Available online at www.sciencedirect.com

ScienceDirect

Data & Knowledge Engineering 59 (2006) 435–459

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

# Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation ☆

Matteo Golfarelli [a], Jens Lechtenbörger [b,*], Stefano Rizzi [a], Gottfried Vossen [b]

[a] DEIS, University of Bologna, Italy
[b] Department of Information Systems, University of Muenster, Leonardo-Campus 3, 48149 Muenster, Germany

Received 4 September 2005; accepted 21 September 2005
Available online 18 October 2005

**Abstract**

As several mature implementations of data warehousing systems are fully operational, a crucial role in preserving their up-to-dateness is played by the ability to manage the changes that the data warehouse (DW) schema undergoes over time in response to evolving business requirements. In this paper we propose an approach to schema versioning in DWs, where the designer may decide to undertake some actions on old data aimed at increasing the flexibility in formulating cross-version queries, i.e., queries spanning multiple schema versions. First, we introduce a representation of DW schemata as graphs of simple functional dependencies, and discuss its properties. Then, after defining an algebra of schema graph modification operations aimed at creating new schema versions, we discuss how augmented schemata can be introduced to increase flexibility in cross-version querying. Next, we show how a history of versions for DW schemata is managed and discuss the relationship between the temporal horizon spanned by a query and the schema on which it can consistently be formulated.
© 2005 Elsevier B.V. All rights reserved.

*Keywords:* Data warehousing; Schema versioning; Cross-version querying; Schema augmentation

## 1. Introduction

Data Warehouses (DWs) are databases specialized for business intelligence applications and can be seen as collections of *multidimensional cubes* centered on facts of interest for decisional processes. A cube models a set of *events*, each identified by a set of *dimensions* and described by a set of numerical *measures*. Typically, for each dimension a hierarchy of *properties* expresses interesting aggregation levels. A distinctive feature of DWs is that of storing historical data; hence, a temporal dimension is always present.

Data warehousing systems have been rapidly spreading within the industrial world over the last decade, due to their undeniable contribution to increasing the effectiveness and efficiency of decision making processes within business and scientific domains. This wide diffusion was supported by remarkable research results

☆ A preliminary version of this paper appeared in the Proc. 3rd Workshop on Evolution and Change in Data Management, 2004.
* Corresponding author. Tel.: +49 251 8338158; fax: +49 251 8338159.
*E-mail addresses:* golfare@csr.unibo.it (M. Golfarelli), lechten@wi.uni-muenster.de (J. Lechtenbörger), srizzi@deis.unibo.it (S. Rizzi), vossen@wi.uni-muenster.de (G. Vossen).

0169-023X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2005.09.004

aimed at increasing querying performance [17,33], at improving the quality of data [34], and at refining the design process [14] on the one hand, as well as by the quick advancement of commercial tools on the other.

Today, as several mature implementations of data warehousing systems are fully operational within medium to large contexts, the continuous evolution of the application domains is bringing to the forefront the dynamic aspects related to describing how the information stored in the DW changes over time from two points of view:

- *At the data level*: Though historical values for measures are easily stored due to the presence of temporal dimensions that timestamp the events, the multidimensional model implicitly assumes that the dimensions and the related properties are entirely static. This assumption is clearly unrealistic in most cases; for instance, a company may add new categories of products to its catalog while others can be dropped, or the category of a product may change in response to the marketing policy.
- *At the schema level*: The DW schema may change in response to the evolving business requirements. New properties and measures may become necessary (e.g., a subcategory property could be added to allow more detailed analysis), while others may become obsolete. Even the set of dimensions characterizing a cube may be required to change.

Note that, in comparison with operational databases, temporal issues are more pressing in DWs since queries frequently span long periods of time; thus, it is very common that they are required to cross the boundaries of different versions of data and/or schema. Besides, the criticality of the problem is obviously higher for DWs that have been established for a long time, since unhandled evolutions will determine a stronger gap between the reality and its representation within the database, which will soon become obsolete and useless.

So far, research has mainly addressed changes at the data level, i.e., changes in instances of aggregation hierarchies (the so-called *slowly-changing dimensions* [21]), and some commercial systems already allow to track changes in data and to effectively query cubes based on different temporal scenarios. For instance, SAP-BW [32] allows the user to choose which version of the hierarchies to use while querying (e.g., *aggregate the sales according to the categories that were true on 1/1/2000*). On the other hand, schema versioning in DWs has only partially been explored and no dedicated commercial tools or restructuring methods are available to the designer. Thus, both an extension of tools and support for designers are urgently needed.

Indeed, according to the frequently cited definition of a DW by Inmon [18] one of the characteristic features of a DW is its non-volatility, which means that data is integrated into the DW once and remains unchanged afterwards. Importantly, this feature implies that the re-execution of a single query will always produce a single consistent result. In other words, past analysis results can be verified and then inspected by means of more detailed OLAP sessions at any point in time. While commercial solutions may support non-volatility in the presence of changes at the data level (e.g., SAP-BW under the term "historical truth"), non-volatility in the presence of changes at the schema level has not received much attention. In fact, it is easy to see that the ability to re-execute previous queries in the presence of schema changes requires access to past schema versions.

In this paper we propose an approach to schema versioning in DWs, specifically oriented to support the formulation of *cross-version queries*, i.e., queries spanning multiple schema versions. Our main contributions are the following:

(1) *Schema graphs* are introduced in order to univocally represent DW schemata as graphs of simple functional dependencies, and an algebra of graph operations to determine new versions of a DW schema is defined. Importantly, our graph model captures the core of all multidimensional data models proposed previously.
(2) The issues related to *data migration*, i.e., how to consistently move data between schema versions, are discussed. In particular, a dichotomy in the treatment of fact instances and dimension instances is established.
(3) *Augmented schemata* are introduced in order to increase flexibility in cross-version querying. The augmented schema associated with a version is the most general schema describing the data that are actually recorded for that version and thus are available for querying purposes.
(4) The sequencing of versions to form *schema histories* in presence of augmented schemata is discussed, and the relationship between the temporal horizon spanned by a query and the schema on which it can

consistently be formulated is analyzed. Based on the notion of *schema intersection*, a novel approach towards cross-version querying is defined.

The remainder of this paper is outlined as follows. We discuss related work in Section 1.1, and we give an overview of our approach and present a motivating example in Section 1.2. In Section 2 we propose a graph-based representation of DW schemata. In Section 3 we describe the elementary changes a schema may undergo, while in Section 4 we show how these changes create new versions. In Section 5 we discuss how versions are concatenated into histories. In Section 6 we focus on cross-version querying, and we draw the conclusions in Section 8.

## *1.1. Related work*

### *1.1.1. Temporal databases*

A large part of the literature on schema versioning in relational databases is surveyed in [31]. With reference to terminology introduced in [19] our approach is framed as *schema versioning* since past schema definitions are retained so that all data may be accessed both retrospectively and prospectively through user-definable version interfaces; additionally, with reference to [31] we are dealing with *partial schema versioning* as no retrospective update is allowed to final users. Note that, in contrast, schema *evolution* allows modifications of the schema without loss of data but does not require the maintenance of a schema history.

As to querying in the presence of multiple schema versions, while TSQL2 [35] only allows users to punctually specify the schema version according to which data are queried, queries spanning multiple schema versions are considered in [10,15].

### *1.1.2. Data warehouse evolution and versioning*

In the DW field, a number of approaches for managing slowly-changing dimensions were devised (see for instance [11,25,38]).

As to schema evolution/versioning, mainly five approaches can be found in the literature. First, in [29], the impact of evolution on the quality of the warehousing process is discussed in general terms, and a supporting meta-model is outlined. Second, in [37] a prototype supporting dimension updates at both the data and schema levels is presented, and the impact of evolutions on materialized views is analyzed. Third, in [7,6], an algebra of basic operators to support evolution of the conceptual schema of a DW is proposed, and the effect of each operator on the instances is analyzed. In all these approaches, versioning is not supported and the problem of querying multiple schema versions is not mentioned. Fourth, in [12], the COMET model to support schema evolution is proposed. The paper is mainly focused on the constraints to be fulfilled in order to ensure the integrity of the temporal model; though the problem of queries spanning multiple schema versions is mentioned, the related issues are not explored, and the discussion of how to map instances from one version to another is only outlined. Fifth and finally, [5] proposes an approach to versioning where, besides "real" versions determined by changes in the application domain, also "alternative" versions to be used for what–if analysis are introduced. Even here, cross-version queries are only mentioned.

On the commercial side, the versioning problem has only marginally been addressed. For instance, *SQL Compare* is a tool for comparing and synchronizing SQL Server database schemata, to be used when changes made to the schema of a local database need to be pushed to a central database on a remote server [30]. On the other hand, the *Oracle Change Management Pack* is aimed to report and track the evolving state of meta-data, thus allowing to compare database schemata, and to generate and execute scripts to carry out the changes [1]. In both cases, the possibility of formulating a single query on multiple databases with different schemata is not even mentioned. Recently, the need for improving the handling of dynamic aspects in DWs has been raised by the KALIDO team. Their solution [2], based on an original data organization called *generic data modeling*, offers a set of tools for handling data changes but does not address the versioning problem.

## *1.2. Overview of the approach and example*

In this section we introduce our approach based on a working example. Consider a schema $S_0$ modeling the shipments of parts to customers all over the world. A conceptual schema for the shipment fact is depicted in

Fig. 1(a) using the Dimensional Fact Model (DFM) formalism [13]. Although we chose the DFM formalism among the variety of graphical multidimensional data models to illustrate the sample scenario, the results obtained in this paper are not restricted to that particular model.

The fact shown in Fig. 1(a) has two measures, namely QtyShipped and ShippingCosts, and five dimensions, namely Date, Part, Customer, Deal, and ShipMode. A hierarchy of properties is attached to each dimension; the meaning of each arc is that of a many-to-one association, i.e., a functional dependency.

Suppose now that, at $t_1 = 1/1/2003$, $S_0$ undergoes a major revision aimed at better fulfilling some changing business requirements. In particular, in the new version $S_1$:

(1) The temporal granularity has changed from Date to Month.
(2) A classification into subcategories has been inserted into the part hierarchy.
(3) A new constraint has been modeled in the customer hierarchy, stating that sale districts belong to nations, and that for each customer the nation of her sale district is the nation of her city.
(4) The incentive has become independent of the shipment terms.

Then, at $t_2 = 1/1/2004$, another version $S_2$ is created as follows:

(1) Two new measures ShippingCostsEU and ShippingCostsLIT are added.
(2) The ShipMode dimension is eliminated.

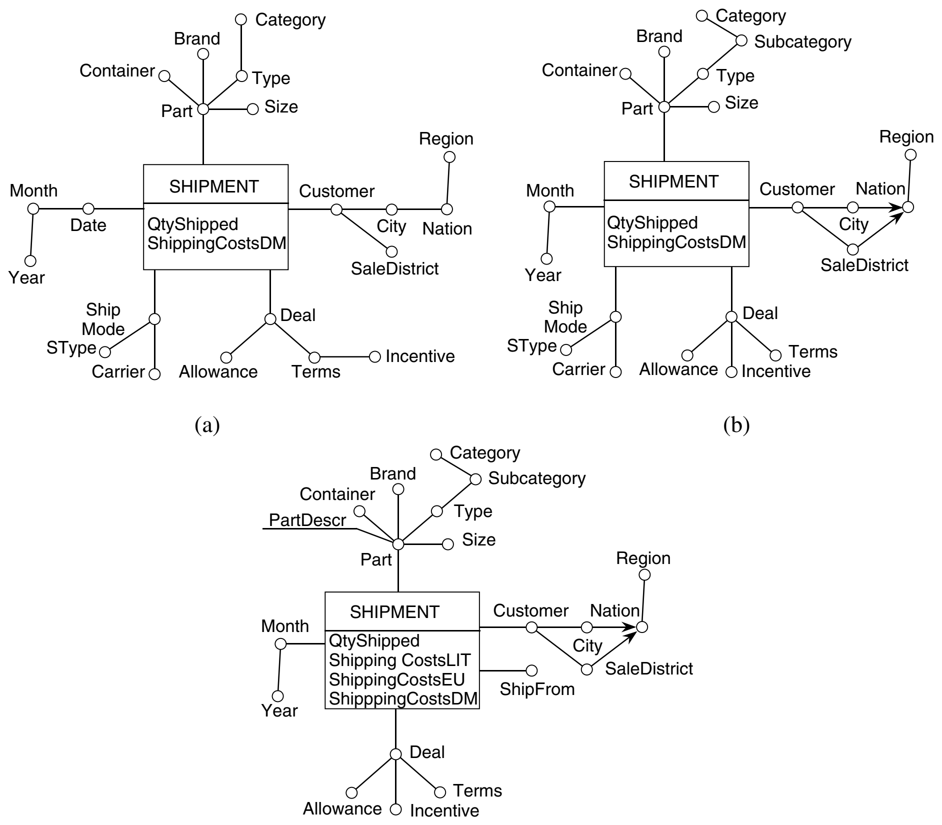


Fig. 1. Conceptual schemata for three versions of the shipment fact: $S_0$ (a), $S_1$ (b), and $S_2$ (c).

(3) A ShipFrom dimension is added.
(4) A descriptive attribute PartDescr is added to Part.

The conceptual schemata for $S_1$ and $S_2$ are depicted in Fig. 1(b) and (c).

Within a system not supporting versioning, at the time of change all data would be migrated to the new schema version. On the other hand, if the system supports versioning, all previous schema versions will still be available for querying together with the data recorded during their validity time. In this case, the user could be given the possibility of deciding which schema version is to be used to query data. For instance, the 2002 data could be queried under schema $S_1$, introduced in 2003; in particular, one might ask for the distribution of the shipping costs for 2002 according to subcategories, introduced in 2003.

In our approach, the schema modifications occurring in the course of DW operation lead to the creation of a *history of schema versions*. All of these versions are available for querying purposes, and the relevant versions for a particular analysis scenario may either be chosen explicitly by the user or implicitly by the query subsystem. The key idea is to support flexible cross-version querying by allowing the designer to enrich previous versions using the knowledge of current schema modifications. For this purpose, when creating a new schema version the designer may choose to create *augmented schemata* that extend previous schema versions to reflect the current schema extension, both at the schema and the instance level.

To be more precise, let $S$ be the current schema version and $S'$ be the new version. Given the differences between $S$ and $S'$, a set of possible *augmentation actions* on past data are proposed to the designer; these actions may entail checking past data for additional constraints or inserting new data based on user feedback. (Details are presented in Section 4.2.) The set of actions the designer decides to undertake leads to defining and populating an augmented schema $S^{\mathrm{AUG}}$, associated with $S$, that will be used instead of $S$, transparently to the final user, to answer queries spanning the validity interval of $S$. Importantly, $S^{\mathrm{AUG}}$ is always an extension of $S$, in the sense that the instance of $S$ can be computed as a projection of $S^{\mathrm{AUG}}$.

Consider, for instance, the schema modification operation that introduces attribute Subcategory, performed at time $t_1 = 1/1/2003$ to produce version $S_1$. For all parts still shipped after $t_1$ (including both parts introduced after $t_1$ and parts already existing before $t_1$), a subcategory will clearly have to be defined as part of data migration, so that queries involving Subcategory can be answered for all shipments from $t_1$ on. However, if the user is interested in achieving cross-version querying on years 2002 and 2003, i.e., if she asks to query even old data (shipments of parts no longer existing at $t_1$) on Subcategory, it is necessary to:

(1) define an augmented schema for $S_0$, denoted $S_0^{\mathrm{AUG}}$, that contains the new attribute Subcategory;
(2) (either physically or virtually) move old data entries from $S_0$ to $S_0^{\mathrm{AUG}}$; and
(3) assign the appropriate values for Subcategory to old data entries in $S_0^{\mathrm{AUG}}$.

This process will allow queries involving Subcategory to be answered on old data via the instance of $S_0^{\mathrm{AUG}}$. Note that, while the first two steps are entirely managed by the versioning system, the last one is the designer's responsibility.

As another example, consider adding the constraint between sale districts and nations. In this case, the designer could ask the system to check if the functional dependency between SaleDistrict and Nation holds on past data too (it might already have been true when $S_0$ was created, but might have been missed at design time; or it could be enforced via data cleansing): if so, the augmented schema $S_0^{\mathrm{AUG}}$ will be enriched with this dependency, which increases the potential for roll-up and drill-down operations during OLAP sessions.

## 2. Formal representation of DW schemata

In this section we define notation and vocabulary, and we recall results on simple FDs that form the basis for the graph-theoretic framework used throughout this paper.

## 2.1. Simple functional dependencies

We assume the reader to be familiar with the basics of relational databases and FDs. Following standard notation (see, e.g., [26]), capital letters from the beginning (respectively ending) of the alphabet denote single (respectively sets of) attributes, "$\equiv$" denotes equivalence of sets of FDs, and $F^+$ is the closure of the FDs in $F$.

An FD $X \to Y$ is *simple* if $|X| = |Y| = 1$ (note that simple FDs have also been called *unary* [9]). Given a set $F$ of simple FDs over $X$ we say that $F$ is *acyclic* (respectively *cyclic*) if the directed graph $(X, F)$ (i.e., the graph that contains the attributes in $X$ as nodes and that contains an arc $(A, B)$ if $A \to B \in F$) is acyclic (respectively cyclic).

Finally, we recall that a set $F$ of FDs is *canonical* if [26]:[1]

- every FD $X \to Y \in F$ satisfies $|Y| = 1$,
- $F$ is left-reduced, i.e., for each FD $X \to A \in F$ there is no $Y \subsetneqq X$ such that $Y \to A \in F^+$, and
- $F$ is nonredundant, i.e., there is no $F' \subsetneqq F$ such that $F' \equiv F$.

For every set $F$ of FDs there is at least one *canonical cover*, i.e., a canonical set $F^0$ of FDs such that $F \equiv F^0$ [26].

## 2.2. Schema graphs

In order to talk about schema versioning, we first have to fix a representation for DW schemata on top of which operations for schema modifications can be defined. In this section, we introduce a graph-based representation for schemata called *schema graph*, which captures the core of multidimensional models such as the DFM. Intuitively, in line with [23], a DW schema is a directed graph, where the nodes are attributes (either properties or measures), and arcs represent simple FDs of a canonical cover. The representation of DW schemata in terms of graphs allows us to define schema modifications by means of four elementary graph manipulations, namely adding and deleting of nodes and arcs, and to analyze the schema versioning problem in a simple and intuitive setting. Besides, as shown in [22], it provides a considerable simplification over hypergraph based approaches that have previously been used to represent schemata involving general FDs (see, e.g., [4]).

Formally, we represent a DW schema in terms of one or more schema graphs.

**Definition 1** (*Schema Graph*). A *schema graph* is a directed graph $S = (\widehat{U}, F)$ with nodes $\widehat{U} = \{E\} \cup U$ and arcs $F$, where

(1) $E$ is called *fact node* and represents a placeholder for the fact itself (meaning that its values are the single events that have occurred, i.e., the single tuples of the fact table);
(2) $U$ is a set of attributes (including properties and measures);
(3) $F$ is a set of simple FDs defined over $\{E\} \cup U$ in the DW schema;
(4) $E$ has only outgoing arcs, and there is a path from $E$ to every attribute in $U$.

$S$ is called *canonical* schema graph if $F$ is canonical.

Canonical schema graphs for the shipment facts in Fig. 1 are shown in Fig. 2.

Throughout this paper we assume that schema graphs satisfy the *universal relation schema assumption* (URSA) [26], which is a standard assumption in relational database design. URSA ties an attribute name to its semantics, i.e., among the set of schema graphs describing the DW schema all occurrences of an attribute name are assumed to have the same meaning. Thus, in our example, the two different concepts "type of part" and "type of ship mode" are represented in terms of two attributes with distinct names (Type and SType); on the other hand, the attributes occurring in the Part hierarchy are allowed to appear with the same names in other schema graphs if those schema graphs deal with the same part concept (which will be the case for other versions of shipment facts or, e.g., invoice or revenue related facts).

[1] Canonical sets are called *minimal* in [36], while the notion of minimality of [26] has a different meaning.

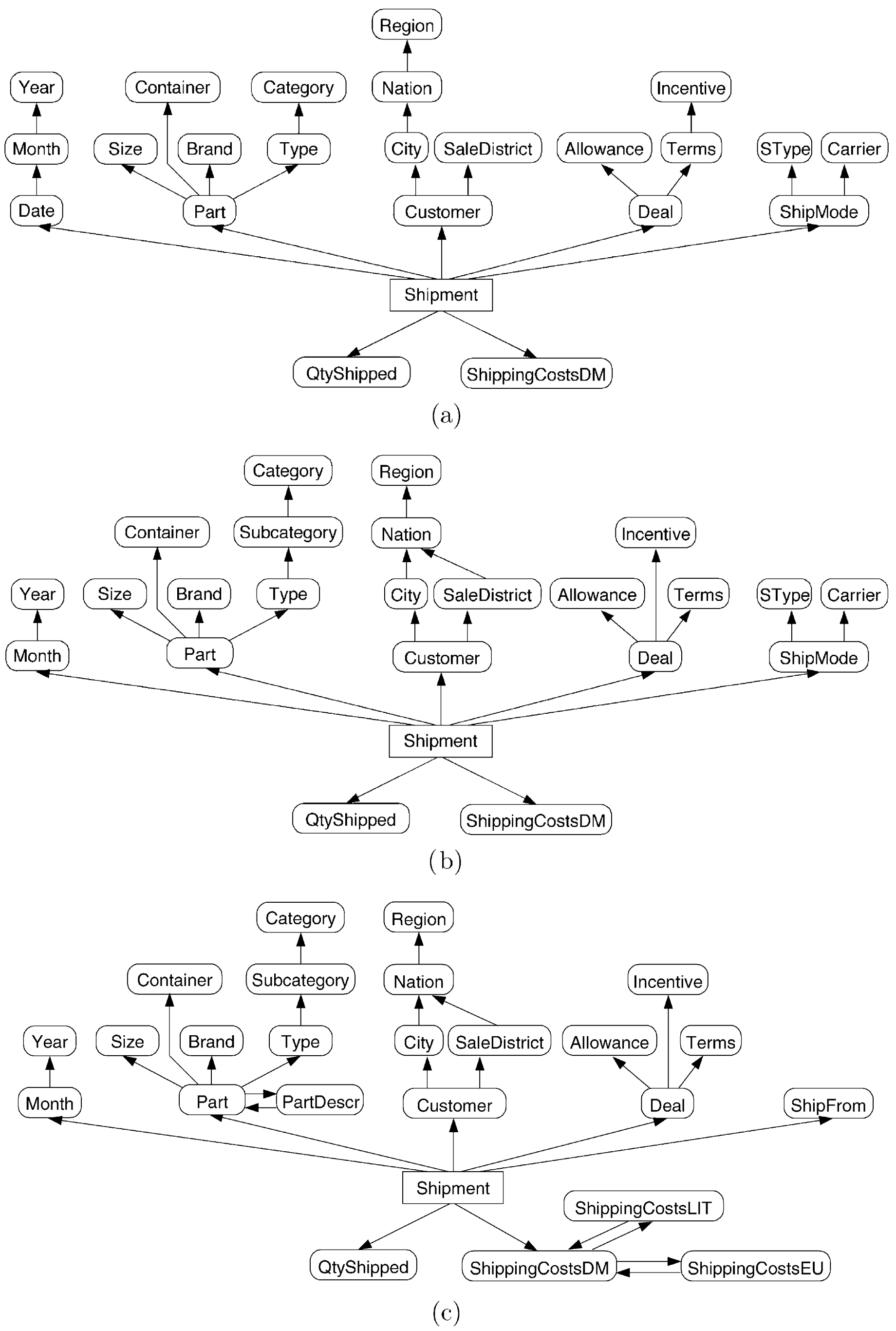


Fig. 2. Schema graphs $S_0$ (a), $S_1$ (b), and $S_2$ (c).

Finally, we note that, with reference to the multidimensional model, an FD $f \in F$ has an impact on the semantics of attributes as follows:

(1) $f = E \rightarrow A$
- $A$ may be a dimension. Since the values of $E$ represent single events, in this case $f$ expresses the fact that each event is related to exactly one value for each dimension.

- $A$ may be a measure. In this case $f$ represents the fact that each event is associated with exactly one value for each measure.

(2) $f = B \rightarrow A$

- $B$ may be a dimension or a property, and $A$ a property. In this case, $B \rightarrow A$ models a many-to-one association within a hierarchy (for instance, $B$ is City and $A$ is Nation). From the implementation point of view, this means that the correspondence between each value of $B$ and exactly one value of $A$ is explicitly stored in the database (for instance, within a dimension table of a star schema).
- $B$ may be a measure, and $A$ a *derived measure*. In this case, $B \rightarrow A$ models the fact that $A$ can be derived from $B$ (for instance, $B$ is the daily fluctuation of a share and $A$ is its absolute value). From the implementation point of view, there are two choices: Either store $A$ as a separate attribute, which introduces redundant data in the DW but may be advantageous if the computation of $A$ from $B$ is expensive; or store a function in the meta-data repository allowing $A$ to be computed from $B$ on-the-fly.

In view of these observations it should be clear that the questions whether an attribute defines (1) a measure or a dimension and (2) a derived measure or a property cannot be answered based on our graph representation. Thus, this additional information is recorded in the meta-data repository (see Section 2.5).

## 2.3. Reduced schema graphs

In comparison with general schema graphs, the class of canonical schema graphs has the important advantage of providing a non-redundant, i.e., compact schema representation. On the other hand, in order to obtain unique results for schema modification operations, we also need to make sure that we are dealing with a *uniquely determined* representation. This section shows how any schema graph can be transformed into a uniquely determined canonical form called *reduced schema graph*.

We begin by observing that, for *acyclic* sets of simple FDs, canonical covers are uniquely determined and can be computed via transitive reduction.

**Definition 2** (*Transitive Closure and Reduction*). Let $U$ be a set of attributes, and $F$ be a set of simple FDs over $U$. The *transitive closure of* $F$, denoted $F^*$, is inductively defined as follows:

(1) $f \in F \Rightarrow f \in F^*$,
(2) $A \rightarrow B \in F^* \wedge B \rightarrow C \in F^* \Rightarrow A \rightarrow C \in F^*$.

A *transitive reduction of* $F$ is a minimal set $F^-$ of simple FDs over $U$ such that $F^* = (F^-)^*$.

To illustrate the differences between transitive closure $F^*$ and closure $F^+$ of FDs $F$, we observe that by definition $F^*$ contains only simple FDs. In contrast, if the FDs in $F$ involve two or more attributes then $F^+$ contains additional non-simple FDs (exponential in the number of attributes) that are implied by $F^*$.

**Example 1.** For $F = \{A \rightarrow B, B \rightarrow C\}$ we have

$$F^* = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\},$$

$$\begin{aligned} F^+ = \{&A \rightarrow B, B \rightarrow C, A \rightarrow C, A \rightarrow A, AB \rightarrow A, AC \rightarrow A, ABC \rightarrow A,\\ &B \rightarrow B, AB \rightarrow B, AC \rightarrow B, BC \rightarrow B, ABC \rightarrow B, C \rightarrow C, AB \rightarrow C, AC \rightarrow C, BC \rightarrow C, ABC \rightarrow C\}. \end{aligned}$$

As shown in [3], in case of acyclic graphs the transitive reduction $F^-$ is uniquely determined and $F^- \subseteq F$. Besides, from [22] we recall that $F^-$ is a canonical cover of $F$. Thus, if $F$ is acyclic, the reduced schema graph for $S = (\widehat{U}, F)$ is $S^- = (\widehat{U}, F^-)$.

On the other hand, in real-world scenarios acyclicity of FDs may not be given as cycles occur in at least the following two cases:

- *Descriptive properties*. Though most associations between properties of DW schemata have many-to-one multiplicity, in some cases they may have one-to-one multiplicity. Typically, this happens when a property is associated with one or more univocal descriptions (e.g., the technical staff may refer to products by their codes while sale agents may use product names).
- *Derived measures*. Two measures may be derivable from each other by applying some fixed formula (for instance, Euros and Italian Liras can be transformed back and forth by applying constant conversion factors).

The following example shows that canonical covers are no longer unique if FDs are cyclic.

**Example 2.** Consider measures $A_1$ and $A_2$ whose values are derivable from each other (such as item prices listed in two currencies), i.e., we have cyclic FDs $A_1 \rightarrow A_2$ and $A_2 \rightarrow A_1$, and a measure $A_3$ that can be derived from $A_1$ and also from $A_2$ (such as an item tax that is computed from the price). Hence, we have a set $F$ of FDs $\{A_1 \rightarrow A_2, A_2 \rightarrow A_1, A_1 \rightarrow A_3, A_2 \rightarrow A_3\}$, and it is easily verified that $\{A_1 \rightarrow A_2, A_2 \rightarrow A_1, A_1 \rightarrow A_3\}$ and $\{A_1 \rightarrow A_2, A_2 \rightarrow A_1, A_2 \rightarrow A_3\}$ are both canonical covers of $F$.

In the remainder of this section we show how a *uniquely determined* reduced form for schema graphs can be determined even in the presence of cyclic FDs [22]. Consider a schema graph $S = (\widehat{U}, F)$, where $F$ is neither necessarily canonical nor acyclic. The relation $\equiv_F$ on $\widehat{U}$ defined by

$$A \equiv_F B \quad \text{iff} \quad A \rightarrow B \in F^+ \wedge B \rightarrow A \in F^+$$

for $A, B \in \widehat{U}$ is an equivalence relation; $\widehat{U}/_{\equiv_F}$ is used to denote the set of equivalence classes induced by $\equiv_F$ on $\widehat{U}$. Then, we consider the acyclic directed graph where each node is one equivalence class $X \in \widehat{U}/_{\equiv_F}$ and an arc goes from $X$ to $Y$, $X \neq Y$, if there are attributes $A \in X$ and $B \in Y$ such that $A \rightarrow B \in F$. The transitive reduction of this graph, called the *equivalent acyclic schema graph* for $S$ and denoted by $S^a = (\widehat{U}/_{\equiv_F}, F^a)$, is acyclic (by construction) and uniquely determined (as transitive reduction is unique for acyclic graphs). Now, let $<_S$ be a total order on $\widehat{U}$ (e.g., user-specified or system-generated based on some sorting criterion such as attribute creation timestamp or attribute name).

**Definition 3** (*Implicit FDs*). Let $\widehat{U}$ be a set of attributes with total order $<_S$, let $F$ be a (possibly redundant and/or cyclic) set of simple FDs over $\widehat{U}$, and let $X = \{A_1, \ldots, A_n\} \in \widehat{U}/_{\equiv_F}$, $n \geqslant 1$. Let $A_1 <_S A_2 <_S \cdots <_S A_n$ be the ordering of attributes in $X$ according to $<_S$. The *implicit FDs for X* (*w.r.t.* $<_S$) are given by $F_X = \{A_1 \rightarrow A_2, A_2 \rightarrow A_3, \ldots, A_{n-1} \rightarrow A_n, A_n \rightarrow A_1\}$.

**Definition 4** (*Reduced Schema Graph*). Let $S = (\widehat{U}, F)$ be a schema graph and $S^a = (\widehat{U}/_{\equiv_F}, F^a)$ be the equivalent acyclic schema graph for $S$. The *reduced schema graph for S* is the directed graph $S^- = (\widehat{U}, F^-)$, where

$$F^- = \bigcup_{X \rightarrow Y \in F^a} \{\min X \rightarrow \min Y\} \cup \bigcup_{X \in \widehat{U}/_{\equiv_F}} F_X.$$

From [3,22] we know that the reduced schema graph $S^- = (\widehat{U}, F^-)$ for $S$ is a uniquely determined transitive reduction of $S$, and $F^-$ is a canonical cover of $F$. In the remainder of this paper, given a set $F$ of simple FDs, we will use "−" to denote the *reduction operator* that produces the uniquely determined canonical cover $F^-$ of $F$ according to Definition 4.

**Example 3.** Consider the schema graph $S_2$ in Fig. 2(c), where Part has an equivalent property PartDescr and shipping costs are expressed in EU, LIT, and DM. The reduced form for $S_2$, based on the total order induced by attribute names, is shown in Fig. 3.

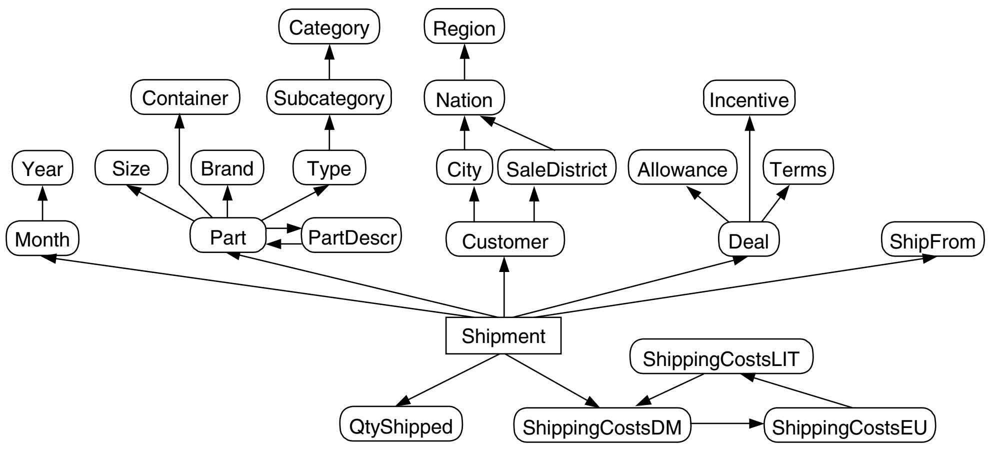


Fig. 3. Reduced form for schema graph $S_2$.

## 2.4. Projection on schema graphs

Given a set $F$ of FDs over attributes $U$ and given $X \subseteq U$, the *projection of F to X* is given by $\pi_X(F) := \{A_1 \to A_2 \in F^+ | A_1A_2 \subseteq X\}$. Based on the results of [23,22], in this section we show that the projection operation $\pi$ is closed on schema graphs. The importance of this result lies in the fact that it allows us to use projection to define precisely the effect of the *deletion* of an attribute from a schema graph (cf. Definition 6 later on). Indeed, when deleting an attribute our aim is to retain "as much information concerning FDs as possible", which is suitably formalized via $\pi$. However, as projection is defined as subset of a closure, the following challenges (which are visible in Example 1 above) arise in applying $\pi$ directly:

(1) In general the projection of a set of simple FDs involves non-simple FDs, which are outside our framework.
(2) The result of the closure may grow exponentially in the number of involved attributes.

In our setting of simple FDs, however, the following results hold.

**Lemma 1** [23]. *Let U be a set of attributes, let F be a set of simple FDs over U, and let $A \in U$ and $X \subseteq U$ such that $X \neq \emptyset$, $A \notin X$, and $X \to A \in F^+$. Then there is a sequence of $n \geqslant 2$ attributes $A_1, \ldots, A_n \in U$ such that $A_1 \in X$, $A_n = A$, and $A_i \to A_{i+1} \in F$, $1 \leqslant i \leqslant n-1$.*

**Lemma 2** [22]. *Let U be a set of attributes, F be a set of simple FDs over U, and $X \subseteq U$. Let $F_X = \pi_X(F)^0$ and $F'_X = \{A_1 \to A_2 \in F^* \mid A_1A_2 \subseteq X\}^-$.*

(1) *$F_X$ contains only simple* FDs.
(2) *$F_X \equiv F'_X$.*

In view of Lemma 2, from now on we assume that, for a set $F$ of simple FDs, projection is defined as $\pi_X(F) := \{A_1 \to A_2 \in F^* | A_1A_2 \subseteq X\}$, which by definition is a simple set of FDs.

**Theorem 1.** *Let $S = (\widehat{U}, F)$ be a schema graph, and let $X \subseteq \widehat{U}$ such that $E \in X$. Then $(X, \{A_1 \to A_2 \in F^* \mid A_1A_2 \subseteq X\}^-)$ is a reduced schema graph.*

**Proof.** Let $F_{X_1} = \pi_X(F) = \{A_1 \to A_2 \in F^* | A_1A_2 \subseteq X\}$ and $F_{X_2} = F^-_{X_1}$. We have to verify that $(X, F_{X_2})$ is a schema graph, i.e., that (i) $F_{X_2}$ is a set of simple FDs, (ii) $E$ has only outgoing arcs in $F_{X_2}$, and (iii) there is a path in $F_{X_2}$ from $E$ to every attribute in $X_2$. Once we have established (i)–(iii), the remaining claims follow as for any set $F_0$ of simple FDs the set $F^-_0$ is by construction a uniquely determined canonical cover. First, (i) follows from the fact that by definition $F_{X_1}$ contains only simple FDs. Hence, by construction $F_{X_2} = F^{-}_{X_1}$ contains only simple FDs as well. Now, concerning (ii) we note that $E$ has only outgoing arcs in $F$ and hence in $F^*$. Moreover, by definition $\pi$ only removes FDs from $F^*$, which implies that $E$ has only outgoing arcs in $F_{X_1}$. As $F_{X_2}$ is a canonical cover of $F_{X_1}$ the claim follows. Concerning (iii), let $A \in X$. We have to show $E \to A \in F^*_{X_2}$, from which the claim follows by Lemma 1. As there is a path from $E$ to $A$ in $F$ we have $E \to A \in F^*$, and by definition of projection we find $E \to A \in F_{X_1}$, which implies $E \to A \in F^*_{X_2}$ and ends the proof. □

## 2.5. Meta-data

The formalism of schema graphs captures just that core of multidimensional schemata which we need as basis to define a powerful schema modification algebra in the next section. Nevertheless, we assume that schema graphs are managed as part of a larger meta-data repository, which contains all kinds of schema information, in particular, information that is not captured in our graph notation.

For example, for all versions of schema graphs defined over time the meta-data repository includes specifications of attribute *domains*, *classifications* of attributes into dimensions, measures, and properties, *summarizability types* or *restriction levels* of measures (cf. [24]), *derivation* specifications for derived measures, and *summarizability constraints* (cf. [13,23]). In particular we assume that, in accordance with what several OLAP tools do, each measure is associated with exactly one aggregation operator for each dimension. Thus, measures are not just attributes: they have some built-in semantics, coded in meta-data, that states how they will be aggregated along each hierarchy.

## 3. Schema modification algebra

Our proposal towards schema versioning rests upon four simple schema modification operations, namely $\texttt{Add}_\texttt{A}()$ to add a new attribute, $\texttt{Del}_\texttt{A}()$ to delete an existing attribute, $\texttt{Add}_\texttt{F}()$ to add an arc involving existing attributes (i.e., an FD), and $\texttt{Del}_\texttt{F}()$ to remove an existing arc. For each of these operations we define its effect on the schema. Note that, from now on, we will always consider schema graphs in their reduced form, and thus use the terms *schema* and *reduced schema graph* interchangeably.

In addition to the four schema modification operations defined below, we assume that there are (1) an operation to create an initial schema $S = (\{E\}, \emptyset)$ that contains only the fact node without attributes or arcs and (2) an operation to delete an existing schema. We do not consider these operations any further.

In order to specify schema modification operations formally, let $S = (\widehat{U}, F)$ be a schema. For each modification operation $M(Z)$ (where $M$ is $\texttt{Add}_\texttt{A}$ or $\texttt{Del}_\texttt{A}$ and $Z$ is an attribute, or $M$ is $\texttt{Add}_\texttt{F}$ or $\texttt{Del}_\texttt{F}$ and $Z$ is an FD), we define the new schema $\texttt{New}(S, M(Z))$ obtained when applying $M$ on current schema $S$.

**Definition 5.** Let $S = (\widehat{U}, F)$ be a reduced schema graph, and let $A$ be an attribute. Then we have

$$\texttt{New}(S, \texttt{Add}_\texttt{A}(A)) := (\widehat{U} \cup \{A\}, (F \cup \{E \to A\})^{-}).$$

We call attention to the fact that in Definition 5 we do not distinguish the cases whether $A$ does already occur in $S = (\widehat{U}, F)$ or not. Indeed, on the one hand Definition 5 implies that $S$ remains unchanged if $A$ does already occur in $S$. On the other, if $A$ is a newly inserted attribute then it is directly connected by an arc to the fact node $E$.[2] Besides, when adding an attribute, the designer will be required to specify whether it is a measure or a property; this information will be recorded in meta-data.

**Definition 6.** Let $S = (\widehat{U}, F)$ be a reduced schema graph, and let $A \in U$ be an attribute. Then we have

$$\texttt{New}(S, \texttt{Del}_\texttt{A}(A)) := (\widehat{U} \setminus \{A\}, \pi_{\widehat{U} \setminus \{A\}}(F)^{-}).$$

[2] The unrestricted usage of $\texttt{Add}_\texttt{A}$ might introduce homonym conflicts, i.e., the designer could try to add an attribute although another attribute with the same name but a different meaning occurs somewhere else. To avoid such conflicts, in an implementation the meta-data repository needs be checked whether some schema version contains an attribute with the same name.

Thus, in view of Theorem 1 the deletion of attribute $A$ is defined by removing $A$ and retaining FDs not involving $A$ via projection.

**Definition 7.** Let $S = (\widehat{U}, F)$ be a reduced schema graph, and let $f = A_1 \to A_2$ be an FD involving attributes in $U$. Then we have

$$\texttt{New}(S, \texttt{Add}_\texttt{F}(f)) := (\widehat{U}, (F \cup \{f\})^-).$$

We note that the insertion of a new FD may introduce redundancies, which are removed via reduction "–".

**Definition 8.** Let $S = (\widehat{U}, F)$ be a reduced schema graph, and let $f = A_1 \to A_2$ be an existing FD in $F$, where $A_1 \neq E$. Let

$$F' = F \setminus \{f\} \cup \{A_0 \to A_2 \mid (\exists A_0 \in \widehat{U}) A_0 \to A_1 \in F\} \cup \{A_1 \to A_3 \mid (\exists A_3 \in U) A_2 \to A_3 \in F\}.$$

Then we have

$$\texttt{New}(S, \texttt{Del}_\texttt{F}(f)) := (\widehat{U}, (F')^-).$$

The intuition underlying Definition 8 is as follows: First, the specified FD $f = A_1 \to A_2$ gets deleted via set difference. Then, previous transitive dependencies are retained by adding (i) FDs to $A_2$ from all nodes $A_0$ determining $A_1$ (possibly $A_0 = E$) and (ii) FDs from $A_1$ to all nodes $A_3$ determined by $A_2$.

We note that the "appropriate" deletion of FDs is more intricate than it might seem at first sight. For example, we cannot simplify the deletion of an FD $f$ by defining the new set of FDs to be $(F^+ \setminus \{f\})^0$ (recall that $F^0$ denotes the canonical cover of $F$). Indeed, given $A_1 \to A_2 \in F$ we have $A_1X \to A_2 \in F^+$ for an arbitrary set $X$. Although $A_1X \to A_2$ is not left-reduced with respect to $F$, it may be left-reduced with respect to $F^+ \setminus \{f\}$, which implies that $(F^+ \setminus \{f\})^0$ is not guaranteed to be a set of simple FDs. Consider for instance $F = \{E \to A_1, E \to A_2, A_2 \to A_3\}$ and $\texttt{Del}_\texttt{F}(A_2 \to A_3)$: here, we have $A_1A_2 \to A_3 \in F^+$, and $A_1A_2 \to A_3$ is also contained in $(F^+ \setminus \{A_2 \to A_3\})^0$, which does not correspond to the users' intuition. Moreover, we observe that if we tried to define deletion of an FD via $(F^* \setminus \{f\})^-$ (to get rid of non-simple FDs), we were still facing a severe problem: Users would be unable to delete a single FD from a cycle involving three or more attributes, as the FD to be removed would be implied by the remaining FDs. Thus, users would be unable to pull out attributes from cycles involving three or more attributes. With our definition, none of these problems arises.

**Example 4.** The sequence of operations applied in order to arrive at $S_1$ starting from $S_0$ is

$\texttt{Del}_\texttt{A}(\textsf{Date})$,
$\texttt{Add}_\texttt{A}(\textsf{Subcategory}), \texttt{Add}_\texttt{F}(\textsf{Type} \to \textsf{Subcategory}), \texttt{Add}_\texttt{F}(\textsf{Subcategory} \to \textsf{Category})$,
$\texttt{Add}_\texttt{F}(\textsf{SaleDistrict} \to \textsf{Nation})$,
$\texttt{Del}_\texttt{F}(\textsf{Terms} \to \textsf{Incentive})$

In particular, the resulting reduced schema graphs for the part hierarchy after each operation in line 2 are shown in Fig. 4. The sequence of operations applied in order to arrive at $S_2$ starting from $S_1$ is

$\texttt{Add}_\texttt{A}(\textsf{PartDescr}), \texttt{Add}_\texttt{F}(\textsf{Part} \to \textsf{PartDescr}), \texttt{Add}_\texttt{F}(\textsf{PartDescr} \to \textsf{Part})$,
$\texttt{Add}_\texttt{A}(\textsf{ShipFrom})$,
$\texttt{Del}_\texttt{A}(\textsf{SType}), \texttt{Del}_\texttt{A}(\textsf{Carrier}), \texttt{Del}_\texttt{A}(\textsf{ShipMode})$,
$\texttt{Add}_\texttt{A}(\textsf{ShippingCostsEU}), \texttt{Add}_\texttt{A}(\textsf{ShippingCostsLIT})$,
$\texttt{Add}_\texttt{F}(\textsf{ShippingCostsDM} \to \textsf{ShippingCostsEU})$,
$\texttt{Add}_\texttt{F}(\textsf{ShippingCostsEU} \to \textsf{ShippingCostsDM})$,
$\texttt{Add}_\texttt{F}(\textsf{ShippingCostsDM} \to \textsf{ShippingCostsLIT})$,
$\texttt{Add}_\texttt{F}(\textsf{ShippingCostsLIT} \to \textsf{ShippingCostsDM})$.

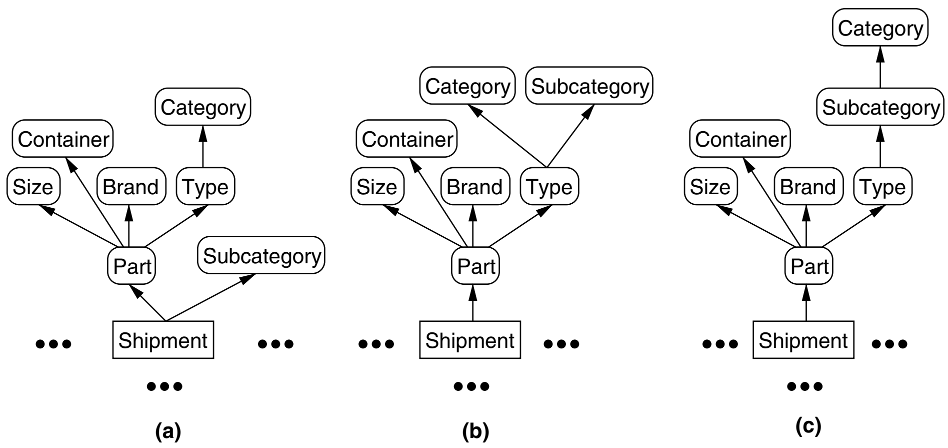


Fig. 4. The schema graph for the shipment fact after (a) including new attribute Subcategory, (b) including new arc Type → Subcategory, and (c) including new arc Subcategory → Category.

We are now in the position to show that Definitions 5–8 formalize the expected behavior of schema modifications, in particular that the modification operations are closed on schema graphs.

**Theorem 2.** *Let* $S = (\widehat{U}, F)$ *be a reduced schema graph.*

(1) *Let* $(\widehat{U}', F') = \mathtt{New}(S, \mathtt{Add}_{\mathtt{A}}(A))$. *Then* $A \in \widehat{U}'$.
(2) *Let* $(\widehat{U}', F') = \mathtt{New}(S, \mathtt{Del}_{\mathtt{A}}(A))$. *Then* $A \notin \widehat{U}'$.
(3) *Let* $(\widehat{U}', F') = \mathtt{New}(S, \mathtt{Add}_{\mathtt{F}}(f))$. *Then* $f \in F'^{+}$.
(4) *Let* $(\widehat{U}', F') = \mathtt{New}(S, \mathtt{Del}_{\mathtt{F}}(f))$. *Then* $f \notin F'^{+}$.

Additionally, in all of the above cases, $(\widehat{U}', F')$ is a reduced schema graph.

**Proof.** Statements (1)–(3) follow immediately from Definitions 5–7, respectively. Statement (4) follows from Definition 8, observing that $F$ is canonical and hence in particular non-redundant. It remains to show that $(\widehat{U}', F')$ is a reduced schema graph, i.e., that (a) $F'$ is a set of simple FDs, (b) $E$ has only outgoing arcs and there is a path from $E$ to every attribute in $\widehat{U}'$, and (c) $(\widehat{U}', F')$ is in reduced form. For $\mathtt{Add}_{\mathtt{A}}$, $\mathtt{Add}_{\mathtt{F}}$, and $\mathtt{Del}_{\mathtt{F}}$, (a) and (b) are immediate. For $\mathtt{Del}_{\mathtt{A}}$, (a) and (b) follow from Theorem 1. For all four operations, (c) follows immediately from the definition of the reduction operator "−". □

Intuitively, Theorem 2 states that our schema modification operations preserve valid schema graphs and are guaranteed to produce non-redundant and uniquely determined results.

## 4. Versions

We call a *version* a schema that reflects the business requirements during a given time interval, called its *validity*, that starts upon schema creation time and extends until the next version is created.[3] The validity of the current version, created at time $t$, is $[t, +\infty]$. A version is populated with the events occurring during its validity and can be queried by the user.

A new version is the result of a sequence of modification operations, which we call *schema modification transaction*, or simply *transaction*. In analogy to the usual transaction concept, intermediate results obtained

[3] In accordance with [27] we argue that there is no need to distinguish valid time from transaction time in the context of schema versioning.

after applying single schema modifications are invisible for querying purposes. Moreover, intermediate schemata are neither populated with events, nor are they associated with augmented schemata.

A transaction produces (1) a new version and (2) an augmented schema for each previous version, all of which are (either physically or virtually) populated with data. In Section 4.1 we address (1), in particular we discuss how the new version is populated. In Section 4.2 we address (2), i.e., we illustrate how augmented schemata are created at the end of transactions in order to increase flexibility in cross-version querying.

## 4.1. Data migration

Given version $S$, let the sequence of operations $M_1(Z_1),\ldots,M_h(Z_h)$ be the executed schema modification transaction. Then, the new version $S'$ is defined by executing the modification operations one after another, i.e., $S' = \mathtt{New}(S_h, M_h(Z_h))$, where $S_1 = S$ and $S_i = \mathtt{New}(S_{i-1}, M_{i-1}(Z_{i-1}))$ for $i = 2,\ldots,h$.

In order to populate $S'$, it is necessary to carry out some *migration actions* to consistently move data from the previous version to the new one. If $S'$ is created at time $t$, migration actions involve *the data that are valid at t*, i.e., data whose validity spans both $S$ and $S'$. Importantly, in the DW context the validity may be defined differently for two categories of data, namely events (in the star schema, tuples of fact tables) and instances of hierarchies (tuples of dimension tables):

- Events occur at a particular moment in time. Consistently with [20], we assume that the validity of an event that occurs at time $t_e$ is the zero-length interval $[t_e, t_e]$; thus, an event naturally conforms to exactly one version (the one valid at $t_e$). Consequently, when a new version $S'$ is created, while all future events will necessarily conform to $S'$, no data migration will be required for past events. (Note that this is a consequence of the fact that, in a *versioning* approach, all past versions are retained together with their data. Conversely, in the approach to schema evolution in DWs described in [7], migration is carried out also for events: in fact, in evolution past versions are not retained.)
- Instances of hierarchies are generally valid during time intervals (e.g., a part is valid from the time it is first shipped to the time it is declared obsolete), so their validity may span different versions. Thus, if $S'$ is created at time $t$, for each hierarchy instance that is valid at $t$ it may be necessary to migrate it from $S$ to $S'$.

In order to determine the migration actions to be performed after a transaction, independently of the specific sequence of modification operations that implements the transaction, we define the *net effect* for attributes and FDs with reference to the transaction. Let $S' = (\widehat{U}', F')$ be the new version obtained by applying the transaction to version $S = (\widehat{U}, F)$. Then we define

$$\mathtt{Diff}^+_{\mathtt{A}}(S,S') := U' \setminus U \text{ (set of added attributes)}$$
$$\mathtt{Diff}^+_{\mathtt{F}}(S,S') := F' \setminus F^* \text{ (set of added FDs)}$$
$$\mathtt{Diff}^-_{\mathtt{A}}(S,S') := U \setminus U' \text{ (set of deleted attributes)}$$

We note that it is not necessary to define $\mathtt{Diff}^-_{\mathtt{F}}(S,S')$ (set of deleted FDs) since no action needs to be performed for deleted FDs. In fact, if an FD gets deleted then a constraint on previous instances gets removed; thus, there is no need for alignment of previous instances with the new schema version.

The migration actions associated with the elements in $\mathtt{Diff}^+_{\mathtt{A}}(S,S')$, $\mathtt{Diff}^-_{\mathtt{A}}(S,S')$, and $\mathtt{Diff}^+_{\mathtt{F}}(S,S')$ are reported in Table 1 and defined in the following. All of these actions are supported and managed by the versioning system, under the designer's guidance where required.

Table 1
Migration actions associated to added or deleted attributes/FDs

| Element | Condition | Migration action |
|---|---|---|
| $A \in \mathtt{Diff}^+_{\mathtt{A}}(S,S')$ | $(E \to A) \notin F'$ $A$ is property | Add values for $A$ |
| $A \in \mathtt{Diff}^-_{\mathtt{A}}(S,S')$ | $(E \to A) \notin F'$ $A$ is property | delete $A$ |
| $f \in \mathtt{Diff}^+_{\mathtt{F}}(S,S')$ | – | enforce $f$ |

(1) *Add values for A:* A new property $A$ has been added to a hierarchy. Each valid hierarchy instance is migrated to $S'$ by providing values for $A$. For instance, when property Subcategory is added, then all currently valid subcategory values must be recorded under $S'$.
(2) *Delete A:* Property $A$ has been deleted, so its values are simply dropped from all valid instances migrated to $S'$.
(3) *Enforce f:* A new FD $f$ has been added. We distinguish two cases: (1) If the new FD involves two already existing properties, it is necessary to check if $f$ holds for all valid instances and possibly to enforce it in $S'$ by modifying, under the user guidance, the values of one or both properties involved in $f$. (2) If the new FD involves a new property $A$ (i.e., one that has just been added), then the values for $A$ must be provided in such a way that this FD is satisfied. For instance, when Subcategory is added to the part hierarchy as in Fig. 4(c), then each valid part type in $S_2$ must be associated with exactly one subcategory and all types included in each subcategory must belong to the same category.

In all cases not covered by the table, no action needs to be performed. In fact, the impact of adding or deleting a measure or a dimension is restricted to events, which are not involved in migration as explained above; finally, deleting an FD requires no change on hierarchy instances.

Again, we emphasize that in a versioning approach all past versions must be retained. Thus, for each migrated hierarchy instance, both versions of data will be available for querying: the one conforming to schema $S$, used for accessing the events that occurred before time $t$, and the one conforming to $S'$, used for accessing the events that occurred after $t$.

**Example 5.** In the shipment example, at time $t_1 = 1/1/2003$ the new version $S_1$ is created from version $S_0$. Among other things, this transaction entails adding Subcategory to the part hierarchy (consistently with the FDs Type $\rightarrow$ Subcategory and Subcategory $\rightarrow$ Category) and adding an FD from SaleDistrict to Nation. Fig. 5 shows the data in the dimension tables for customers and parts just before $t_1$ and some months after $t_1$. After $t_1$ each dimension table has two copies, belonging to $S_0$ and $S_1$ respectively. Fields from and to express the validity of parts (for instance, they may have been introduced to manage parts as a slowly-changing dimension), while customers are assumed to be always valid. At time $t_1$, all the valid parts (those whose to timestamp is open) are migrated to $S_1$ and their subcategory is added; then, in February, part ♯3 is dismissed

S0.DT_CUST (before and after $t_1$)

| idCust | Customer | SaleDistrict | City | Nation | … |
|---|---|---|---|---|---|
| 1 | Bianchi | MidEurope | Milan | Italy | … |
| 2 | Rossi | SouthEurope | Rome | Italy | … |
| 3 | Schmidt | MidEurope | Lienz | Austria | … |
| 4 | Bauer | MidEurope | Innsbruck | Austria | … |

S0.DT_PARTS (before $t_1$)

| idPart | Part | Type | Category | from | to | … |
|---|---|---|---|---|---|---|
| 1 | VSX-D514 | Amplifier | Electronics | 1-2001 | 9-2002 | … |
| 2 | VSX-D914 | Amplifier | Electronics | 6-2002 | - | … |
| 3 | TZ-MC05 | Speaker | Electronics | 3-2001 | - | … |
| 4 | PDP-5045 | TV | Electronics | 7-2002 | - | … |
| 5 | PDP-4345 | TV | Electronics | 2-2000 | 1-2001 | … |

S0.DT_PARTS (after $t_1$)

| idPart | Part | Type | Category | from | to | … |
|---|---|---|---|---|---|---|
| 1 | VSX-D514 | Amplifier | Electronics | 1-2001 | 9-2002 | … |
| 2 | VSX-D914 | Amplifier | Electronics | 6-2002 | 12-2002 | … |
| 3 | TZ-MC05 | Speaker | Electronics | 3-2001 | 12-2002 | … |
| 4 | PDP-5045 | TV | Electronics | 7-2002 | 12-2002 | … |
| 5 | PDP-4345 | TV | Electronics | 2-2000 | 1-2001 | … |

S1.DT_PARTS (after $t_1$)

| idPart | Part | Type | Subcat. | Category | from | to | … |
|---|---|---|---|---|---|---|---|
| 2 | VSX-D914 | Amplifier | HiFi | Electronics | 1-2003 | - | … |
| 3 | TZ-MC05 | Speaker | HiFi | Electronics | 1-2003 | 2-2003 | … |
| 4 | PDP-5045 | TV | Video | Electronics | 1-2003 | - | … |
| 6 | DV-275 | DVDPlayer | Video | Electronics | 4-2003 | - | … |

S1.DT_CUST (after $t_1$)

| idCust | Customer | SaleDistrict | City | Nation | … |
|---|---|---|---|---|---|
| 1 | Bianchi | SouthEurope | Milan | Italy | … |
| 2 | Rossi | SouthEurope | Rome | Italy | … |
| 3 | Schmidt | MidEurope | Lienz | Austria | … |
| 4 | Bauer | MidEurope | Innsbruck | Austria | … |

Fig. 5. Dimension tables for customers and parts in the shipment example, before and after time $t_1$.

and, in April, a new part (♯6) is inserted. As to customers, all of them are migrated. Since the new FD SaleDistrict → Nation does not hold for MidEurope, the sale district of customer ♯1 is modified in $S_1$. Note that, at $t_1$, the open timestamps of parts are closed: in fact, from a conceptual point of view, it is clear that instances cannot survive their schema.

## 4.2. Augmentation

Let $S' = (\widehat{U}', F')$ be the new version obtained by applying, at time $t$, a transaction to version $S = (\widehat{U}, F)$. Let $S_i$ be a previous version (possibly, $S_i$ coincides with $S$ itself), and $S_i^{\mathrm{AUG}}$ be its augmented schema. While migration involves the data whose validity spans both $S$ and $S'$, the *augmentation* of $S_i$—which determines the new augmented schema for $S_i$—involves the data whose validity overlaps with the validity of $S_i$. Thus, while migration is only performed on hierarchy instances that are valid at time $t$, augmentation also concerns the events that occurred before $t$ and the hierarchy instances that were no more valid at $t$.

As in migration, also here the possible actions do not depend on the specific sequence of modification operators applied, but on the net effect for attributes and FDs with reference to the transaction. More specifically:

- If attribute $A$ has been added in the course of the transaction, i.e., if $A \in \mathtt{Diff}_{\mathtt{A}}^{+}(S, S')$, then the designer may want to include $A$ in the augmented schema for $S_i$ to enable cross-version queries involving $A$.
- If $f$ is a new FD, i.e., if $f = A \rightarrow B \in \mathtt{Diff}_{\mathtt{F}}^{+}(S, S')$, then the designer may want to include $f$ in the augmented schema for $S_i$ to enable cross-version roll-up and drill-down operations involving $A$ and $B$.

In this respect, we call attention to the fact that we only consider augmentations in response to operations that *add* attributes or FDs: in fact, the utility of augmenting the current version with deleted attributes/FDs seems highly questionable.[4] On the one hand, we do not see why a designer should first delete an attribute only to state that data associated with this attribute should be maintained in an augmented schema. On the other, we assume that deletions of FDs are triggered by real-world events that invalidate these FDs (the deletion of valid FDs reduces the information of a schema without need, which does not seem reasonable); hence, there is no way, not even in an augmented schema, to maintain these FDs.

Now, we define the augmentation operation $\mathtt{Aug}(S_i^{\mathrm{AUG}}, S, S')$ that (further) augments the augmented schema $S_i^{\mathrm{AUG}} = (\widehat{U}_i, F_i)$ based on the designer's choice in response to the schema change from $S$ to $S'$. Let $\widehat{\mathtt{Diff}}_{\mathtt{A}}(S, S')$ and $\widehat{\mathtt{Diff}}_{\mathtt{F}}(S, S')$ be the subsets of $\mathtt{Diff}_{\mathtt{A}}^{+}(S, S')$ and $\mathtt{Diff}_{\mathtt{F}}^{+}(S, S')$, respectively, including only the attributes and FDs the designer has chosen to augment. We note that all attributes occurring in FDs of $\widehat{\mathtt{Diff}}_{\mathtt{F}}(S, S')$ must be contained in $\widehat{U}_i \cup \widehat{\mathtt{Diff}}_{\mathtt{A}}(S, S')$, as only those FDs can be augmented whose attributes occur in the augmented schema.[5] Then the *new augmented version for* $S_i$ is defined as follows:

$$\mathtt{Aug}(S_i^{\mathrm{AUG}}, S, S') := \left(\widehat{U}_i \cup \widehat{\mathtt{Diff}}_{\mathtt{A}}(S, S'), \left(F_i \cup \pi_{\widehat{U}_i \cup \widehat{\mathtt{Diff}}_{\mathtt{A}}(S, S')}(\widehat{\mathtt{Diff}}_{\mathtt{F}}(S, S'))\right)^{-}\right).$$

The augmentation actions associated with each element in $\widehat{\mathtt{Diff}}_{\mathtt{A}}(S, S')$ and $\widehat{\mathtt{Diff}}_{\mathtt{F}}(S, S')$ are reported in Table 2 and defined as follows:

(1) *Estimate values for A*: A new measure $A$ has been added. To enable cross-version querying, the designer provides values for $A$ for the events recorded under $S_i$, typically by deriving an estimate based on the values of the other measures. For instance, when measure Discount is added to the shipment fact, if the discount applied depends on the shipped quantity bracket, its values for past events may be easily estimated from measure Qty shipped.

[4] Designers can, of course, delete attributes and FDs from schema versions. The point is that such deletions do not lead to augmentations.

[5] If an attribute $A$ occurs in an FD to be augmented but not in $\widehat{U}_i \cup \widehat{\mathtt{Diff}}_{\mathtt{A}}(S, S')$ then the system should warn the designer that she probably forgot to augment $A$. In this case, the designer needs to revise her choices concerning potential actions accordingly.

Table 2
Augmentation actions associated to added attributes/FDs

| Element | Condition | | Augm. action |
|---|---|---|---|
| $A \in \widehat{\mathtt{Diff}}_{\mathtt{A}}(S, S')$ | $(E \to A) \in F_i$ | $A$ is measure | Estimate values for $A$ |
| | | $A$ is dimension | Disaggregate measure values |
| | $(E \to A) \notin F_i$ | $A$ is derived measure | Compute values for $A$ |
| | | $A$ is property | Add values for $A$ |
| $f \in \widehat{\mathtt{Diff}}_{\mathtt{F}}(S, S')$ | – | | Check if $f$ holds |

(2) *Disaggregate measure values*: A new dimension $A$ has been added. To enable cross-version querying, the designer must disaggregate past events by $A$ according to some business rule or by adopting a statistical interpolation approach that exploits multiple summary tables to deduce the correlation between measures [28]. For instance, a likely reason for adding dimension ShipFrom is that, while in the past all shipments were made from the same warehouse $w$, now they are occasionally made from other warehouses: in this case, all past events can be easily related to $w$.

(3) *Compute values for A*: A derived measure $A$ has been added; by definition of derived measure, the values of $A$ for past events are computed by applying some known computation to another measure.

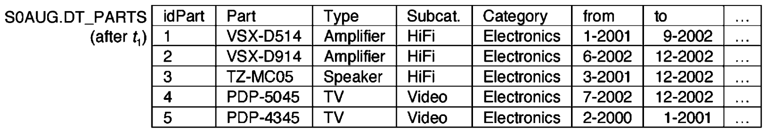
(4) *Add values for A*: A new property $A$ has been added, so the designer may provide values for $A$. For instance, when Subcategory is added, the designer may provide values for all subcategories.

(5) *Check if f holds*: A new FD $f$ has been added. As for migration, we distinguish two cases. (1) If $f$ involves two already existing attributes, it is necessary to check whether $f$, that was added for $S'$, also holds for $S_i$; this can be automatically done by inspecting the instance of $S_i$. For instance, when SaleDistrict $\to$ Nation is added, the system checks that no sale district including customers from different nations exists. If the check fails, the designer is warned: $f$ cannot be augmented since this would require to change the reality as recorded in the past. (2) If $f$ involves a new attribute $A$ (i.e., one that has just been added), then the values for $A$ must be provided in such a way that $f$ is satisfied.

**Example 6.** In the shipment example, initially we start from version $S_0$ where we have $S_0^{\mathrm{AUG}} = S_0$. When new version $S_1$ is created from current version $S_0$, we have

$$\mathtt{Diff}^{+}_{\mathtt{A}}(S_0, S_1) = \{\mathsf{Subcategory}\}$$
$$\mathtt{Diff}^{+}_{\mathtt{F}}(S_0, S_1) = \{\mathsf{SaleDistrict} \to \mathsf{Nation}, \mathsf{Type} \to \mathsf{Subcategory}, \mathsf{Subcategory} \to \mathsf{Category}\}$$

Thus, the actions the designer can undertake on data valid during $S_0$ to augment $S_0^{\mathrm{AUG}}$ are (1) to provide values for Subcategory and (2) to let the system check if SaleDistrict $\to$ Nation holds on $S_0$. Assuming the designer decides to undertake action (1), she has the additional choice to augment any subset of the FDs in $\mathtt{Diff}^{+}_{\mathtt{F}}(S_0, S_1)$ involving Subcategory, and this additional choice is actually materialized by assigning values to Subcategory consistently with these FDs. As to (2), we note that the FD SaleDistrict $\to$ Nation cannot be augmented since, in Example 5, it does not hold for $S_0$. Then we have $\widehat{\mathtt{Diff}}_{\mathtt{A}}(S, S') = \mathtt{Diff}^{+}_{\mathtt{A}}$ and $\widehat{\mathtt{Diff}}_{\mathtt{F}}(S, S') = \{\mathsf{Type} \to \mathsf{Subcategory}, \mathsf{Subcategory} \to \mathsf{Category}\}$, which determines the new augmented schema for $S_0$, $S_0^{\mathrm{AUG}} = \mathtt{Aug}(S_0, S_0, S_1)$. Note that $S_0^{\mathrm{AUG}} \neq S_1$ since it does not include SaleDistrict $\to$ Nation while it also includes attributes and FDs that have been deleted on the way to $S_1$ (e.g., Date and Terms $\to$ Incentive appear in $S_0^{\mathrm{AUG}}$ but not in $S_1$). Fig. 6 shows, with reference to Example 5, the instances of the dimension table for parts within the new augmented schema for $S_0$. Differently from the migration case, here all the parts that have been valid under $S_0$ are augmented by specifying their subcategory.


S0AUG.DT_PARTS (after $t_1$)

| idPart | Part | Type | Subcat. | Category | from | to | ... |
|---|---|---|---|---|---|---|---|
| 1 | VSX-D514 | Amplifier | HiFi | Electronics | 1-2001 | 9-2002 | ... |
| 2 | VSX-D914 | Amplifier | HiFi | Electronics | 6-2002 | 12-2002 | ... |
| 3 | TZ-MC05 | Speaker | HiFi | Electronics | 3-2001 | 12-2002 | ... |
| 4 | PDP-5045 | TV | Video | Electronics | 7-2002 | 12-2002 | ... |
| 5 | PDP-4345 | TV | Video | Electronics | 2-2000 | 1-2001 | ... |

Fig. 6. Augmented dimension table for parts in the shipment example.

## 5. Versioning

In this section we consider schema versioning based on sequences of versions as defined in the previous section, which influence the history of schemata and augmented schemata. Formally, a *history* is a sequence $H$ of one or more triples representing versions of the form ($S$, $S^{\mathrm{AUG}}$, $t$), where $S$ is a version, $S^{\mathrm{AUG}}$ is the related augmented schema, and $t$ is the start of the validity interval of $S$:

$$H = ((S_0, S_0^{\mathrm{AUG}}, t_0), \ldots, (S_n, S_n^{\mathrm{AUG}}, t_n)),$$

where $n \geqslant 0$ and $t_{i-1} < t_i$ for $1 \leqslant i \leqslant n$. Note that, in every history, for the last triple $(S_n, S_n^{\mathrm{AUG}}, t_n)$ we have $S_n^{\mathrm{AUG}} = S_n$ as augmentation only enriches *previous* versions using knowledge of the current modifications.

Given version $S_0$ created at time $t_0$, the initial history is

$$H = ((S_0, S_0^{\mathrm{AUG}}, t_0)),$$

where $S_0^{\mathrm{AUG}} = S_0$. Schema modifications then change histories as follows. Let $H = ((S_0, S_0^{\mathrm{AUG}}, t_0), \ldots, (S_{n-1}, S_{n-1}^{\mathrm{AUG}}, t_{n-1}), (S_n, S_n^{\mathrm{AUG}}, t_n))$ be a history, and let $S_{n+1}$ be the new version at time $t_{n+1} > t_n$; then the resulting history $H'$ is

$$H' = ((S_0, \mathtt{Aug}(S_0^{\mathrm{AUG}}, S_n, S_{n+1}), t_0), \ldots, (S_n, \mathtt{Aug}(S_n^{\mathrm{AUG}}, S_n, S_{n+1}), t_n), (S_{n+1}, S_{n+1}^{\mathrm{AUG}}, t_{n+1})),$$

where $S_{n+1}^{\mathrm{AUG}} := S_{n+1}$.

We point out that a schema modification might potentially change any or all augmented schemata contained in the history. E.g., adding a new FD at time $n + 1$, which has been valid but unknown throughout the history, may lead to a “back propagation” of this FD into every augmented schema in the history. Moreover, note that new augmentations of previous schemata are based on the *augmented* schemata as recorded in the history, not on the schemata themselves. Thus, augmentations resulting from different modifications are accumulated over time, resulting in augmented schemata whose information content—and, hence, potential for answering queries—is growing monotonically with every modification.

We close this section by observing that, assuming to rely on a relational DBMS, a relevant choice concerns how to physically implement augmented schemata and histories. In the literature two approaches are proposed: namely *single-pool* and *multi-pool* [15]. In a single-pool implementation all schemata are associated with a unique, shared, extensional repository, so that the same objects cannot have different values for the same properties when “viewed” through different schemata. On the other hand, in a multi-pool implementation each schema is associated with a “private” extensional data pool; different data pools may contain the same objects having (possibly) completely independent representations and evolutions. Although the multi-pool solution may look more flexible at a first glance, a single-pool solution has usually been considered satisfactory for implementation as it limits the storage space overhead due to coexistence of multiple schemata. Both solutions do support our approach; however, a detailed comparison is outside the scope of this paper.

## 6. Querying across versions

In this section we discuss how our approach to versioning supports cross-version queries, i.e., queries whose temporal horizon spans multiple versions.

Preliminarily, we remark that OLAP sessions in DWs are aimed at effectively supporting decisional processes, thus they are characterized by high dynamics and interactivity. A session consists of a sequence of queries, where each query $q$ is transformed into the next one $q'$ by applying an OLAP operator. For instance, starting from a query asking for the total quantity of parts of each type shipped on each month, the user could be interested in analyzing in more detail a specific type: thus, she could apply a drill-down operator to retrieve the total quantity of each part of that type shipped on each month. Then, she could apply the roll-up operator to measure how many items of each part were shipped on the different years in order to catch a glimpse of the trend. Hence, since OLAP operators mainly navigate the FDs expressed by the hierarchies in the multidimensional schema, specifying the version for query formulation in the OLAP context does not only mean declaring which attributes are available for formulating the next query $q'$, but also representing the FDs among attributes in order to determine how $q'$ can be obtained from the previous query $q$.

In this sense, the *formulation context* for an OLAP query is well represented by a schema graph. If the OLAP session spans a single version, the schema graph is the associated one. Conversely, when multiple versions are involved, a schema under which *all* data involved can be queried uniformly must be determined. In our approach, such a schema is univocally determined by the temporal interval $T$ covered by the data to be analyzed, as the largest schema that retains its validity throughout $T$. In particular, since $T$ may span different versions, we define an *intersection* operator, denoted by $\otimes$, for determining the common schema between two different versions.

**Definition 9.** Let $S = (\{E\} \cup U, F)$ and $S' = (\{E\} \cup U', F')$. Then the *intersection of $S$ and $S'$*, denoted by $S \otimes S'$, is the schema defined as $S \otimes S' = (\{E\} \cup (U \cap U'), (F^* \cap F'^*)^-)$.

Intuitively, the intersection between two versions $S$ and $S'$ is the schema under which data recorded under $S$ or $S'$ can be queried uniformly. In fact, it includes only the attributes belonging to both $S$ and $S'$, as well as their common FDs. An example of intersection is reported in Fig. 7.

We next show that the intersection operator is closed for schema graphs and that it is commutative and associative, which allows us to apply the operator to *sets* of schema graphs.

**Theorem 3**

(1) *Let $S$ and $S'$ be schema graphs. Then $S \otimes S'$ is a schema graph.*
(2) *Operator $\otimes$ is commutative and associative.*

**Proof**

(1) We have to show that there is a path from $E$ to every attribute in the intersection. Let $A \in (U \cap U') \setminus \{E\}$. As $S$ and $S'$ are schema graphs, there are paths from $E$ to $A$ in $S$ and $S'$, i.e., $E \to A \in F^*$ and $E \to A \in F'^*$. Thus, $E \to A \in (F^* \cap F'^*)$, i.e., there is a path from $E$ to $A$ in $(F^* \cap F'^*)$. Then, by construction, there is a path from $E$ to $A$ in $(F^* \cap F'^*)^-$, which is what we had to show.
(2) The fact that schema graph intersection is commutative follows immediately from commutativity of set intersection. To verify that schema graph intersection is associative we have to check $(S_1 \otimes S_2) \otimes S_3 = S_1 \otimes (S_2 \otimes S_3)$, i.e.,

$$(\{E\} \cup ((U_1 \cap U_2) \cap U_3), (((F_1^* \cap F_2^*)^-)^* \cap F_3^*)^-) = (\{E\} \cup (U_1 \cap (U_2 \cap U_3)), (F_1^* \cap ((F_2^* \cap F_3^*)^-)^*)^-)$$

Clearly, equality in the first component (i.e., attributes) follows from associativity of set intersection. Concerning equality in the second component (i.e., FDs) we establish the following facts for all sets $F$, $F_1$, and $F_2$ of simple FDs: (a) $(F^-)^* = F^*$ and (b) $F_1^* \cap F_2^* = (F_1^* \cap F_2^*)^*$

Afterwards equality in the second component is derived as follows for $F_1, F_2, F_3$:

$$\begin{aligned}(((F_1^* \cap F_2^*)^-)^* \cap F_3^*)^- &\overset{(a)}{=} ((F_1^* \cap F_2^*)^* \cap F_3^*)^- \overset{(b)}{=} ((F_1^* \cap F_2^*) \cap F_3^*)^- = (F_1^* \cap (F_2^* \cap F_3^*))^- \\ &\overset{(b)}{=} (F_1^* \cap (F_2^* \cap F_3^*)^*)^- \overset{(a)}{=} (F_1^* \cap ((F_2^* \cap F_3^*)^-)^*)^-\end{aligned}$$

Fact (a) follows immediately from the observation that path reachability remains invariant under transitive reduction (Definition 2) and schema graph reduction (Definition 4). Concerning fact (b) we have to show $f \in F_1^* \cap F_2^*$ for $f \in (F_1^* \cap F_2^*)^*$. (The other inclusion is trivial.) Let $A_1 \to A_n \in (F_1^* \cap F_2^*)^*$. Due to the inductive definition of "*" we either have $A_1 \to A_n \in (F_1^* \cap F_2^*)$, in which case there is nothing to show, or there are $A_1, \ldots, A_n$ for $n \geqslant 3$ such that $A_i \to A_{i+1} \in (F_1^* \cap F_2^*)$, $1 \leqslant i \leqslant n-1$. In the latter case we have

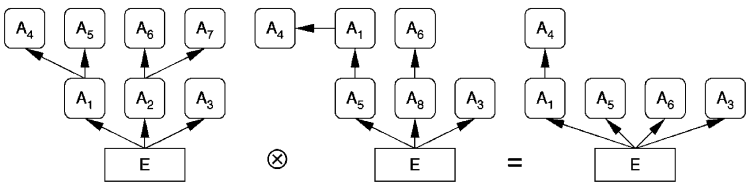


Fig. 7. Intersection between two schema graphs.

$A_i \rightarrow A_{i+1} \in F_1^*$ and $A_i \rightarrow A_{i+1} \in F_2^*$, $1 \leqslant i \leqslant n-1$. Thus, we have $A_1 \rightarrow A_n \in (F_j^*)^* = F_j^*$, $j = 1,2$. Consequently, we have $A_1 \rightarrow A_n \in (F_1^* \cap F_2^*)$, which concludes the proof. □

In view of Theorem 3 a common schema for cross-version querying can now be defined as follows based on schema intersection.

**Definition 10.** Given a history $H$ and a (not necessarily connected) temporal interval $T$, we call the *span* of $T$ on $H$ the set

$$\texttt{Span}(H,T) = \{S_i^{\text{AUG}} | (S_i, S_i^{\text{AUG}}, t_i) \in H \wedge [t_i, t_{i+1}[ \cap T \neq \emptyset\}$$

(conventionally assuming $t_{n+1} = +\infty$).

Given a history $H$ and a temporal interval $T$, the *common schema* on $H$ along $T$ is defined as $\texttt{Com}(H,T) = \otimes_{\texttt{Span}(H,T)} S_i^{\text{AUG}}$.

Let $q$ be the last query formulated, and $T$ be the interval determined by the predicates in $q$ on the temporal hierarchy (if no predicate is present then $T = ]-\infty, +\infty[$). The formulation context for the next query $q'$ is expressed by the schema graph $\texttt{Com}(H, T)$. Note that the OLAP operator applied to transform $q$ into $q'$ may entail changing $T$ into a new interval $T'$; in this case, the formulation context for getting a new query $q''$ from $q'$ will be defined by $\texttt{Com}(H, T')$.

**Example 7.** Let $H = ((S_0, S_0^{\text{AUG}}, t_0), (S_1, S_1^{\text{AUG}}, t_1), (S_2, S_2^{\text{AUG}}, t_2))$ be the history for the shipment fact (recall that we have $t_1 = 1/1/2003$ and $t_2 = 1/1/2004$), and let $q =$ "*Compute the total quantity of each part category shipped from each warehouse to each customer nation since July* 2002". The temporal interval of $q$ is $T = [7/1/2002, +\infty[$, hence $\texttt{Span}(H, T) = \{S_0, S_1, S_2\}$. Fig. 8 shows the formulation context, defined by $S_0^{\text{AUG}} \otimes S_1^{\text{AUG}} \otimes S_2^{\text{AUG}}$, in two situations: when no augmentation has been made, and when all possible augmentations have been made.

First of all, we observe that $q$ is well-formulated only if ShipFrom has been augmented for both previous versions, since otherwise one of the required attributes does not belong to the formulation context.

Then we observe that, for instance, (1) drilling down from Category to Subcategory will be possible only if subcategories and their relationship with categories have been established also for 2002 data; (2) drilling down from Nation to SaleDistrict will be possible only if the FD from sale districts to nations has been verified to hold also before 2003, which is the assumption underlying Fig. 8.

Finally, we note that if ShippingCostsEU is augmented (which is particularly simple due to the existence of constant conversion factors) then queries involving ShippingCostsEU can be evaluated over any time period, although shipping costs were recorded exclusively in DM until $t_2$. Moreover, we point out that the augmentation of the derived measure ShippingCostsEU can be implemented at virtually no storage cost (e.g., in terms of a view that applies the constant conversion factor).

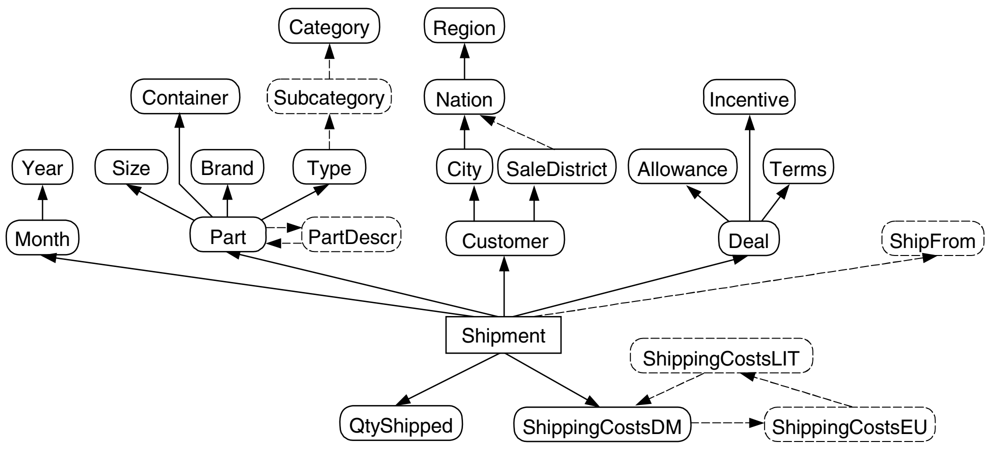


Fig. 8. Formulation contexts for the query in Example 7 without augmentation (in plain lines) and with augmentation (in plain and dashed lines).

We close this section with some remarks about query rewriting over versions. Due to addition or removal of dimensions, the granularity of a given measure $M$ (meant as the combination of dimensions under which its values are recorded) may change from one version to the other. Thus, a given cross-version query $q$ may require aggregation on some (finer) version $S'$, and no aggregation on another (coarser) one $S''$. Importantly, the aggregation operator used to summarize $M$ on $S'$ is the one associated with $M$ in the meta-data repository. If at some time, during augmentation, the designer decides to disaggregate $M$, it is obviously necessary that this is done consistently with this aggregation operator, i.e., in such a way that reaggregating $M$ at the original granularity exactly returns the original values.

Disaggregation also brings to the foreground some subtle summarizability issues. We recall from [16] that aggregate functions are either *distributive* or *algebraic* or *holistic*. Repeated applications of a distributive aggregate function (e.g., SUM) during a sequence of roll-up operations lead to the correct final result. If an aggregate function is not distributive, i.e., if it is algebraic (e.g., AVG) or holistic (e.g., Median), then in general the repeated application during a sequence of roll-up operations leads to an incorrect result. Now, consider an initial schema version $S$ where some measure is recorded under a non-distributive aggregate function, e.g., average quantities in stock (measure Stock) are recorded per product and month. Next, in the new version $S'$ these quantities are recorded at a finer granularity, e.g., per product at the end of each day. If the designer chooses to augment data associated with $S$ to reflect this change, then according to Table 2 measure Stock has to be disaggregated to obtain daily values. Here, daily values can be obtained easily by just using the original monthly measure as new daily measure for each day of the month, as shown in Table 3. Clearly, in this way the averages of the daily measures per month yield the original monthly measures. However, it is less clear how a roll-up aggregation to the year level should be computed. In fact, as the first half of a year (i.e., 01-Jan until 30-Jun) has 181 days while the second one (i.e., 01-Jul until 31-Dec) has 184, the aggregation of daily measures to level year will yield a different result ($\frac{181\cdot 10+184\cdot 20}{365} \approx 15.04$) than the aggregation of monthly measures to level year ($\frac{6\cdot 10+6\cdot 20}{12} = 15$). Following our above argumentation concerning non-distributive aggregate functions, one might think that taking averages of daily measures to level year (with result 15.04) should be correct, while averaging the average monthly values (with result 15) should be avoided. Nevertheless, we recall that the monthly values are the ones that were really recorded in the system, while the daily ones were obtained by disaggregation during augmentation. In particular, on schema version $S$ users might already have seen the value 15 computed from monthly values. As augmentation should not introduce inconsistencies by changing the results of queries, we must make sure that in this particular scenario the daily values are *not* used to compute the aggregates per year.

We envisage the following two alternative approaches to deal with inconsistencies arising from roll-up operations involving disaggregated measures. First, the query subsystem could simply warn the user whenever disaggregated measures have contributed to a query result. Second, and more ideally, the query subsystem could try to avoid the use of disaggregated measures in roll-up aggregations. For example, in the above scenario roll-up operations beyond the month level would always be computed from monthly measures but not from daily ones. However, a systematic analysis of how to avoid the use of disaggregated measures in general and of disaggregated and derived ones in particular still needs to be done.

Table 3
Summarizability issues with disaggregation

| Monthly | | | Daily | | |
|---|---|---|---|---|---|
| Product | Month | Stock | Product | Day | Stock |
| p1 | Jan-2003 | 10 | p1 | 01-Jan-2003 | 10 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| p1 | Jun-2003 | 10 | p1 | 30-Jun-2003 | 10 |
| p1 | Jul-2003 | 20 | p1 | 01-Jul-2003 | 20 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| p1 | Dec-2003 | 20 | p1 | 31-Dec-2003 | 20 |

## 7. Computational complexity

To support our approach towards schema versioning, some central operations have to be implemented. The aim of this section is to indicate that all of these operations can be implemented efficiently (i.e., by means of polynomial time algorithms), which suggests that our approach can indeed be used in an *interactive* design and versioning process. We point out that we do *not* try to derive exact complexity bounds; instead, we just point out some well-known polynomial time algorithms that could be used in an ad-hoc implementation. (In particular, we do not address possibly more efficient incremental algorithms.)

It should not come as a surprise that the complexity of computing transitive reductions underlies the following analysis. Consequently, we recall from [3] that each graph has a transitive reduction that can be computed in polynomial time and, moreover, that the complexity of computing transitive reductions is of the same complexity as computing transitive closures (which can be computed in $O(n^3)$ time using Warshall's algorithm where $n$ is the number of the graph's nodes).

(1) *Schema graph reduction*. To compute the reduced schema graph $S^-$ of $S$ the following procedure can be applied: (a) The strongly connected components of $S$ are identified in linear time (see., e.g., [8]). (b) The equivalent acyclic schema graph $S^a = (\widehat{U}/_{\equiv_F}, F^a)$ for $S$ is constructed from the strongly connected components of $S$ in polynomial time via transitive reduction. (c) The reduced schema graph $S^- = (\widehat{U}, F^-)$ is computed from $S^a = (\widehat{U}/_{\equiv_F}, F^a)$ in polynomial time according to Definition 4.
(2) *Projection*. To compute a projection $\pi_X(F)$ according to Lemma 2 a subset of a transitive closure has to be selected, which can be done in polynomial time.
(3) *Schema modification operations*. Operation $\texttt{Add}_\texttt{A}$ requires adding the attribute to the current schema, which is done in constant time. Operation $\texttt{Del}_\texttt{A}$ can be seen as a composition of projection followed by reduction, both of which are polynomial time as we have seen already. Operations $\texttt{Add}_\texttt{F}$ and $\texttt{Del}_\texttt{F}$ each require a single reduction, which can be done in polynomial time.
(4) *Schema augmentation*. Augmentation requires the computation of $\texttt{Diff}^+_\texttt{A}$ and $\texttt{Diff}^+_\texttt{F}$, which involve standard (polynomial time) set operations and transitive closures. Afterwards, augmentation proceeds in terms of (polynomial time) modification operations on augmented schema versions.
(5) *Schema intersection*. Schema intersection $\otimes$ is defined in terms of a composition of set operations, transitive closure, and reduction; hence, it can be computed in polynomial time as well.

## 8. Conclusions and further remarks

In this paper we have presented an approach towards DW schema versioning. Importantly, our approach relies on a conceptually simple graph model that captures the core of state-of-the-art data models for DWs. Based on the standard graph operations of transitive closure and reduction, we have defined four intuitively appealing schema modification operations in the context of graphical DW schemata. We have shown how single schema modifications lead to a history of versions that contain augmented schemata in addition to "ordinary" schemata, and we have defined an intersection operator that allows us to determine whether a given query, possibly spanning several versions, can be answered based on the information contained in augmented schemata.

As a side remark, we note that our approach can also be used for horizontal benchmarking, where a company compares its own performance against competitors' performance. E.g., in our sample scenario assume that there is another company shipping similar products but according to a different geographical classification. To include the competitor's shipments for horizontal benchmarking, all one has to do is to (a) create a new schema version, (b) add the competitor's geographical classification, e.g., in terms of a new alternative aggregation path such as Customer → CompetitorSaleDistrict → CompetitorRegion, and (c) augment previous schema versions with the new aggregation path.

Appropriate schema versioning and in particular schema augmentation involves considerable manual work by DW administrators. Nevertheless, we are not aware of "simpler" viable alternatives. E.g., one could be tempted to try (a) to express augmented information in terms of views over new data and (b) to answer cross-version queries based on established techniques for answering queries using views. However, this approach is infeasible for the following reason: Consider the insertion of a new dimensional attribute such as Subcategory at time $t_1$ into the Part hierarchy in our sample scenario. To enable cross-version queries involving Subcategory on schema versions prior to $t_1$, somehow Subcategory values have to be "found" for Part instances prior to $t_1$, *even* for parts that do *not* exist any longer after $t_1$. Clearly, those parts do not occur in versions after $t_1$ and hence they do neither occur in views over versions after $t_1$. Consequently, a manual assignment of subcategories to parts cannot be avoided if cross-version queries need to be supported. (Note that an automated assignment of default or null values would not increase the analysis potential.)

Next, we observe that our approach to versioning can be regarded as "pure" since data *never* gets deleted. In particular, if an attribute $A$ is deleted by $\mathtt{Del}_\mathtt{A}$ then a new version is created whose schema does not contain $A$ any longer; hence, $A$-values will not be stored for *new* data any longer. However, previous versions and augmented schemata still contain $A$, and their associated instances contain values for $A$, which allows us to answer queries involving $A$ on old data.

While this fully complies with the idea of versioning [19,31], for practical purposes designers might want to delete data physically to free disk space. This could be easily achieved in at least two ways. First, designers might want to delete entire (old) versions and their associated data, which can be implemented by deleting appropriate data entries and/or tables of those schemata from the database. Second, designers might want to erase individual attributes from the history of versions (as if the attributes never existed). In this case, all tables (belonging to versions and augmented schemata for those versions) containing the specified attribute have to be altered to drop those attributes.

Finally, we remark that our approach to DW schema versioning provides an initial step that covers the majority of schema elements occurring in real world data warehouses. However, future research is necessary to include side concepts such as (1) *cross-dimensional attributes* (e.g., an attribute VAT that is functionally determined by the pair of attributes product and location is a cross-dimensional attribute) and (2) derived measures that can be computed from *sets* of other measures. Indeed, both concepts go beyond our assumption of simple FDs and require general FDs that could be represented by using a schema *hyper*-graph, where hyperarcs go from *sets* of attributes to single attributes. The generalization of our approach to the setting of hypergraphs remains an open issue.

## Acknowledgements

The authors would like to thank the reviewers whose careful reading and constructive remarks helped to improve the presentation.

## References

[1] Oracle Change Management Pack. Technical report, Oracle Technical White Paper, 2000.

[2] KALIDO dynamic information warehouse—a technical overview. Technical report, KALIDO White Paper, 2004.

[3] A.V. Aho, M.R. Garey, J.D. Ullman, The transitive reduction of a directed graph, SIAM Journal on Computing 1 (2) (1972) 131–137.

[4] G. Ausiello, A. D'Atri, D. Saccà, Graph algorithms for functional dependency manipulation, Journal of the ACM 30 (4) (1983) 752–766.

[5] B. Bębel, J. Eder, C. Koncilia, T. Morzy, R. Wrembel, Creation and management of versions in multiversion data warehouse, in: Proc. SAC, Nicosia, Cyprus, 2004, pp. 717–723.

[6] M. Blaschka, FIESTA—a framework for schema evolution in multidimensional databases, Ph.D. thesis, Technische Universitat Munchen, Germany, 2000.

[7] M. Blaschka, C. Sapia, G. Höfling, On schema evolution in multidimensional databases, in: Proc. DaWaK, 1999, pp. 153–164.

[8] T.H. Cormen, C. Stein, R.L. Rivest, C.E. Leiserson, Introduction to Algorithms, McGraw-Hill, 2001.

[9] S.S. Cosmadakis, P.C. Kanellakis, Functional and inclusion dependencies: a graph theoretic approach, in: Proc. 3rd PODS, 1984, pp. 29–37.

[10] C. De Castro, F. Grandi, M.R. Scalas, On schema versioning in temporal databases, in: S. Clifford, A. Tuzhilin (Eds.), Recent Advances in Temporal Databases, Zurich, Switzerland, 1995, pp. 272–294.

[11] J. Eder, C. Koncilia, Changes of dimension data in temporal data warehouses, in: Proc. DaWaK, 2001, pp. 284–293.

[12] J. Eder, C. Koncilia, T. Morzy, The COMET metamodel for temporal data warehouses, in: Proc. CAiSE, 2002, pp. 83–99.

[13] M. Golfarelli, D. Maio, S. Rizzi, The Dimensional Fact Model: a conceptual model for data warehouses, International Journal of Cooperative Information Systems 7 (2–3) (1998) 215–247.

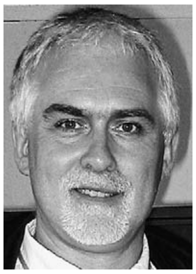
[14] M. Golfarelli, S. Rizzi, A methodological framework for data warehouse design, in: Proc. 1st DOLAP Workshop, Washington, 1998, pp. 3–9.

[15] F. Grandi. A relational multi-schema data model and query language for full support of schema versioning, in: Proc. 10th SEBD, Isola d'Elba, Italy, 2002.
[16] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, Data Cube: a relational aggregation operator generalizing group-by, cross-tab, and sub totals, Data Mining and Knowledge Discovery 1 (1) (1997) 29–53.
[17] H. Gupta, V. Harinarayan, A. Rajaraman, J.D. Ullman, Index selection for OLAP, in: Proc. ICDE, 1997, pp. 208–219.
[18] W.H. Inmon, Building the Data Warehouse, third ed., John Wiley & Sons, 2002.
[19] C. Jensen, J. Clifford, R. Elmasri, S.K. Gadia, P.J. Hayes, S. Jajodia, A consensus glossary of temporal database concept, ACM SIGMOD Record 23 (1) (1994) 52–64.
[20] C. Jensen et al., The consensus glossary of temporal database concepts, in: O. Etzion, S. Jajodia, S. Sripada (Eds.), Temporal Databases: Research and Practice, Springer, 1998, pp. 367–405.
[21] R. Kimball, L. Reeves, M. Ross, W. Thornthwaite, The Data Warehouse Lifecycle Toolkit, John Wiley & Sons, 1998.
[22] J. Lechtenbörger, Computing unique canonical covers for simple FDS via transitive reduction, Information Processing Letters 92 (4) (2004) 169–174.
[23] J. Lechtenbörger, G. Vossen, Multidimensional normal forms for data warehouse design, Information Systems 28 (5) (2003) 415–434.
[24] H. Lenz, A. Shoshani, Summarizability in OLAP and statistical databases, in: Proc. 9th SSDBM, 1997, pp. 132–143.
[25] C. Letz, E. Henn, G. Vossen, Consistency in data warehouse dimensions, in: Proc. IDEAS, 2002, pp. 224–232.
[26] D. Maier, The Theory of Relational Databases, Computer Science Press, 1983.
[27] E. McKenzie, R. Snodgrass, Schema evolution and the relational algebra, Information Systems 15 (2) (1990) 207–232.
[28] E. Pourabbas, A. Shoshani, Answering joint queries from multiple aggregate OLAP databases, in: Proc. 5th DaWaK, Prague, 2003.
[29] C. Quix, Repository support for data warehouse evolution, in: Proc. DMDW, 1999.
[30] Red-Gate Software, Compare and synchronize SQL database structures, 2004.
[31] J. Roddick, A survey of schema versioning issues for database systems, Information and Software Technology 37 (7) (1995) 383–393.
[32] SAP. Multi-dimensional modeling with BW: ASAP for BW accelerator, Technical report, SAP America Inc. and SAP AG, 2000.
[33] D. Theodoratos, M. Bouzeghoub, A general framework for the view selection problem for data warehouse design and evolution, in: Proc. DOLAP, McLean, 2000.
[34] D. Theodoratos, M. Bouzeghoub, Data currency quality satisfaction in the design of a data warehouse, International Journal of Cooperative Information Systems 10 (3) (2001) 299–326.
[35] TSQL2 Language Design Committee, The TSQL2 Temporal Query Language, Kluwer Academic Publishers, 1995.
[36] J.D. UllmanDatabase and Knowledge-Base Systems, vol. 1, Computer Science Press, 1988.
[37] A. Vaisman, A. Mendelzon, W. Ruaro, S. Cymerman, Supporting dimension updates in an OLAP server, in: Proc. CAiSE, 2002, pp. 67–82.
[38] J. Yang, Temporal data warehousing, Ph.D. thesis, Stanford University, 2001.

**Matteo Golfarelli** received his degree in Computer Science in 1995 and his Ph.D. for his work on autonomous agents in 1998 from the University of Bologna, Italy. In 2000 he joined the Computer Science Department as a researcher. Since 2005 he is Associate Professor at the University of Bologna, teaching Information Systems and Database Systems. He participated in several international and national research projects, including the EU PANDA thematic network concerning pattern-base management systems. He has published over 40 papers in refereed journals and international conferences in the fields of data warehousing, pattern recognition, mobile robotics, multi-agent systems. He served in the PC of several international conferences and as a reviewer in journals. His current research interests include all the aspects related to Business Intelligence and Data Warehousing, in particular multidimensional modeling, what–if analysis and Business Performance Management.

**Jens Lechtenbörger** obtained his Ph.D. in computer science from the University of Münster, Germany, with a thesis on "Data Warehouse Schema Design," for which he received a dissertation award at the German database conference BTW 2003. Currently, he is a lecturer at the Department of Information Systems at the University of Münster. His publications appear in major international conferences and journals, for which he also serves as reviewer. His current research interest include all aspects of database systems, in particular data warehouse modeling.

**Stefano Rizzi** received his Ph.D. for his work on autonomous agents in 1996 from the University of Bologna, Italy. Currently he is Full Professor at the University of Bologna, where is the head of the Data Warehousing Laboratory and teaches Advanced Information Systems and Software Engineering. He has published about 70 papers in refereed journals and international conferences, mainly in the fields of data warehousing, pattern recognition, mobile robotics, and multi-agent systems. He joined several national research projects on the above areas and has been involved in the EU PANDA thematic network concerning pattern-base management systems. He currently is scientific supervisor of the data warehousing project of the University of Bologna. His current research interests include data warehouse design, advanced architectures for business intelligence, ontology visualization, and bioinformatics.



**Gottfried Vossen** is a Professor of Computer Science at the Department of Information Systems, University of Muenster in Germany. He received his master's and Ph.D. degrees as well as the German habilitation in 1981, 1986, and 1990, resp., all from the Technical University of Aachen in Germany. He has held visiting positions at the University of California in San Diego, at several German universities including the recently founded Hasso-Plattner-Institute for Software Systems Engineering in Potsdam near Berlin, at Karlstad University in Sweden and at The University of Waikato in Hamilton, New Zealand. His research interests include conceptual as well as application-oriented problems concerning databases, information systems, XML, and workflow management, where a current emphasis is on data warehouses, meta query languages, home information systems, process management, and ontology design for applications related to the semantic web. He has been member in numerous program committees of international conferences and workshops. He is an author or co-author of more than 100 publications, and an author, co-author, or co-editor of 20 books on databases, business process modelling, the Web, e-commerce, and computer architecture.