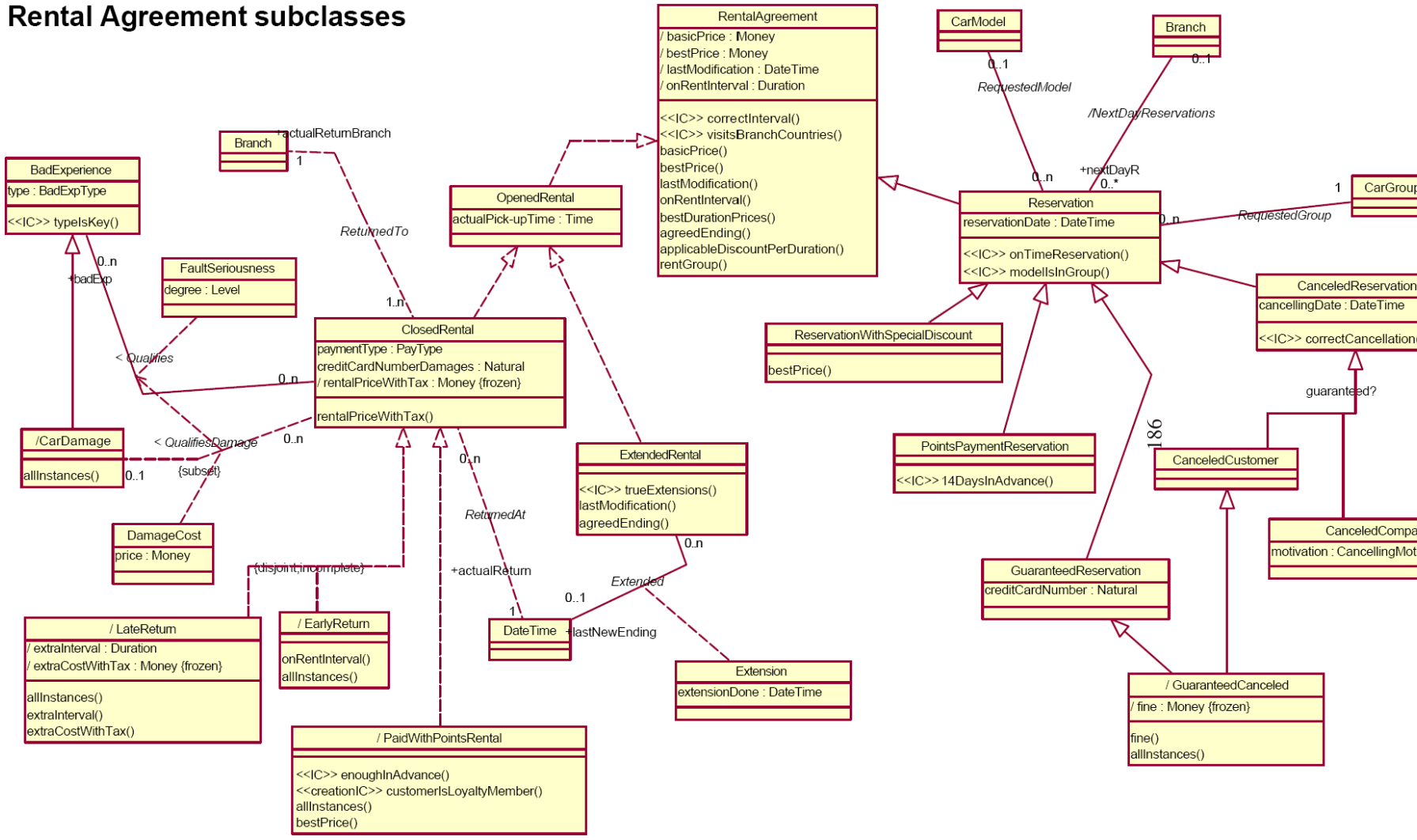
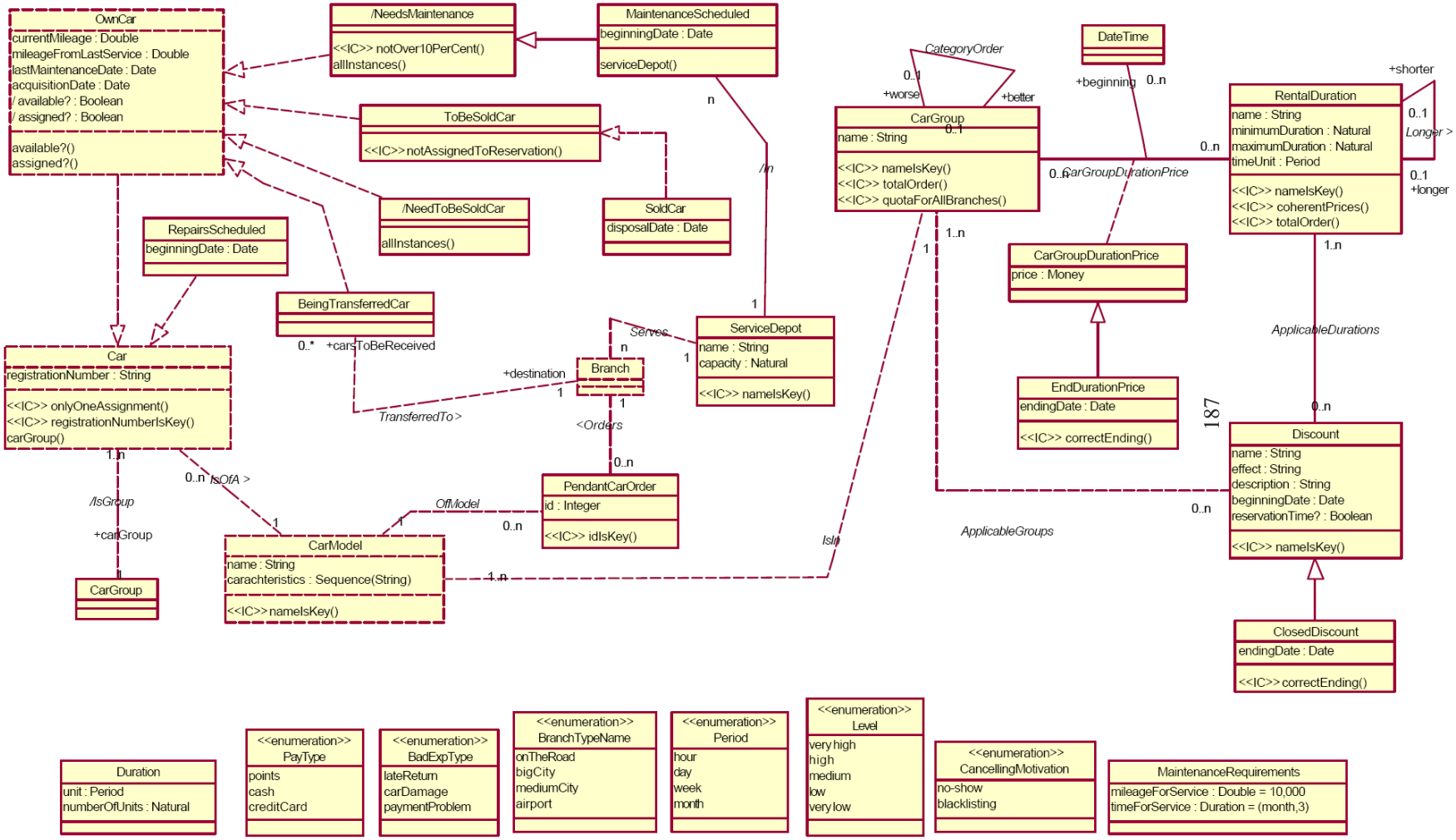


Figure B.2: EU-Car Rentals class diagram: rental agreement

Rental Agreement subclasses



Cars, Discounts and Enumerations



Appendix C

List of Publications

This appendix presents the list of publications concerning this thesis. It is divided in 3 sections, each one referring to the related work (chapter 2), the MDBE (chapter 3) and AMDO (chapter 4) chapters.

C.1 Related to Chapter 2

- Oscar Romero and Alberto Abelló. A Survey of Multidimensional Modeling Methodologies. *International Journal of Data Warehousing and Mining (IJDWM)*. Volume 5(2). IGI Publishing, April - June 2009. Pages 1-23.

Abstract: In the last years, data warehousing systems have gained relevance to support decision making within organizations. The core component of these systems is the data warehouse and nowadays it is widely assumed that the data warehouse design must follow the multidimensional paradigm. Many methods have been presented to support the multidimensional design of the data warehouse. First methods introduced were requirement-driven but the semantics of a data warehouse (since the data warehouse is the result of homogenizing and integrating relevant data of the organization in a single and detailed view of the organization business) require to also consider data sources along the design process. Considering data sources gave rise to several data-driven methods that automate the data warehouse design process from relational data sources. Currently, research on multidimensional modeling is still a hot topic and we have two main research lines. On the one hand, new hybrid automatic methods have been introduced proposing to combine data-driven and requirement-driven approaches. These methods focus on automating the whole process and improving the feedback retrieved by each approach to produce better results. On the other hand, new approaches focus on considering other kind of structured data sources that have gained relevance in the last years such as ontologies or XML. In this paper we present a survey of multidimensional design methods. We present the most rele-

vant methods introduced in the literature and a detailed comparison showing main features of each approach.

Scope: This publication maps to Section 2.1, as it discusses the current state of the art for multidimensional modeling. It introduces the comparison between the methods (as well as the criteria used for the comparison) and sets a discussion about the conclusions reached.

- Alberto Abelló and Oscar Romero. On-line Analytical Processing. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 19491954. Springer, 2009.

Abstract: The Encyclopedia of Databases provide a comprehensive reference to about 1,400 entries, covering key concepts and terms in the broad field of database systems. Entries include in-depth essays and shorter descriptions of terms, definition, key words, historical background, illustrations, key applications, and a bibliography. It provides a simple-to-use alphabetical format and extensive cross references enable both user-friendly and efficient searches.

The Encyclopedia of Databases, a comprehensive work, provides easy access to relevant information on all aspects of very large databases. This encyclopedia features alphabetical organization of concepts covering main areas of very large databases. These 1000 entries offer convenient access to information in the field of databases with definitions and illustrations of basic terminology, concepts, methods, and algorithms, references to literature, and cross-references to other entries and journal articles. Topics for the encyclopedia were selected by a distinguished international advisory board, and written by world class experts in the field.

The Encyclopedia of Databases is designed to meet the needs of research scientists, professors and graduate-level students in computer science and engineering. This encyclopedia is also suitable for practitioners in industry.

Abstract: In this publication we properly define OLAP, and provide an insightful look at the historical background and current state of the art concerning this concept. Part of this publication maps to Section 2.2.

C.2 Related to Chapter 3

- Oscar Romero and Alberto Abelló. On the Need of a Reference Algebra for OLAP. In proceedings of the *9th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'07)*. Lecture Notes of Computer Science. Volume 4654. Springer, 2007. Pages 99-110.

Abstract: Although multidimensionality has been widely accepted as the best solution for conceptual modelling, there is not such agreement about the set of operators to handle multidimensional data. This paper presents a comparison of the existing multidimensional algebras trying to find a common backbone, as well as it discusses the necessity of a reference multidimensional algebra and the current state of the art.

Scope: This paper maps to Sections 2.2 and 3.2.1.

- Oscar Romero and Alberto Abelló. Improving Automatic SQL Translation for ROLAP Tools. In proceedings of the *9th Jornadas en Ingeniería del Software y Bases de Datos (JISBD'05)*. Volume 284. Thomson-Paraninfo, 2005. Pages 123-130.

Abstract: In the last years, despite a vast amount of work have been devoted to modeling multidimensionality, the multidimensional algebra translation to SQL have been overlooked. ROLAP tools automatically generate a cube-query according to the operations performed by the user. The SQL translation does not represent a problem when considering isolated multidimensional operations but when mixing up together modifications brought about by a set of operations in the same cube-query, some conflicts could emerge depending on the operations involved. Therefore, if these problems are not detected and treated appropriately, the automatic translation can retrieve unexpected results. In this paper, we define and classify conflicts raised when automatically translating a multidimensional algebra to SQL, and analyze how to solve or minimize their impact.

Scope: This paper maps to Section 3.2.2.

- Oscar Romero and Alberto Abelló. Multidimensional Design by Examples. In proceedings of the *8th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'06)*. Lecture Notes of Computer Science. Volume 4081. Springer, 2006. Pages 85-94.

Abstract: In this paper we present a method to validate user multidimensional requirements expressed in terms of SQL queries. With this aim, our approach automatically generates and proposes the set of multidimensional schemas satisfying the user requirements, from the organizational relational schemas. If no multidimensional schema is generated for a query, we can state that this requirement is not multidimensional.

Scope: This paper is a preliminary version of Section 3.4.

- Oscar Romero and Alberto Abelló. Automatic Multidimensional Design of Data Warehouses from Requirements. Technical Report LSI-09-25-R. Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), 2009 (*submitted for publication*).

Abstract: The ideal scenario to derive the multidimensional conceptual schema of a data warehouse would embrace 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. However, current automatable methodologies follow a full data-driven approach whereas current requirement-driven approaches are not automatable as they tend to work with requirements at a high level of abstraction. In this paper we present a novel method 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.

Scope: This paper is an extended version of the previous paper and maps to the whole Section 3.4. Furthermore, it is a revisited work including all the last advances we made in the last two years that has been submitted to a journal. It also includes Sections 3.1, 3.3 and 3.5.

- Oscar Romero and Alberto Abelló. MDBE: Automatic Multidimensional Modeling. In proceedings of the *27th International Conference on Conceptual Modeling (ER'08)*. Demo Track. Lecture Notes of Computer Science. Volume 5231. Springer, 2008. Pages 534-535.

Abstract: The goal of this demonstration is to present MDBE, a tool implementing our methodology for automatically deriving multidimensional schemas from relational sources, bearing in mind the end-user requirements. Our approach starts gathering the end-user information requirements that will be mapped over the data sources as SQL queries. Based on the constraints that a query must preserve to make multidimensional sense, MDBE automatically derives multidimensional schemas which agree with both the input requirements and the data sources.

Scope: This paper maps to Section 3.6.

C.3 Related to Chapter 4

- Oscar Romero and Alberto Abelló. Generating Multidimensional Schemas from the Semantic Web. In proceedings of the *CAiSE'07 Forum at the 19th International Conference*

on Advanced Information Systems Engineering (CAiSE Forum'07). Volume 247. CEUR-WS.org, 2007. Pages 69-72.

Abstract: In this paper, we introduce a semi-automatable method aimed to find the business multidimensional concepts from an ontology representing the organization domain. With these premises, our approach falls into the Semantic Web research area, where ontologies play a key role to provide a common vocabulary describing the meaning of relevant terms and relationships among them.

Scope: This poster presents a preliminary version of the ideas discussed in Chapter 4.

- Oscar Romero and Alberto Abelló. Automating Multidimensional Design from Ontologies. In proceedings of the *ACM 10th International Workshop on Data Warehousing and OLAP (DOLAP'07)*. ACM, 2007. Pages 1-8.

Abstract: This paper presents a new approach to automate the multidimensional design of Data Warehouses. In our approach we propose a semi-automatable method aimed to find the business multidimensional concepts from a domain ontology representing different and potentially heterogeneous data sources of our business domain. In short, our method identifies business multidimensional concepts from heterogeneous data sources having nothing in common but that they are all described by an ontology.

Scope: This paper maps to Section 4.3. It presents a preliminary version of AMDO's core, and a preliminary version of the algorithm for computing bases in Section 4.5.1.

- Oscar Romero and Alberto Abelló. A Framework for Multidimensional Design of Data Warehouses from Ontologies. Technical Report LSI-09-26-R. Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), 2009 (*submitted for publication*).

Abstract: Some research efforts have proposed the automation of the data warehouse design in order to free this task of being (completely) performed by an expert, and facilitate the whole process. Most advanced approaches exclusively work over relational sources and perform a detailed analysis of the data sources to identify the multidimensional concepts in a reengineering process. Starting from a logical schema, however, may present some inconveniences. A logical schema is tied to the design decisions made when devising the system and these decisions either made to fulfill the system requirements (for instance, improve query answering, avoid insertion / deletion anomalies, preserve features inherited from legacy systems, etc.) or naively made by nonexpert users, have a big impact on the quality of the multidimensional schemas got by current automatable approaches. In this paper, we introduce our approach for automatically deriving the multidimensional schema from a domain ontology. Our goals are mainly two: i) we want to improve the

quality of the output got (by working over a conceptual formalization of the domain instead of a logical one) and ii) we want to automate the process. This second goal is the main reason for choosing ontologies instead of other conceptual formalizations, as ontology languages provide reasoning services that will facilitate the automation of our task.

Scope: This paper maps to Sections 4.1, 4.2, 4.3 and 4.4.2. It is an extended and improved version of previous paper that has been submitted to a journal.

- Oscar Romero and Alberto Abelló. Discovering Meaningful Keys from Ontologies. Technical Report LSI-09-24-R. Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), 2009.

Abstract: Object identification is a crucial step in most information systems. Nowadays, we have many different ways to identify entities such as surrogates, keys and object identifiers. However, not all of them guarantee the entity identity. Many works have been introduced in the literature for discovering meaningful keys, but all of them work at the logical or data level and they share some inherent constraints. Addressing it at the logical level, we may miss some important data dependencies, while the cost to identify data dependencies at the data level may not be affordable. In this paper we propose an approach for discovering meaningful keys from domain ontologies. In our approach, we guide the process at the conceptual level and we introduce a set of pruning rules for improving the performance by reducing the number of key hypotheses generated and to be verified with data. Finally, we also introduce a simulation over a real world case study to show the feasibility of our method.

Scope: This paper maps to Section 4.5.1. It introduces a detailed and improved version of our previous algorithm for discovering bases (i.e., keys).

- Oscar Romero, Diego Calvanese, Alberto Abelló and Mariano Rodríguez. Discovering Functional Dependencies from Ontologies. In proceedings of the *ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP'09)*. ACM, 2009. Pages 1-8.

Abstract: Discovering functional dependencies is a fundamental step in the design of relational databases and in most system reengineering processes. Typically, this task has been performed over relational databases, at the logical or physical level. Those works addressing it at the logical level, often make some unrealistic assumptions (such as completeness of the data structures or similar names for semantically related attributes), while those addressing it at the physical level propose solutions that are computationally expensive, whose performance deteriorates with a large number of attributes or instances, and which cannot tolerate erroneous data. To overcome these limitations, while also better capturing data dependencies, we propose to rely instead on a conceptual representation of the domain of interest, specified in ER and formalized through a domain ontology expressed

in the DL-Lite Description Logic.We propose an algorithm to discover functional dependencies from the domain ontology that exploits the inference capabilities of DL-Lite, thus fully taking into account the semantics of the domain. We also provide an evaluation of our approach in a real-world scenario.

Scope: This paper focuses on using generic reasoning for computing the multidimensional patterns over ontologies. Thus, it maps to Sections 4.4.3 and 4.4.

Bibliography

[ACK+07] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. Reasoning over Extended ER Models. In *Proc. of 26th Int. Conf. on Conceptual Modeling*, volume 4801 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2007.

[AGS97] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimensional Databases. In *Proc. of the 13th Int. Conf. on Data Engineering*, pages 232–243. IEEE, 1997.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[ASS03] A. Abelló, J. Samos, and F. Saltor. Implementing Operations to Navigate Semantic Star Schemas. In *Proc. of 6th Int. Workshop on Data Warehousing and OLAP*, pages 56–62. ACM, 2003.

[ASS06] A. Abelló, J. Samos, and F. Saltor. **YAM**^{2} (Yet Another Multidimensional Model): An Extension of UML. *Information Systems*, 31(6):541–567, 2006.

[BCC+01] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, and S. Paraboschi. Designing Data Marts for Data Warehouses. *ACM Trans. Softw. Eng. Methodol.*, 10(4):452–483, 2001.

[BCG05] G.D. Berardi, D. Calvanese, and D. Giacomo. Reasoning on UML Class Diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.

[BCM+03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BL04] M. J. A. Berry and G. S. Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. John Wiley & Sons, Inc., 2004.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5), 2001.

[BvE99] M. Böhnlein and A. Ulbrich vom Ende. Deriving Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems. In *Proc. of 2nd Int. Workshop on Data Warehousing and OLAP*, pages 15–21. ACM, 1999.

[CCDGL02] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. A Formal Framework for Reasoning on UML Class Diagrams. In *Proc. of 11th Int. Symposium on Foundations of Intelligent Systems*, volume 2366 of *Lecture Notes of Computer Science*, pages 503–513. Springer, 2002.

[CCS93] E. F Codd, S.B. Codd, and C.T. Salley. Providing OLAP (On Line Analytical Processing) to Users-Analysts: an IT Mandate. *E. F. Codd and Associates*, 1993.

[CDGL01] D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In *Proc. of IJCAI 2001*, pages 155–160, 2001.

[CDGL+06] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data Complexity of Query Answering in Description Logics. In *Proc. of Knowledge Representation 2006*, pages 260–270, 2006.

[CDGL+07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[CDGL+08] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-Based Identification Constraints in Description Logics. In *Proc. of 11th Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 231–241. AAAI Press, 2008.

[CDGL+09] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Conceptual Modeling for Data Integration. In A. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, editors, *Essays in Honour of John Mylopoulos*, Lecture Notes in Computer Science. Springer, 2009.

[Che76] P. P. S. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[CKV90] S. S. Cosmadakis, P. C. Kanellakis, and M. Vardi. Polynomial-Time Implication Problems for Unary Inclusion Dependencies. *J. of the ACM*, 37(1):15–46, January 1990.

[CLN98] D. Calvanese, M. Lenzerini, and D. Nardi. Description Logics for Conceptual Data Modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publishers, 1998.

[CMTV00] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest Pair Queries in Spatial Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 189–200. ACM, 2000.

[Cod72] E. F. Codd. Relational Completeness of Data Base Sublanguages. *Database Systems*, pages 65–98, 1972.

[Cod90] E. F. Codd. *The Relational Model for Database Management, Version 2*. Addison-Wesley, 1990.

[CT97] L. Cabibbo and R. Torlone. Querying Multidimensional Databases. In *Proc. of the 6th International Workshop on Database Programming Languages*, volume 1369 of *LCNS*, pages 319–335. Springer, 1997.

[CT98a] L. Cabibbo and R. Torlone. A Logical Approach to Multidimensional Databases. In *Proc. of 6th Int. Conf. on Extending Database Technology*, volume 1377 of *Lecture Notes of Computer Science*, pages 183–197. Springer, 1998.

[CT98b] L. Cabibbo and R. Torlone. From a Procedural to a Visual Query Language for OLAP. In *Proc. of the 10th Int. Conf. on Scientific and Statistical Database Management*, pages 74–83. IEEE, 1998.

[DKM08] J. Demetrovics, G.O.H. Katona, and D. Miklós. Functional Dependencies Distorted by Errors. *Discrete Applied Mathematics*, 156(6):862–869, 2008.

[DT95] J. Demetrovics and V. D. Thi. Some Remarks On Generating Armstrong And Inferring Functional Dependencies Relation. *Acta Cybern.*, 12(2):167–180, 1995.

[fBIR] Stanford Center for Biomedical Informatics Research. Protégé-OWL API. http://protege.stanford.edu/plugins/owl/api/ (last access 17/12/2009).

[FBSV00] E. Franconi, F. Baader, U. Sattler, and P. Vassiliadis. Multidimensional Data Models and Aggregation. In *Fundamentals of Data Warehousing*. Springer, 2000. M. Jarke, M. Lenzerini, Y. Vassilious and P. Vassiliadis editors.

[FK04] E. Franconi and A. Kamble. The GMD Data Model and Algebra for Multidimensional Information. In *Proc. of the 16th Int. Conf. on Advanced Information Systems Engineering*, volume 3084 of *Lecture Notes of Computer Science*, pages 446–462. Springer, 2004.

[FQO03] Leonor Frías, Anna Queralt, and Antoni Olivé. EU-Rent Car Rentals Specification. Technical report, "Departament de Llenguatges i Sistemes Informtics", 2003. http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=690 (last access 17/12/2009).

[FS99] P. A. Flach and I. Savnik. Database Dependency Discovery: A Machine Learning Approach. *AI Commun.*, 12(3):139–160, 1999.

[GDD07] D. Gaševic, D. Djuric, and V. Devedžic. MDA-based Automatic OWL Ontology Development. *Int. Journal on Software Tools for Technology Transfer*, 9(2):103–117, 2007.

[GL97] M. Gyssens and L. Lakshmanan. A Foundation for Multi-dimensional Databases. In *Proc. of 23rd Int. Conf. on Very Large Data Bases*, pages 106–115. Morgan Kaufmann, 1997.

[GMR98] M. Golfarelli, D. Maio, and S. Rizzi. The Dimensional Fact Model: A Conceptual Model for Data Warehouses. *Int. Journal of Cooperative Information Systems*, 7(2-3):215–247, 1998.

[Goo] Google. Google Scholar. http://scholar.google.com/ (last access 17/12/2009).

[GR98] M. Golfarelli and S. Rizzi. Methodological Framework for Data Warehouse Design. In *Proc. of 1st ACM Int. Workshop on Data Warehousing*, pages 3–9. ACM, 1998.

[GR09] M. Golfarelli and S. Rizzi. *Data Warehouse Design. Modern Principles and Methodologies.* McGraw-Hill, 2009.

[GRG05] P. Giorgini, S. Rizzi, and M. Garzetti. Goal-oriented Requirement Analysis for Data Warehouse Design. In *Proc. of 8th Int. Workshop on Data Warehousing and OLAP*, pages 47–56. ACM Press, 2005.

[Groa] Object Management Group. Object Constraint Language (OCL), Version 2.0. http://www.omg.org/technology/documents/formal/ocl.htm (last access 17/12/09).

[Grob] Object Management Group. Unified Modeling Language (UML), Version 2.2. http://www.omg.org/technology/documents/formal/uml.htm (last access 17/12/09).

[Har] Harzing.com. Publish or Perish. www.harzing.com/pop.htm (last access 17/12/2009).

[HCTJ93] J.L. Hainaut, M. Chandelon, C. Tonneau, and M. Joris. Contribution to a Theory of Database Reverse Engineering. In *Proc. of the 1st Working Conf. on Reverse Engineering*, pages 161–170. IEEE, 1993.

[HLV00] B. Hüsemann, J. Lechtenbörger, and G. Vossen. Conceptual Data Warehouse Modeling. In *Proc. of 2nd Int. Workshop on Design and Management of Data Warehouses*, page 6. CEUR-WS.org, 2000.

[HM01] V. Haarslev and R. Möller. Description of the RACER System and its Applications. In *Working Notes of the 2001 Int. Description Logics Workshop*. CEUR-WS.org, 2001.

[HM08] T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. Morgan Kauffman, 2008.

[Hor98] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of 6th Conf. on Principles of Knowledge Representation and Reasoning*, pages 636–649. Morgan Kaufmann, 1998.

[HS97] M-S. Hacid and U. Sattler. An Object-Centered Multi-dimensional Data Model with Hierarchically Structured Dimensions. In *Proc. of IEEE Knowledge and Data Engineering Exchange Workshop*. IEEE, 1997.

[HS98] M-S. Hacid and U. Sattler. Modeling Multidimensional Database: A formal Object-Centered Approach. In *Proc. of the 6th European Conference on Information Systems*, 1998.

[Inm92] W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., 1992.

[JHP04] M. R. Jensen, T. Holmgren, and T. B. Pedersen. Discovering Multidimensional Structure in Relational Data. In *6th Int. Conf. on Data Warehousing and Knowledge Discovery*, volume 3181 of *Lecture Notes of Computer Science*, pages 138–148. Springer, 2004.

[Kim96] R. Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, Inc., 1996.

[Klu82] A. Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the Association for Computing Machinery.*, 29(3):699–717, 1982.

[KRTR98] R. Kimball, L. Reeves, W. Thornthwaite, and M. Ross. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley & Sons, Inc., 1998.

[Lar99] K.S. Larsen. On Grouping in Relational Algebra. *Int. Journal of Foundations of Computer Science.*, 10(3):301–311, 1999.

[LAW98] W. Lehner, J. Albrecht, and H. Wedekind. Normal Forms for Multidimensional Databases. In *Proc. of 10th Int. Conf. on Statistical and Scientific Database Management*, pages 63–72. IEEE, 1998.

[LBMS02] B. List, R. M. Bruckner, K. Machaczek, and J. Schiefer. A Comparison of Data Warehouse Development Methodologies Case Study of the Process Warehouse. In *Proc. of 13th Int. Conf. on Database and Expert Systems Applications*, volume 2453 of *Lecture Notes of Computer Science*, pages 203–215. Springer, 2002.

[Leh98] W. Lehner. Modelling Large Scale OLAP Scenarios. In *Proc. of 6th Int. Conf. on Extending Database Technology*, volume 1377 of *Lecture Notes of Computer Science*, pages 153–167. Springer, 1998.

[Len02] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of 21th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 233–246. ACM, 2002.

[Lim97] W. M. Lim. Discovery of Constraints from Data for Information System Reverse Engineering. In *Proc. of the 2nd Australian Soft. Eng. Conf.*, pages 39–48. IEEE, 1997.

[LS97] H.J. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Proc. of 9th Int. Conf. on Scientific and Statistical Database Management*, pages 132–143. IEEE, 1997.

[LW96] C. Li and X.S. Wang. A Data Model for Supporting On-Line Analytical Processing. In *Proc. of 5th Int. Conf. on Information and Knowledge Management*, pages 81–88. ACM, 1996.

[M. 92] M. Kantola, H. Mannila, K.-J. Rih, and H. Siirtola. Discovering Functional and Inclusion Dependencies in Relational Databases. *Int. J. of Intelligent Systems 7*, pages 591–607, 1992.

[Mic] Microsoft. MDX Specification. http://msdn.microsoft.com/en-us/library/aa216767.aspx. Last access: 17/12/2009.

[MK00] D.L. Moody and M.A. Kortink. From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design. In *Proc. of 2nd Int. Workshop on Design and Management of Data Warehouses*. CEUR-WS.org, 2000.

[MLT09] J.N. Mazón, J. Lechtenbörger, and Juan Trujillo. A Survey on Summarizability Issues in Multidimensional Modeling. *Data & Knowledge Engineering*, 68(12):1452–1469, 2009.

[MR92] H. Mannila and K. Räihä. On the Complexity of Inferring Functional Dependencies. *Discrete Applied Mathematics*, 40(2):237–243, 1992.

[MTL07] J.N. Mazón, J. Trujillo, and J. Lechtenborger. Reconciling Requirement-Driven Data Warehouses with Data Sources Via Multidimensional Normal Forms. *Data & Knowledge Engineering*, 23(3):725–751, 2007.

[Oli04] Antoni Olivé. On the Role of Conceptual Schemas in Information Systems Development. In *Proc. of 9th Int. Conf. on Reliable Software Technologies (Ada-Europe 2004)*, volume 3063 of *Lecture Notes in Computer Science*, pages 16–34. Springer, 2004.

[P. 09] P. O'Neil and E. O'Neil and X. Chen. The Star Schema Benchmark, http://www.cs.umb.edu/∼poneil/StarSchemaB.PDF (last access 17/12/2009).

[PACW06] N. Prat, J. Akoka, and I. Comyn-Wattiau. A UML-based Data Warehouse Design Method. *Decision Support Systems*, 42(3):1449–1473, 2006.

[PD02] C. Phipps and K. C. Davis. Automating Data Warehouse Conceptual Schema Design and Evaluation. In *Proc. of 4th Int. Workshop on Design and Management of Data Warehouses*, volume 58, pages 23–32. CEUR-WS.org, 2002.

[Ped00] T.B. Pedersen. *Aspects of Data Modeling and Query Processing for Complex Multidimensional Data*. PhD thesis, Faculty of Engineering and Science, 2000.

[Pen05] Nigel Pendse. Market Segment Analysis. *OLAP Report*, Last updated on February 11, 2005. http://www.olapreport.com/Segments.htm (Last access: July 2009).

[Pen08] Nigel Pendse. What is OLAP? *OLAP Report*, Last updated on March 3, 2008. http://www.olapreport.com/fasmi.htm (Last access: July 2009).

[PJ01] T.B. Pedersen and C.S. Jensen. Multidimensional Database Technology. *IEEE Computer*, 34(12):40–46, 2001.

[PLC+08] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking Data to Ontologies. *Journal on Data Semantics*, 10:133–173, 2008.

[PSG04] N. Prakash, Y. Singh, and A. Gosain. Informational Scenarios for Data Warehouse Requirements Elicitation. In *23rd International Conference on Conceptual Modeling, ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2004.

[RA05] O. Romero and A. Abelló. Improving Automatic SQL Translation for ROLAP Tools. *Proc. of 9th Jornadas en Ingeniera del Software y Bases de Datos*, 284(5):123–130, 2005.

[RA06] O. Romero and A. Abelló. Multidimensional Design by Examples. In *Proc. of 8th Int. Conf. on Data Warehousing and Knowledge Discovery*, volume 4081 of *Lecture Notes of Computer Science*, pages 85–94. Springer, 2006.

[RA07a] O. Romero and A. Abelló. Automating Multidimensional Design from Ontologies. In *Proc. of ACM 10th Int. Workshop on Data Warehousing and OLAP*, pages 1–8. ACM, 2007.

[RA07b] O. Romero and A. Abelló. On the Need of a Reference Algebra for OLAP. In *Proc. of 9th Int. Conf. on Data Warehousing and Knowledge Discovery*, volume 4654 of *Lecture Notes of Computer Science*, pages 99–110. Springer, 2007.

[RALT06] S. Rizzi, A. Abelló, J. Lechtenbörger, and J. Trujillo. Research in Data Warehouse Modeling and Design: Dead or Alive? In *Proc. of ACM 9th International Workshop on Data Warehousing and OLAP*, pages 3–10. ACM, 2006.

[RB01] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[RCARM09] O. Romero, D. Calvanese, A. Abelló, and M. Rodriguez-Muro. Discovering Functional Dependencies for Multidimensional Design. In *Proc. of ACM 12th Int. Conf. on Data Warehousing and OLAP*, pages 1–8. ACM, 2009.

[RG03] R. Ramakrishnan and J. Gehrke, editors. *Database Management Systems*. McGraw Hill, 2003.

[SBHR06] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. GORDIAN: Efficient and Scalable Discovery of Composite Keys. In *Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006)*, pages 691–702. ACM, 2006.

[SKD07] I.Y. Song, R. Khare, and B. Dai. SAMSTAR: A Semi-Automated Lexical Method for Generating STAR Schemas from an ER Diagram. In *Proc. of the 10th Int Workshop on Data Warehousing and OLAP*, pages 9–16. ACM, 2007.

[SLB02] J. Schiefer, B. List, and R. M. Bruckner. A Holistic Approach For Managing Requirements Of Data Warehouse Systems. In *8th Americas Conference on Information Systems (AMCIS 2002)*, pages 77–87, 2002.

[TD97] H. Thomas and A. Datta. A Conceptual Model and Algebra for On-Line Analytical Processing in Data Warehouses. In *Proc. of the 7th Workshop on Information Technologies and Systems*, pages 91–100, 1997.

[TD01] H. Thomas and A. Datta. A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases. *Information Systems*, 12(1):83–102, 2001.

[Tra09] Transaction Processing Performance Council. TPC-H Version 2.9.0, http://www.tpc.org/tpch/default.asp (last access 17/12/2009).

[TW05] D. Toman and G. E. Weddell. On the Interaction Between Inverse Features and Path-functional Dependencies in Description Logics. In *Proc. of IJCAI 2005*, pages 603–608, 2005.

[TW08] D. Toman and G. E. Weddell. On Keys and Functional Dependencies as First-Class Citizens in Description Logics. *J. of Automated Reasoning*, 40(2–3):117–132, 2008.

[TZ04] H. B. K. T. and Y. Zhao. Automated Elicitation of Functional Dependencies from Source Codes of Database Transactions. *Information & Software Technology*, 46(2):109–117, 2004.

[Var82] M. Y. Vardi. The Complexity of Relational Query Languages. In *Proc. of STOC'82*, pages 137–146, 1982.

[Var96] M. Y. Vardi. Why is Modal Logic So Robustly Decidable? In *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. American Mathematical Society, 1996.

[Vas98] P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube operations. In *Proc. of the 10th Statistical and Scientific Database Management*, pages 53–62. IEEE, 1998.

[Vas00] P. Vassiliadis. *Data Warehouse Modeling and Quality Issues*. PhD thesis, Dept. of Electrical and Computer Engineering (National Technical University of Athens), 2000.

[VBR03] B. Vrdoljak, M. Banek, and S. Rizzi. Designing Web Warehouses from XML Schemas. In *Proc. of 5th Int. Conf. on Data Warehousing and Knowledge Discovery*, volume 2737 of *Lecture Notes of Computer Science*, pages 89–98. Springer, 2003.

[VS99] P. Vassiliadis and T.K. Sellis. A Survey of Logical Models for OLAP Databases. *SIGMOD Record*, 28(4):64–69, ACM, 1999.

[W3C] W3C. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/ (last access 17/12/2009).

[WS03] R. Winter and B. Strauch. A Method for Demand-Driven Information Requirements Analysis in DW Projects. In *Proc. of 36th Annual Hawaii Int. Conf. on System Sciences*, pages 231–239. IEEE, 2003.

[YP04] X. Yin and T.B. Pedersen. Evaluating XML-Extended OLAP Queries Based on a Physical Algebra. In *Proc. of 7th Int. Workshop on Data Warehousing and OLAP*. ACM, 2004.

[Yu97] E.S.K. Yu. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In *Proc. of the 3rd IEEE Int. Symposium on Requirements Engineering*, pages 226–235. IEEE, 1997.