Schema Versioning in Data Warehouses

Matteo Golfarelli[1], Jens Lechtenbörger[2], Stefano Rizzi[1], and Gottfried Vossen[2]

[1] DEIS, University of Bologna, Italy
[2] Dept. of Information Systems, University of Muenster, Germany

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. After introducing an algebra of DW schema operations, we define a history of versions for data warehouse schemata and discuss the relationship between the temporal horizon spanned by a query and the schema on which it can consistently be formulated.

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* expressing interesting aggregation levels is defined. 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 increase the effectiveness and efficiency of decision making processes within business and scientific domains. 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 extensional 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, which is clearly unrealistic.
- *At the intensional level*: The DW schema may change in response to the evolving business requirements: new properties and measures may become necessary, while others may become obsolete.

S. Wang et al. (Eds.): ER Workshops 2004, LNCS 3289, pp. 415–428, 2004.

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, that will soon become obsolete and useless.

So far, research and DW vendors have mainly addressed changes at the extensional level (see [1, 2] for instance); schema versioning has only partially been explored and no dedicated commercial tools or restructuring methodologies are available to the designer. Thus, both an extension of tools and a support to designers are urgently needed. 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. The main contributions are:

1. *Schema graphs* are introduced in order to univocally represent DW schemata as graphs (Section 2), and an algebra of graphs operations to determine new versions of a DW schema is defined (Section 3.1).
2. *Augmented schemata* are introduced in order to increase flexibility in cross-version querying (Section 3.2). 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.
3. The sequencing of versions to form *schema histories* in presence of augmented schemata is discussed (Section 3.3), and the relationship between the temporal horizon spanned by a query and the schema on which it can consistently be formulated is analyzed (Section 4).

1.1 Approach Overview and Motivating 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 DFM formalism [3]. The fact has two measures, QtyShipped and ShippingCostsDM, 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. In the new version S_1: (a) The temporal granularity has changed from Date to Month; (b) A classification into subcategories has been inserted into the part hierarchy; (c) A new constraint has been modeled in the customer hierarchy, stating that sale districts belong to nations; (d) The incentive has become independent of the shipment terms. Then, at $t_2 = 1/1/2004$, another version S_2 is created as follows: (a) Two new measures ShippingCostsEU and ShippingCostsLIT are added; (b) The ShipMode dimension is eliminated; (c) A ShipFrom dimension is added; (d) A descriptive attribute PartDescr is added to Part. The conceptual schemata for S_2 is depicted in Fig. 1(b).

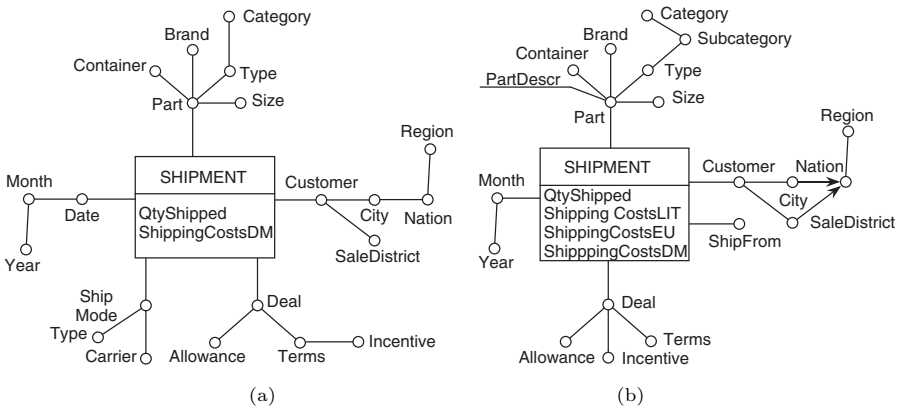

Fig. 1. Conceptual schemata for two versions of the shipment fact: S_0 (a) and S_2 (b)

In our approach, the schema modifications occurring in the course of DW operation lead to the creation of a *history of schema versions*. All versions are available for querying purposes, and those relevant for a particular analysis scenario may either be chosen explicitly by the user or implicitly by the query subsystem. In order to improve cross-version querying, 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 *actions* on past data are proposed to the designer; they may entail checking past data for additional constraints or inserting new data based on user feedback. The set of actions the designer decides to undertake leads to defining and populating an augmented schema S^{AUG}, associated with S, that will be used, instead of S, to determine if a given query q spanning the validity interval of S is correct, i.e., if the data required by q can be consistently queried under the schema specified by q. Importantly, S^{AUG} is always an extension of S, i.e. the instance of S can be computed as a projection of the instance of S^{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 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, 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 more existing at t_1) on Subcategory, it is necessary to: (1) define an augmented schema for S_0, denoted S_0^{AUG}, that contains the new attribute Subcategory; (2) migrate old data entries from S_0 to S_0^{AUG}; and (3) assign the appropriate values for Subcategory to old data entries in S_0^{AUG}. This process will allow queries involving Subcategory to be answered on old data via the instance of S_0^{AUG}. Note

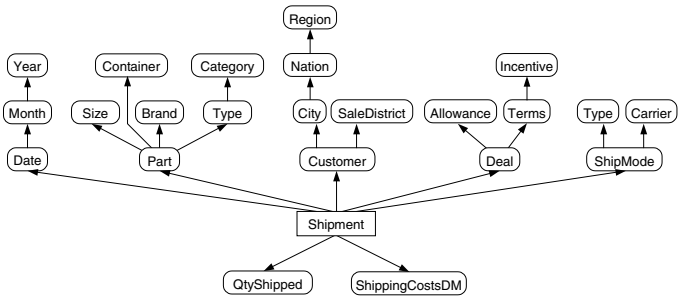

Fig. 2. Schema graph S_0

that, while the first two steps will entirely be managed by the versioning system, the last one requires further input from the designer.

2 Schema Graphs

Following standard notation for relational databases, capital letters from the beginning (ending) of the alphabet denote single (sets of) attributes. We restrict our attention to simple FDs, i.e., FDs of the form $A \rightarrow B$, and in this setting we use F^* to denote the usual graph-theoretic transitive closure of F (where sets of simple FDs are interpreted as directed graphs). We also recall that a set F of FDs is *canonical* if: (a) $(X \rightarrow Y \in F) \Longrightarrow (|Y| = 1)$; (b) $(X \rightarrow A \in F) \wedge (Y \subsetneqq X) \Longrightarrow (Y \not\rightarrow A)$; and (c) $(F' \subsetneqq F) \Longrightarrow (F' \not\equiv F)$. For every set F of FDs there is at least one *canonical cover*, i.e., a canonical set of FDs equivalent to F [4].

In order to talk about versioning, we first have to fix a representation for DW schemata. 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 and allows to define schema modifications by means of four elementary graph manipulations, namely adding and deleting nodes and arcs, and to analyze the schema versioning problem in a simple and intuitive setting.

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

1. E *is called* fact node *and represents a placeholder for the fact itself;*
2. U *is a set of attributes (including properties and measures);*
3. F *is a canonical set of simple FDs defined over* $\{E\} \cup U$*;*
4. E *has only outgoing arcs, and there is a path from* E *to each attribute in* U*.*

The schema graph for the shipment fact in Fig. 1(a) is shown in Fig. 2.

Adopting a canonical form for representing the FDs in schema graphs has the important advantage of providing a non-redundant, i.e., *compact* 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. In the following we will discuss how a set F of simple

FDs can be put in a uniquely determined canonical form, denoted with F^- and called *reduced form*.

We begin by observing that, for *acyclic* sets of simple FDs, canonical covers are uniquely determined [5] and can be computed via transitive reduction [6]. Thus, if F is acyclic, F^- can be defined as the transitive reduction of F. On the other hand, in real-world DW schemata, cycles in F may occur since either (a) 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; or (b) two measures may be derivable from each other – e.g., Euros can be transformed to Italian Liras by applying a constant conversion factor. The approach we follow to *uniquely* select a canonical form for cyclic sets of FDs is formalized in [5]; for space reasons, here we report only an intuitive description.

Consider a schema graph $S = (\hat{U}, F)$, where F is neither necessarily canonical nor acyclic. First, we define an equivalence relation $\equiv_F$ on $\hat{U}$ as: $A \equiv_F B$ iff $(A \rightarrow B \in F^*) \wedge (B \rightarrow A \in F^*)$ for $A, B \in \hat{U}$. Then, we consider the acyclic directed graph where each node is one equivalence class X induced by $\equiv_F$ and an arc goes from X to 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, denoted S', is acyclic and uniquely determined (as transitive reduction is unique for acyclic graphs). Now, let a total order on $\hat{U}$ be given (e.g., user-specified or system-generated based on some sorting criterion such as attribute name). The reduced form for F, F^-, is composed as follows: (1) for each arc (X, Y) in S', one FD from the minimum attribute in X to the minimum attribute in Y; (2) for each equivalence class $X = \{A_1 \dots A_n\}$ where $A_1 < \dots < A_n$, a set of FDs $\{A_1 \rightarrow A_2, \dots, A_{n-1} \rightarrow A_n, A_n \rightarrow A_1\}$. It is possible to prove that F^- is a transitive reduction of F and a canonical cover of F [5]. In the remainder of the paper, we will always consider schema graphs where the set of FDs is in reduced form, and we will use the terms *schema* and *schema graph* interchangeably.

Example 1. Consider the shipment fact in Fig. 1(b), where Part has an equivalent property PartDescr and shipping costs are expressed in EU, LIT, and DM. The schema graph based on the total order induced by attribute names is shown in Figure 3.

Given a schema graph $S = (\hat{U}, F)$ and a set $X \subseteq \hat{U}$, the *projection of F to X* is defined as $\pi_X(F) := \{A \rightarrow B \in F^* \mid AB \subseteq X\}$. Based on [7] and [5] it is possible to show that, if $E \in X$, projection is closed on schema graphs.

3 Versioning

In Section 3.1 we define four modification operations that manipulate schema $S = (\hat{U}, F)$, 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 FD involving existing attributes, and $\texttt{Del}_\texttt{F}()$ to remove an existing FD. Then, in Section 3.2 we show how a new version is created by applying a sequence of schema modification operations, and we explain

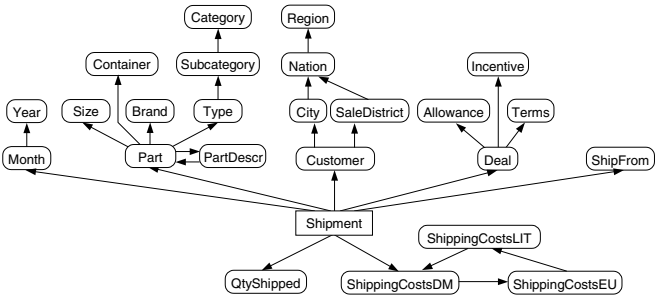

Fig. 3. Schema graph S_2

how augmented schemata are derived based on actions undertaken by the designer. Finally, in Section 3.3 we consider schema versioning based on sequences of versions, which influence the history of schemata and augmented schemata.

3.1 Schema Modification Operations

For each 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 2. *Let $S = (\hat{U}, F)$ be a schema graph and A be a fresh attribute*[1]*. Then $\texttt{New}(S, \texttt{Add}_\texttt{A}(A)) := (\hat{U} \cup \{A\}, F \cup \{E \rightarrow A\})$.*

Definition 3. *Let $S = (\hat{U}, F)$ be a schema graph and $A \in U$ be an attribute. Then $\texttt{New}(S, \texttt{Del}_\texttt{A}(A)) := (\hat{U} \setminus \{A\}, \pi_{\hat{U} \setminus \{A\}}(F)^-)$.*

Definition 4. *Let $S = (\hat{U}, F)$ be a schema graph and $f = A_1 \rightarrow A_2$ be an FD involving attributes in U. Then $\texttt{New}(S, \texttt{Add}_\texttt{F}(f)) := (\hat{U}, (F \cup \{f\})^-)$*

Definition 5. *Let $S = (\hat{U}, F)$ be a schema graph and $f = A_1 \rightarrow A_2$ be an existing FD in F, where $A_1 \neq E$. Then $\texttt{New}(S, \texttt{Del}_\texttt{F}(f)) := (\hat{U}, (F \setminus \{f\} \cup \{E \rightarrow A_2\} \cup \{A_1 \rightarrow A_3 \mid (\exists A_3 \in U)\ A_2 \rightarrow A_3 \in F\})^-)$.*

It is possible to prove that the four operators above are closed, i.e., they yield schema graphs.

Example 2. The sequence of operations applied to add Subcategory to parts is $\texttt{Add}_\texttt{A}(\mathsf{Subcategory}), \texttt{Add}_\texttt{F}(\mathsf{Type} \rightarrow \mathsf{Subcategory}), \texttt{Add}_\texttt{F}(\mathsf{Subcategory} \rightarrow \mathsf{Category})$.

[1] A *fresh* attribute is an attribute that does not occur in S and never occurred in the past.

3.2 Augmented Schemata

We call a *version* a schema that reflects the business requirements during a given time interval. Thus, a version is populated with the events occurring during that interval 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 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.

In temporal databases, versioning is generally associated with the definition of some *data migration strategies* used to consistently move data from the old version to the new one. Data migration is not described in this paper for space reasons; we only briefly note that:

- Events occur at a particular moment in time, so an event occurring at time t conforms to the version that is valid at t and no data migration is necessary.
- Instances of hierarchies are generally active during time intervals (e.g., a part is active from the time it is first shipped to the time it is declared obsolete), so their lifetime may span several versions. Thus, if a new version S is created at time t, for each hierarchy instance that is active both before and after t it may be necessary to migrate it to S. For instance, if S adds a new FD f between already existing attributes, migration requires to check if f holds for all active instances and possibly to enforce it by modifying, under the user guidance, the values of one or both attributes involved in f.

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 and are visible to the querying process. We emphasize that usage of augmented schemata in the course of the querying process is handled transparently to the DW users, i.e., users are only aware of regular versions but not of augmented schemata. Concerning (1), given a version S, let the sequence of operations $M_1(Z_1), \ldots, M_h(Z_h)$ be the executed transaction. Then, the new version S' is defined by executing the modification operations one after another, i.e., $S' = \texttt{New}(S_h, M_h(Z_h))$, where $S_1 = S$ and $S_i = \texttt{New}(S_{i-1}, M_{i-1}(Z_{i-1}))$ for $i = 2, \ldots, h$. In the remainder of this section we address (2) by illustrating how augmented schemata are created at the end of transactions to increase flexibility in cross-version querying.

We anticipate that we only consider augmentations for previous versions and 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[2]. As motivated in Section 1.1, such schema extensions can be reflected

[2] Designers can, of course, delete attributes and FDs from schema versions: the point is that such deletions do not lead to augmentations. Besides, while in a "pure" versioning approach attributes would be deleted only logically, for practical purposes designers might want to delete data physically to free disk space.

Table 1. Actions associated to each new attribute/FD

Element	*Condition*		*Action*
$A \in \texttt{Diff}_\texttt{A}(S, S')$	$(E \rightarrow A) \in F'$	A is measure	estimate values for A
		A is dimension	disaggregate measure values
	$(E \rightarrow A) \notin F'$	A is derived measure	compute values for A
		A is property	consistently add values for A
$f \in \texttt{Diff}_\texttt{F}(S, S')$	-		check if f holds

on previous versions by undertaking appropriate actions, which enable flexible cross-version querying.

Let $S' = (\hat{U}', F')$ be the new version obtained by applying a transaction to version $S = (\hat{U}, F)$, and let $\texttt{Diff}_\texttt{A}(S, S') := U' \setminus U$ (set of new attributes) and $\texttt{Diff}_\texttt{F}(S, S') := F' \setminus F^*$ (set of new FDs). The set of *potential* actions that may be undertaken by the designer on past data in order to increase querying flexibility only depends on $\texttt{Diff}_\texttt{A}(S, S')$ and $\texttt{Diff}_\texttt{F}(S, S')$. The possible actions are reported in Table 1 and defined as follows:

1. *Estimate values for A:* A new measure A has been added. The designer may provide values for A for past events, typically by deriving an estimate based on the values of the other measures. For instance, if a new measure Discount is added to the shipment fact, and the discount applied depends on the shipped quantity bracket, its values for past events may be easily estimated from measure QtyShipped.
2. *Disaggregate measure values:* A new dimension A has been added. The designer may 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 [8]. 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 (e.g. ShippingCostsEU); by definition, the values of A for past events can be computed by applying some known computation to another measure.
4. *Consistently add values for A:* A new property A has been added. The designer may provide values for A in such a way that all specified FDs are satisfied. For instance, when Subcategory is added to the part hierarchy, then each part type must be associated with exactly one subcategory and all types included in each subcategory must belong to the same category.
5. *Check if f holds:* A new FD f has been added. In order to augment f on old version S the designer needs to check whether f also holds for S. To support the designer, the system might perform the necessary (but not sufficient) check that f really holds by inspecting the augmented instance. For instance, when SaleDistrict $\rightarrow$ Nation is added, the system may check

that no sale district including customers from different nations exists. If the check performed by the system fails, f cannot be augmented.

Among the potential actions according to Table 1, the designer may choose to perform some of them but to ignore others: in particular, she will decide to undertake a given action only if, considering the business requirements, she believes that the space/time overhead implied by the action is counterbalanced by the increased querying flexibility. Since *only chosen actions contribute to augmentation*, the overall data volume increase is under the designer's control.

Now, to formalize the notion of schema augmentation, consider a schema modification transaction that starts from version $S = (\hat{U}, F)$ and creates the new version $S' = (\hat{U}', F')$. As explained in the next section, in the presence of *histories* of versions, augmentations may be back-propagated even to older versions. Thus, we assume that for every schema S_i there is already an augmented schema S_i^{AUG}, which is initially identical to S_i but may be augmented in response to every transaction. To this end, we next define the augmentation operation $\mathtt{Aug}(S_i^{AUG}, S, S')$ that (further) augments the augmented schema $S_i^{AUG} = (\hat{U}_i, F_i)$ based on the actions chosen by the designer in response to the schema change from S to S'.

Let $\widehat{\mathtt{Diff_A}}(S, S')$ and $\widehat{\mathtt{Diff_F}}(S, S')$ be the subsets of $\mathtt{Diff_A}(S, S')$ and $\mathtt{Diff_F}(S, S')$, respectively, including only the attributes and FDs whose related actions have been performed by the designer. We note that all attributes occurring in FDs of $\widehat{\mathtt{Diff_F}}(S, S')$ must be contained in $\hat{U}_i \cup \widehat{\mathtt{Diff_A}}(S, S')$, as only those FDs can be augmented whose attributes occur in the augmented schema. Then the *new augmented version for* S_i is defined as follows:

$$\mathtt{Aug}(S_i^{AUG}, S, S') := (\hat{U}_i \cup \widehat{\mathtt{Diff_A}}(S, S'), (F_i \cup \pi_{\hat{U}_i \cup \widehat{\mathtt{Diff_A}}(S,S')}(\widehat{\mathtt{Diff_F}}(S, S')))^-)$$

Informally, the new augmentation for S_i is obtained by adding (1) all the new attributes for which the related actions have been undertaken and (2) all the new valid FDs.

Example 3. In the shipment example, initially we start from version S_0 where $S_0^{AUG} = S_0$. When new version $S_1 = (\hat{U}_1, F_1)$ is created from S_0, we have:

$$\begin{aligned}\mathtt{Diff_A}(S_0, S_1) &= \{\mathsf{Subcategory}\}\\ \mathtt{Diff_F}(S_0, S_1) &= \{\mathsf{SaleDistrict} \rightarrow \mathsf{Nation}, \mathsf{Type} \rightarrow \mathsf{Subcategory},\\ &\qquad \mathsf{Subcategory} \rightarrow \mathsf{Category}\}\end{aligned}$$

Thus, the actions the designer can undertake to augment S_0^{AUG} are: (1) provide values for Subcategory and (2) check if SaleDistrict $\rightarrow$ Nation holds on S_0. Assuming the designer decides to undertake both actions, that the FD SaleDistrict $\rightarrow$ Nation actually holds on S_0 and that subcategory values are added consistently with F_1, we have $\widehat{\mathtt{Diff_A}}(S, S') = \mathtt{Diff_A}(S, S')$ and $\widehat{\mathtt{Diff_F}}(S, S') = \mathtt{Diff_F}(S, S')$. Thus, the new augmented schema for S_0 is $S_0^{AUG} = \mathtt{Aug}(S_0, S_0, S_1) = S_1$.

3.3 Version Histories

A *history* is a sequence H of one or more triples representing versions of the form (S, S^{AUG}, t), where S is a version, S^{AUG} is the related augmented schema, and t is the start of the validity interval of S [3]:

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

where $n \geq 0$ and $t_{i-1} < t_i$ for $1 \leq i \leq n$. Note that, in every history, for the last triple (S_n, S_n^{AUG}, t_n) we have $S_n^{AUG} = S_n$ 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 $H = ((S_0, S_0^{AUG}, t_0))$, where $S_0^{AUG} = S_0$. Schema modifications then change histories as follows. Let $H = ((S_0, S_0^{AUG}, t_0), \ldots, (S_{n-1}, S_{n-1}^{AUG}, t_{n-1}), (S_n, S_n^{AUG}, t_n))$ 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

$$\begin{aligned} H' = (\ & (S_0, \texttt{Aug}(S_0^{AUG}, S_n, S_{n+1}), t_0), \ldots, \\ & (S_n, \texttt{Aug}(S_n^{AUG}, S_n, S_{n+1}), t_n), (S_{n+1}, S_{n+1}^{AUG}, t_{n+1})\ ). \end{aligned}$$

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

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 – hence, potential for answering queries – is growing monotonically with every modification.

Example 4. Consider again the two schema restructurings described in Section 1.1, assuming that all actions are undertaken. The initial history is $((S_0, S_0, t_0))$. At time t_1, when S_1 is created, the history becomes $((S_0, S_1, t_0), (S_1, S_1, t_1))$. Then, at time t_2, when S_2 is created, the history becomes $((S_0, S_2, t_0), (S_1, S_2, t_1), (S_2, S_2, t_1))$.

4 Querying

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

[3] In accordance with [9] we argue that there is no need to distinguish valid time from transaction time in the context of schema versioning. Thus, if a new version S' is created from S at (transaction) time t', the valid time of S' is $[t', +\infty]$, while the valid time of S ends at t'. In other words, we assume that the valid time is defined as an interval that starts upon schema creation time and extends until the next version is created.

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 6. *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.

Given a history H and a (not necessarily connected) temporal interval T, we call the *span* of T on H the set $\texttt{Span}(H, T) = \{S_i^{AUG} \mid (S_i, S_i^{AUG}, t_i) \in H \wedge [t_i, t_{i+1}[\cap T \neq \emptyset\}$ (conventionally assuming $t_{n+1} = +\infty$).

Definition 7. *Given a history* H *and a temporal interval* T, *the* common schema *on* H *along* T *is defined as* $\texttt{Com}(H, T) = \bigotimes_{\texttt{Span}(H,T)} S_i^{AUG}$.

Let q be the last query formulated, and T be the interval determined by the predicates in q on the temporal dimension (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 5. Let $H = ((S_0, S_0^{AUG}, t_0), (S_1, S_1^{AUG}, t_1), (S_2, S_2^{AUG}, t_2))$ be the history for the shipment fact, recall that we have $t_1 = 1/1/2003$ and $t_2 = 1/1/2004$,

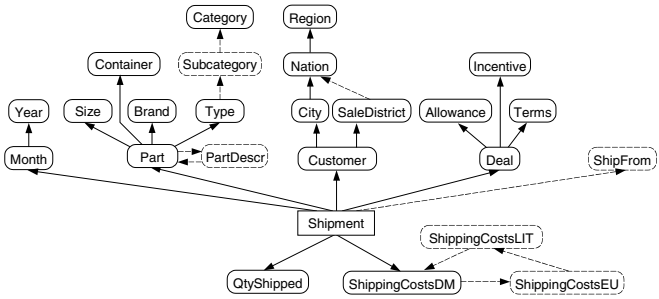

Fig. 4. Formulation contexts for the query in Example 5 without augmentation (in plain lines) and with augmentation (in plain and dashed lines)

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 $\mathtt{Span}(H, T) = \{S_0, S_1, S_2\}$. Fig. 4 shows the formulation context, defined by $S_0^{AUG} \otimes S_1^{AUG} \otimes S_2^{AUG}$, 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, 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.

As to the querying interface, we argue that two approaches for identifying the version for querying, namely *implicit* and *explicit*, should be supported [10]. In this section we considered the implicit approach: given the time interval T of a query, the system computes the widest common schema associated to it. Conversely, in the explicit approach the user chooses a specific version for querying, and the system calculates the widest time interval T that preserves that schema. Obviously, the second approach is best suited for OLAP sessions that analyze data under a specific configuration of attributes and FDs (e.g., *"Compute the total quantity shipped for each month and each subcategory since subcategories have been introduced"*).

5 Conclusions and Related Work

In this paper we have presented an approach towards DW schema versioning. 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 to determine whether a given query, possibly spanning several versions, can be answered based on the information contained in augmented schemata. With reference to the terminology introduced in [11] 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 [12] we are dealing with *partial schema versioning* as no retrospective update is allowed to final users.

In the DW field, mainly four approaches to evolution/versioning can be found in the literature. In [13], the impact of evolution on the quality of the warehousing process is discussed in general terms. In [14] a prototype supporting dimension updates at both the extensional and intensional levels is presented. In [15], an algebra of basic operators to support evolution of the conceptual schema of a DW is proposed. In all these approaches, versioning is not supported and the problem of querying multiple schema versions is not mentioned. Finally, [16] proposes the COMET model to support schema evolution: though the problem of queries spanning multiple schema versions is mentioned, the discussion of how to map instances from one version to another is only outlined.

On the commercial side, the versioning problem has only marginally been addressed, for instance in the *Oracle Change Management Pack* [17] and in KALIDO [18]. In both cases, the possibility of formulating a single query on multiple databases with different schemata is not even mentioned.

References

1. Eder, J., Koncilia, C.: Changes of dimension data in temporal data warehouses. In: Proc. DaWaK. (2001) 284–293
2. Yang, J.: Temporal data warehousing. PhD thesis, Stanford University (2001)
3. Golfarelli, M., Maio, D., Rizzi, S.: The dimensional fact model: a conceptual model for data warehouses. IJCIS **7** (1998) 215–247
4. Maier, D.: The theory of relational databases. Computer Science Press (1983)
5. Lechtenbörger, J.: Computing Unique Canonical Covers for Simple FDs via Transitive Reduction. Technical report, Angewandte Mathematik und Informatik, University of Muenster, Germany. To appear on Information Processing Letters (2004)
6. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM Journal on Computing **1** (1972) 131–137
7. Lechtenbörger, J., Vossen, G.: Multidimensional normal forms for data warehouse design. Information Systems **28** (2003) 415–434
8. Pourabbas, E., Shoshani, A.: Answering Joint Queries from Multiple Aggregate OLAP Databases. In: Proc. 5th DaWaK, Prague (2003)
9. McKenzie, E., Snodgrass, R.: Schema evolution and the relational algebra. Information Systems **15** (1990) 207–232
10. Roddick, J., Snodgrass, R.: Schema versioning. In: The TSQL2 Temporal Query Language. Kluwer Academic Publishers (1995) 425–446
11. Jensen, C.S., Clifford, J., Elmasri, R., Gadia, S.K., Hayes, P.J., Jajodia, S.: A consensus glossary of temporal database concepts. ACM SIGMOD Record **23** (1994) 52–64

12. Roddick, J.: A survey of schema versioning issues for database systems. Information and Software Technology **37** (1995) 383–393
13. Quix, C.: Repository Support for Data Warehouse Evolution. In: Proc. DMDW. (1999)
14. Vaisman, A., Mendelzon, A., Ruaro, W., Cymerman, S.: Supporting dimension updates in an OLAP server. In: Proc. CAiSE. (2002) 67–82
15. Blaschka, M.: FIESTA - A framework for schema evolution in multidimensional databases. PhD thesis, Technische Universitat Munchen, Germany (2000)
16. Eder, J., Koncilia, C., Morzy, T.: The COMET Metamodel for temporal data warehouses. In: Proc. CAiSE. (2002) 83–99
17. Oracle: Oracle change management pack. Oracle Technical White Paper (2000)
18. Kalido: Kalido dynamic information warehouse - a technical overview. KALIDO White Paper (2004)