Complements for Data Warehouses *

D. Laurent
LI/E3I - IUP Informatique
Université de Tours
F-41000 Blois, France
email: laurent@univ-tours.fr

J. Lechtenbörger
Lehrstuhl für Informatik
Universität Münster, Steinfurter Straße 107
D-48149 Münster, Germany
email: lechten@helios.uni-muenster.de

N. Spyratos †
LRI, U.R.A. 410 du CNRS
Bât. 490 - Université de Paris-Sud
F-91405 Orsay Cedex, France
email: spyratos@lri.fr

G. Vossen
Lehrstuhl für Informatik
Universität Münster, Steinfurter Straße 107
D-48149 Münster, Germany
email: vossen@helios.uni-muenster.de

Abstract

Views over databases have recently regained attention in the context of data warehouses, which are seen as materialized views. In this setting, efficient view maintenance is an important issue, for which the notion of self-maintainability *has been identified as desirable. In this paper, we extend self-maintainability to (query and update)* independence*, and we establish an intuitively appealing connection between warehouse independence and* view complements*. Moreover, we study minimal complements and show how to compute them in the presence of key constraints and inclusion dependencies in the underlying databases. Taking advantage of these complements, an algorithm is outlined for the specification of independent warehouses.*

1 Introduction

Views over databases have recently regained attention in the context of *data warehouses*. Indeed, a warehouse can be considered as a collection of *materialized views* over one or more operational databases, for which proper *maintenance* is crucial. Incremental view maintenance has been considered for a long time in the literature already [4, 8, 9, 12]; an overview of maintenance of materialized views appears in [11]. In spite of those rich results, view maintenance in a warehousing environment is complicated by the fact that the sources are decoupled from the warehouse, so that traditional *incremental* view maintenance may exhibit anomalies [27, 28]. In this situation, the notion of *warehouse self-maintainability* has been identified as desirable. Self-maintainability for one view has been investigated in [3, 10, 18], for multiple views in [14], using auxiliary views in [18], and using conditional tables in [21]. In this paper, we generalize self-maintainability to (query and update) *independence*, and exhibit an intuitively appealing and surprising connection between warehouse independence and *view complements* [2]. In addition, we outline an algorithmic approach for the specification of independent warehouses.

A *data warehouse* is nowadays understood as an integrated and time-varying collection of data primarily used in organizational decision making by means of online analytical processing (OLAP) [5, 20]. Typically, it is a standard or specialized DBMS that stores materialized views in order to provide fast access to integrated information [25, 20] extracted from multiple, heterogeneous, autonomous, distributed information sources.

In this paper, we focus on the issue of data integration, and more specifically on warehouse maintenance. Data integration means that data which has been extracted from the sources is merged into the warehouse, initially or after the sources have undergone updates. Integration then means (i) materializing views of the underlying databases, and (ii) maintaining them after updates have occurred at the sources. However, maintenance is more complicated than in traditional databases for various reasons. Indeed, since the information sources are only loosely coupled to the warehouse, they do *not* participate in its maintenance; instead,

*This work was supported by the bilateral French-German PROCOPE program under Grant No. 312/pro-gg.

†Currently on sabbatical leave at Meme Media Laboratory, Hokkaido University, Sapporo, Japan.

they simply report their changes to the warehouse. The warehouse is typically *not* in a position to send queries back to the sources, since that can incur processing delays, the queries may be expensive, and such queries can cause warehouse maintenance anomalies [27, 28]. Even worse, when information sources are highly secure or legacy systems, ad-hoc queries may not be permitted at all. Consequently, it is desirable to ensure that, as much as possible, queries to the sources are not required in order to keep the warehouse data consistent. So the problem is how to maintain the warehouse based on the reported changes at the sources alone. We now illustrate this problem using an example from [26]. As in [26], for the sake of simplicity, we use the relational model for data sources and the relational algebra for specifying views.

Example 1.1 Consider the warehouse scenario shown in Figure 1, where the warehouse consists of the single view $Sold = Sale \bowtie Emp$, and assume that the databases shown currently have the following contents (where *clerk* is assumed to be a key for relation *Emp*):

Sale	*item*	*clerk*
	TV set	Mary
	VCR	Mary
	PC	John

Emp	*clerk*	*age*
	Mary	23
	John	25
	Paula	32

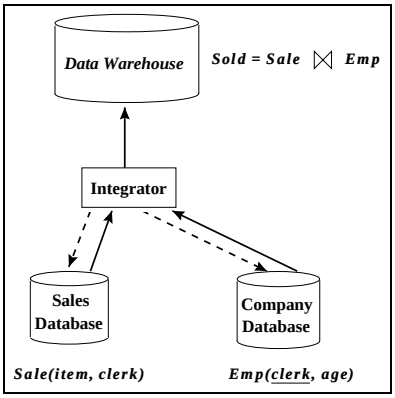

Figure 1. Data warehousing example.

Next, let the Sales database notify the integrator (solid arrows in Figure 1) of the following update: "insert into *Sale* the tuple ⟨Computer, Paula⟩." As we have said, obtaining the information needed by the integrator to maintain the warehouse by querying the sources (dashed arrows) is not an option. Thus, the straightforward approach of having the Company Database join the new tuple with all tuples in relation *Emp* to find out the join tuples is not available. Instead, the warehouse should be able to *maintain itself*; to this end, notice that a subset of the *Emp* relation appears in the warehouse already (as projection of *Sold* onto *clerk* and *age*). It hence suffices to additionally keep the following information in the warehouse:

$$C_1 = Emp \setminus \pi_{clerk,age}(Sold)$$

Since Paula does not appear in the projection of $Sold$ onto $clerk$, a join of tuple ⟨Computer, Paula⟩ with C_1 now yields the data necessary to update the warehouse. Similarly, if the insertion concerns *Emp*, then the integrator will need to know the following set of tuples:

$$C_2 = Sale \setminus \pi_{item,clerk}(Sold)$$

In fact, as is easily seen, C_1 and C_2 provide sufficient information for maintaining the warehouse w.r.t. deletions from *Sale* and *Emp* as well.[1] As we shall see in the following section, C_1 and C_2 form a "warehouse complement". □

In view maintenance, when additional queries over base data are never required to maintain a given view, the view is said to be *self-maintainable*; since a self-maintainable warehouse can be updated or refreshed independently from its underlying sources, we call warehouses with that property *update-independent* (this notion will formally be defined in Section 4).

Clearly, most warehouse views are not update-independent. However, update independence can be ensured by storing additional (*auxiliary*) views at the warehouse. For example, in Figure 1, if we add auxiliary views C_1 and C_2 to the warehouse, it becomes update-independent. Obviously, every warehouse can become update-independent, if *all* relevant data from the sources is copied to the warehouse. It appears to be an open problem to determine the minimum amount of extra information needed for update independence of a given view [26]. We remark that the problem of self-maintainability w.r.t. updates, or update-independence, has attracted considerable attention in the past few years, and partial solutions have been proposed, for example based on key and referential integrity constraints [18].

The approach followed in [18] is to first determine a set $\{E_1, \ldots, E_n\}$ of so-called *maintenance expressions* (assuming a single materialized view V) and then to proceed in either of two different ways:

1. From the maintenance expressions $E_1, \ldots, E_n$, extract auxiliary views that, together with the warehouse view, are self-maintainable w.r.t. updates.

2. Given the warehouse view V, "guess" a set of auxiliary views $\{A_1, \ldots, A_l\}$ and then

   - check if the maintenance expressions $E_1, \ldots, E_n$ can be computed from $W = \{V, A_1, \ldots, A_l\}$, using an algorithm to determine whether a query can be answered using a set of views [6, 16, 19] and

[1] For simplicity, we do not consider modifications here.

- check if the views $\{A_1, \ldots, A_l\}$ can be maintained from W.

In this paper we propose a different approach to self-maintainability w.r.t. updates. Our approach can roughly be described as follows:

1. Given a warehouse $\{V_1, \ldots, V_k\}$, determine a "warehouse complement" (i.e., a special set of auxiliary views to be defined in the following section).

2. Use the one-to-one mapping thus created by the warehouse and its complement to apply an incremental maintenance algorithm.[2]

In fact, our approach is the "opposite" of that of [18], in the sense that we first determine the auxiliary views (i.e., the complement) and then compute the maintenance expressions (assuming *any* number of materialized views — not just a single view as in [18]). In addition, we propose to extend the concept of update-independence to queries as well: Indeed, there is good motivation for enabling warehouses to answer queries that could also be posed directly to the given sources. For example, sources may be unavailable or too busy to answer queries; similar to replicated databases, it may then be attractive for an application to have its queries answered from somewhere else, in this case the warehouse. Moreover, source databases might not tolerate queries from outside, or might be unable to answer queries simply because they are not databases and hence do not understand languages such as SQL or relational algebra. Intuitively, a warehouse is self-maintainable w.r.t. queries, or is *query-independent*, if every query to the sources can be answered using the warehouse relations *only*.

Example 1.2 Consider Figure 1 once more as well as the following query to the sources:

$$Q = \pi_{clerk}(Sale) \cup \pi_{clerk}(Emp)$$

(asking for all clerks that appear either in *Sale* or in *Emp*). Clearly, this query cannot be answered by the warehouse, as relation *Sold* contains only those clerks that appear in both *Sale* and *Emp*. Therefore, the warehouse of Figure 1 is not query-independent.

However, like update independence, query independence can be ensured by storing additional (auxiliary) views at the warehouse: If we add auxiliary views C_1 and C_2 as defined in Example 1.1 to the warehouse, the warehouse becomes query-independent. Indeed, with the addition of C_1 and C_2, the warehouse becomes $\{Sold, C_1, C_2\}$ and can recompute both base relations as follows:

$$\begin{aligned} Emp &= \pi_{clerk,age}(Sold) \cup C_1 \\ Sale &= \pi_{item,clerk}(Sold) \cup C_2 \end{aligned}$$

In the "augmented" warehouse, query Q above can be answered by the following query $\overline{Q}$ (that uses *only* warehouse relations): $\overline{Q} = \pi_{clerk}(Sold) \cup \pi_{clerk}(C_1) \cup \pi_{clerk}(C_2)$ □

We now relate the above observations to a notion introduced in [2]: Call a *complement* of a warehouse any set of auxiliary views which (together with the warehouse) can recompute all base relations. For example, the auxiliary views C_1 and C_2 from Example 1.1 constitute a complement of the warehouse shown in Figure 1. Note that every warehouse has at least one complement (since copying all base relations to the warehouse creates a complement), but obviously the interest is in complements that are *minimal* (in a sense to be defined later). Thus, our warehouse from Figure 1 has already two complements, one consisting of C_1 and C_2, and another consisting of the *Sale* and *Emp* relations. It follows that warehouse complements are not unique in general.

The notion of a warehouse complement we will use here derives directly from the notion of view complement first introduced in [2]. However, view complements were used in [2] to translate updates on the view back to updates on the underlying database. Here we use complements to translate in the opposite direction, i.e., to translate queries and updates on the database to queries and updates on the view (i.e., on the warehouse). Complexity issues related to the computation of view complements were studied in [7].

The main contributions of this paper can be summarized as follows:

1. We extend the concept of update-independence to queries as well (and as we shall see, the latter implies the former, i.e., a query-independent warehouse is also update-independent).

2. We provide a formal framework in which query and update independence can rigorously be defined and studied, for any number of materialized views.

3. We present an algorithm for computing the complement of a warehouse defined by selection, projection, and join. Complements are the formal tool for rendering warehouses independent.

4. We provide correctness criteria for the translation of queries and updates from the sources to the warehouse, and show how previous results on self-maintainability w.r.t. updates can be derived from these criteria.

5. We propose an algorithmic approach to the specification of independent warehouses.

The paper is organized as follows: In Section 2 we define view complements and show how to compute minimal complements under certain conditions. We use com-

[2] As we shall see, the warehouse complement creates a one-to-one mapping from states of the base relations to states of the warehouse and its complement.

plements to define query-independent warehouses in Section 3, and update-independent warehouses in Section 4. Then we proceed, in Section 5, to putting things in perspective and discuss various ways to exploit our results, among them an algorithmic approach to the specification of warehouses that are query and update independent. Section 6 contains concluding remarks and suggestions for further research. Proofs are omitted due to lack of space (the interested reader is referred to [15]).

2 View Complements

We assume the reader to be familiar with the basics of relational databases, for example, along the lines of [23, 24]. We consider a fixed set of relation schemata $D = \{R_1, \ldots, R_n\}$, coming from various databases. The set of attributes on which a relation schema $R \in D$ is defined will be denoted by $attr(R)$, or simply by R if no confusion is possible. A database state over D has the form $d = \langle r_1, \ldots, r_n \rangle$, where r_i denotes a relation over R_i, $1 \leq i \leq n$. For defining views, we will use relational algebra.

A view is defined as a relational expression over D. It can be *virtual*, in which case it is recomputed each time the user accesses the view, or it can be *materialized*, in which case the view is physically stored as a relation in the database. In the sequel, we will often use the terms "relational expression", "view definition", and "view" interchangeably. We will denote the result of evaluating a given view V on a database state d by $V(d)$.

The views over D can be ordered w.r.t. their information content in terms of query containment (cf. [23, 24]):

Definition 2.1 Let U and V be two views over D with $attr(U) = attr(V)$. Define $U \leq V$ if for every state d of D the inclusion $U(d) \subseteq V(d)$ holds. Define $U < V$ if $U \leq V$ and there is a state d of D s.t. $U(d) \subsetneq V(d)$ holds. □

The extension of the view ordering to sets of views is straightforward, i.e., if $U = \{U_1, U_2, \ldots, U_k\}$ and $V = \{V_1, V_2, \ldots, V_k\}$ are sets of views over D then $U \leq V$ if $U_i \leq V_i$, $1 \leq i \leq k$, holds for some ordering of the views in U and V.

Clearly, a set V of views over D expresses some (but usually not all) of the information contained in D. Informally, any set $C = \{C_1, \ldots, C_l\}$ of views over D that expresses the information "missing" from V w.r.t. D is called a complement of V w.r.t. D. More formally, we have:

Definition 2.2 Let $D = \{R_1, \ldots, R_n\}$ be a set of relation schemata, and let $V = \{V_1, \ldots, V_k\}$ be a set of views over D. A *complement* of V (w.r.t. D) is any set $C = \{C_1, \ldots, C_l\}, l \geq 0$, of views over D s.t. each R_i is a view over $\{V_1, \ldots, V_k, C_1, \ldots, C_l\}$, i.e., s.t. each R_i can be computed from the views and their complement (using a relational expression). □

For instance, in Example 1.2 the set $C = \{C_1, C_2\}$ is a complement of $V = \{Sold\}$, as *Sale* and *Emp* can be computed from V, C_1, and C_2.

In fact, a set of views V together with a complement sets up a one-to-one mapping from database states to view states as stated in the following proposition:

Proposition 2.1 Let $D = \{R_1, \ldots, R_n\}$ be a set of relation schemata, and let $V = \{V_1, \ldots V_k\}$ and $C = \{C_1, \ldots, C_l\}$ be sets of views over D. Then C is a complement of V iff for all database states d and d', $d \neq d'$ implies $\langle V_1(d), \ldots, V_k(d), C_1(d), \ldots, C_l(d) \rangle \neq \langle V_1(d'), \ldots, V_k(d'), C_1(d'), \ldots, C_l(d') \rangle$. □

For example, using this characterization we can prove that the set $C = \{C_1, C_2\}$ of Example 1.2 is actually a complement of V, by simply showing that for all database states d and d', $d \neq d'$ implies $\langle V(d), C_1(d), C_2(d) \rangle \neq \langle V(d'), C_1(d'), C_2(d') \rangle$, which is easy to see.

Next, we state some assumptions and notation that are necessary for computing a minimal complement of a set of PSJ expressions, in the presence of key and inclusion constraints:

- We assume that all views are *PSJ views* over D, i.e., relational expressions of the form $\pi_Z(\sigma_\phi(R_{i_1} \bowtie \ldots \bowtie R_{i_k}))$, where $R_{i_1}, \ldots, R_{i_k}$ are in D.
- Given a set V of PSJ views and a base relation R, we denote by V_R the set of views in V whose definition involves R.
- For ease of notation, $\pi_Z(R)$ will denote the usual projection of R onto attribute set Z if $Z \subseteq attr(R)$, or the empty relation (over Z) otherwise.

Proposition 2.2 Let $D = \{R_1, \ldots, R_n\}$ be a set of relation schemata without any integrity constraints. Let $V = \{V_1, \ldots V_k\}$ be a set of PSJ views over D. For $1 \leq i \leq n$ define $\overline{R_i} = \bigcup_{V_j \in V_{R_i}} \pi_{R_i}(V_j)$. Then the set of views $C = \{C_1, \ldots, C_n\}$, where C_i is defined by

$$C_i = R_i \setminus \overline{R_i}, \quad 1 \leq i \leq n, \tag{1}$$

is a complement of V. Moreover, each $R_i \in D$ can be recomputed from C_i and $\overline{R_i}$ as follows:

$$R_i = C_i \cup \overline{R_i}, \quad 1 \leq i \leq n. \tag{2}$$

□

The following examples illustrate the computation of complements and the advantages gained by exploiting the

sharing of information between a set V of views and its complement.

Example 2.1 Let D consist of three relation schemata $R(X,Y)$, $S(Y,Z)$, and $T(Z)$. First, let $V = \{V_1\}$ where $V_1 = R \bowtie S \bowtie T$. Applying the previous proposition, we obtain a complement $C = \{C_R, C_S, C_T\}$, where $C_R = R \setminus \pi_{XY}(V_1), C_S = S \setminus \pi_{YZ}(V_1), C_T = T \setminus \pi_Z(V_1)$. Note that this complement is strictly smaller (w.r.t. the view ordering $\leq$) than the trivial complement $C' = D$.

Let now $V = \{V_1, V_2\}$, where $V_1 = R \bowtie S \bowtie T$ and $V_2 = S$. Then the above proposition yields a different complement, namely $C' = \{C'_R, C'_S, C'_T\}$, where $C'_R = C_R, C'_S = S \setminus (\pi_{YZ}(V_1) \cup \pi_{YZ}(V_2)) = \emptyset, C'_T = C_T$. In this case, C'_S will always be empty, showing that the complement C' is strictly smaller than C. Moreover, using Theorem 2.1 below, we can show that C' is in fact a minimal complement. □

The above example was presented in [14] to illustrate that (update) self-maintainability of multiple views might be achieved in certain situations although individual views considered separately are not self-maintainable: It was shown in [14] that V_1 or V_2 alone are not self-maintainable, but the set $\{V_1, V_2\}$ is. From our point of view we can see that V_2 is a superset of C_S. Consequently, all information from S is available for computing incremental changes. Moreover, it is obvious that $\{V_1, V'_2 = C_S\}$, while storing less information than $\{V_1, V_2\}$, is self-maintainable as well. Note that, as this example shows, storing a complement may be more than what is required for (update) self-maintainability.

Comparing our approach to that of [14], we can say the following:

- As we shall see shortly, our approach can incorporate key constraints and inclusion dependencies, while the approach of [14] cannot.
- Roughly speaking, the complements produced by our approach are the queries to base relations needed by the approach of [14] when the warehouse is not self-maintainable.

In general, the computation of complements according to Proposition 2.2 does *not* guarantee minimal results, as the following example shows.

Example 2.2 Consider $D = \{R(A,B,C)\}$ and a set of views $V = \{V_1, V_2, V_3\}$, where $V_1 = \pi_{AB}(R)$, $V_2 = \pi_{BC}(R)$ and $V_3 = \sigma_{B=b}(R)$.

In this situation, Proposition 2.2 yields $C_R = R \setminus V_3$ as a complement of V. However, the following view is also a complement of V:

$$C'_R = (R \bowtie \pi_{AB}((V_1 \bowtie V_2) \setminus R)) \setminus V_3,$$

because we have:

$$R = C'_R \cup V_3 \cup ((V_1 \setminus \pi_{AB}(C'_R \cup V_3)) \bowtie (V_2 \setminus \pi_{BC}(C'_R \cup V_3))).$$

Moreover, it is easy to see that the complement C'_R is in general strictly smaller than C_R. □

However, for SJ views, i.e., for PSJ views where the final projection includes all attributes occurring in D, Proposition 2.2 does produce minimal complements.

Theorem 2.1 Let $D = \{R_1, \ldots, R_n\}$ be a set of relation schemata without any integrity constraints. Let $V = \{V_1, \ldots V_k\}$ be a set of SJ views over D. Then the complement produced by Proposition 2.2 is minimal. □

In the remainder of this section, we explore the impact of integrity constraints on the size and form of minimal complements. More specifically, we look into key constraints and inclusion dependencies, i.e., constraints of the form $\pi_X(R_i) \subseteq \pi_X(R_j)$, for $X \subseteq attr(R_i) \cap attr(R_j)$. For simplicity, we assume that at most one key is declared for every relation schema. Moreover, we assume that the set of inclusion dependencies over a given D is acyclic [1].

The central ideas for the minimization of complements in the presence of constraints lie in the following observations: First, key constraints might permit the computation of lossless joins while recomputing base relations from views. Second, if K_j is a key for R_j, then an inclusion dependency $\pi_X(R_i) \subseteq \pi_X(R_j)$ with $K_j \subseteq X$ implies that $\pi_X(R_i)$ can be seen as a "view" over R_j whose schema contains the key of R_j. Hence, $\pi_X(R_i)$ can be used to compute additional lossless joins for R_j. Instead of using R_i directly, we use its representation in terms of views and complements (cf. Equation (4) of Theorem 2.2 below).[3]

We introduce some notation first: Let V be a set of PSJ views and let R_j be a relation schema with key K_j.

- We denote by V_{K_j} the set of views in V_{R_j} s.t. V_{R_j} contains K_j, i.e., $V_{K_j} = \{V_i \in V_{R_j} \mid K_j \subseteq Z_i\}$ (where Z_i denotes the set of attributes of view V_i).
- In order to incorporate inclusion dependencies we define

$$V^{ind}_{K_j} = V_{K_j} \cup \left\{ \pi_X(R_i) \;\middle|\; \begin{array}{l} \pi_X(R_i) \subseteq \pi_X(R_j) \text{ and} \\ K_j \subseteq X \end{array} \right\}$$

- We call a subset Y of $V^{ind}_{K_j}$ a *cover* of R_j if
  1. every attribute of R_j is present in some view of Y and
  2. Y is minimal w.r.t. the above property.

  We denote by $\mathcal{C}^{ind}_{R_j}$ the set of all covers of R_j.

[3] For ease of notation we do not allow general inclusion dependencies involving sequences of attributes as in [24], although they could be incorporated by a suitable application of the renaming operator to R_i when computing R_j's complementary relation.

Before stating our main theorem on the computation of complements, we illustrate the above notations by an example.

Example 2.3 Consider the relation schemata $R_1(A, B, C)$, $R_2(A, C, D)$, and $R_3(A, B)$, where A is a key for R_i, $1 \leq i \leq 3$, $\pi_{AB}(R_3) \subseteq \pi_{AB}(R_1)$ and $\pi_{AC}(R_2) \subseteq \pi_{AC}(R_1)$.

Let $V = \{V_1, V_2, V_3, V_4\}$, where $V_1 = R_1 \bowtie R_2, V_2 = R_3, V_3 = \pi_{AB}(R_1), V_4 = \pi_{AC}(R_1)$. Then we have

$$\begin{aligned} V_{K_1} &= \{V_1, V_3, V_4\} \\ V_{K_1}^{ind} &= \{V_1, V_3, V_4, \pi_{AB}(R_3), \pi_{AC}(R_2)\} \\ \mathcal{C}_{R_1}^{ind} &= \{\{V_1\}, \{V_3, V_4\}, \{\pi_{AB}(R_3), V_4\}, \\ &\quad \{V_3, \pi_{AC}(\ R_2)\}, \{\pi_{AB}(R_3), \pi_{AC}(R_2))\}\} \ \square \end{aligned}$$

Theorem 2.2 Let $D = \{R_1, \ldots, R_n\}$ be a set of relation schemata, where K_i is the only key for R_i, $1 \leq i \leq n$, and where inclusion dependencies may be incorporated (as described above). For $1 \leq i \leq n$ define

$$\overline{R_i} = \bigcup_{V_j \in V_{R_i}} \pi_{R_i}(V_j) \text{ and } R_i^{ir} = \bigcup_{Y \in \mathcal{C}_{R_i}^{ind}} \pi_{R_i}(\bowtie_{V_j \in Y} V_j).^4$$

Then the set of views $C = \{C_1, \ldots, C_n\}$, where

$$C_i = R_i \setminus (\overline{R_i} \cup R_i^{ir}), \quad 1 \leq i \leq n, \tag{3}$$

is a complement of V. Moreover, each $R_i \in D$ can be recomputed as follows:

$$R_i = C_i \cup \overline{R_i} \cup R_i^{ir}, \quad 1 \leq i \leq n. \tag{4}$$

In addition, C is minimal among all complements for which the recomputation of base relations is achieved as follows:

1. Joins are always performed along keys, and
2. only complementary views and views from V_K^{ind} are used. $\square$

It is important to note that all joins involved in the recomputation of a base relation are *extension joins* (cf. Equation (4)), and thus can be performed using efficient algorithms [13].

Also note that foreign key constraints (i.e., combinations of key and inclusion constraints) are handled by the previous theorem as well. Such constraints were also used in [18] in order to reduce the size of auxiliary views stored in the warehouse. Note, however, that our method is applicable to any number of views (and not just to a single view as in [18]).

The following example demonstrates the role of constraints in reducing the size of a complement.

Example 2.3 (continued) Consider again the relation schemata $R_1(A, B, C)$, $R_2(A, C, D)$, $R_3(A, B)$ and views $V = \{V_1, V_2, V_3, V_4\}$, and assume first that there are no constraints. Then, according to Theorem 2.2, the views V_3 and V_4 are of no use in the computation of a complement, and we obtain the complementary views $C_1 = R_1 \setminus \pi_{ABC}(V_1)$, $C_2 = R_2 \setminus \pi_{ACD}(V_1)$, and $C_3 = R_3 \setminus V_2 = \emptyset$.

Assume now, that A is a key for R_1. Then we have a lossless join $R_1 = R_1^{ir} = V_3 \bowtie V_4$, and so $C_1 = \emptyset$. The other views remain unchanged.

To see the effect of inclusion dependencies, consider $V' = \{V_1, V_3\}$ in a situation where A is a key for $R_i, 1 \leq i \leq 3$, and $\pi_{AC}(R_2) \subseteq \pi_{AC}(R_1)$. Then we have $C_2 = R_2 \setminus \pi_{ACD}(V_1)$ and $C_3 = \emptyset$, as seen before. Furthermore, $\overline{R_1} = \pi_{ABC}(V_1)$ and $R_1^{ir} = \pi_{ABC}(V_1) \cup \pi_{ABC}(V_3 \bowtie \pi_{AC}(R_2))$. Thus, $R_1^{ir} = \pi_{ABC}(V_1) \cup \pi_{ABC}(V_3 \bowtie \pi_{AC}(C_2 \cup \pi_{ACD}(V_1)))$, giving rise to $C_1 = R_1 \setminus (\overline{R_1} \cup R_1^{ir})$. $\square$

Example 2.4 For a different effect of inclusion dependencies, refer to Figure 1, and assume now that there is a referential integrity constraint stating that every clerk of *Sale* also appears in *Emp* (i.e., $\pi_{clerk}(Sale) \subseteq \pi_{clerk}(Emp)$). As a consequence, every tuple of *Sale* has a join partner in *Emp*. Hence C_2 is always empty, and we obtain $C = \{C_1, \emptyset\}$ as a complement of $V = \{Sold\}$. $\square$

So far we have seen complements and their computation in the presence of key constraints and inclusion dependencies. In the following sections we discuss the importance of complements in the specification of independent warehouses.

3 Query Independence

As we have seen, a *warehouse* over $D = \{R_1, \ldots, R_n\}$ can be seen as a set $\{V_1, \ldots, V_k\}$ of view definitions over D, called the *warehouse definition*. These view definitions are evaluated and stored initially, and then maintained as the base relations change. Thus, a warehouse state is a set of materialized views. For a given database state d, a warehouse state has the form $\langle V_1(d), \ldots, V_k(d)\rangle$.

Roughly, query independence means that every query Q that can be posed to D can be answered in terms of executing a corresponding query $\overline{Q}$ over the warehouse. More formally, query independence can be defined as follows:

Definition 3.1 A warehouse $W = \{V_1, \ldots, V_k\}$ over D is *query-independent* if for every query Q over D there exists a query $\overline{Q}$ over W s.t. $Q = \overline{Q} \circ W$, i.e., $Q(d) = \overline{Q}(W(d))$ for all states d of D (cf. Figure 2).[5] $\square$

As is easily seen, most warehouses are not query-independent. However, in this section we propose an approach for making a given warehouse V (defined as a set of

[4] The superscript *ir* indicates that this view contains tuples derived due to interrelational dependencies.

[5] We call attention to the fact that the symbol W is overloaded. Depending on the context, it stands for the set of warehouse views, or for the function that maps states of D to warehouse states.

PSJ views over D with key constraints and inclusion dependencies) query-independent. Our approach can be summarized as follows:

Step 1: By applying Theorem 2.2 we find a complement C of V, and add its views to the warehouse, i.e., we obtain a new warehouse $W = V \cup C$.

Step 2: There is now a one-to-one mapping from database states to warehouse states (cf. Proposition 2.1). We denote this mapping by W and its inverse by W^{-1}. The inverse is given by Equation (4) (cf. Theorem 2.2).

Step 3: For every query Q over D, we define the query $\overline{Q}$ over W by $\overline{Q} = Q \circ W^{-1}$.

Step 4: We show that the query $\overline{Q}$ provides the desired translation of Q, i.e., $Q = \overline{Q} \circ W$ (see Theorem 3.1 below).

In the following we illustrate this approach by an example, and we show its correctness. Consider the warehouse $V = \{Sold\}$ from Example 2.4 once more, and suppose that *clerk* $\rightarrow$ *age* is a key constraint with referential integrity $\pi_{clerk}(Sale) \subseteq \pi_{clerk}(Emp)$.

Step 1 of the algorithm produces the complement $C = \{C_1\}$ seen earlier, where $C_1 = Emp \setminus \pi_{clerk,age}(Sold)$. Thus, we obtain the new warehouse $W = \{Sold, C_1\}$.

According to Step 2, there is now a one-to-one mapping W from states of $\{Emp, Sale\}$ to warehouse states, and Equation (4) of Theorem 2.2 expresses the inverse W^{-1} of this mapping, namely: $Emp = C_1 \cup \pi_{clerk,age}(Sold)$, $Sale = \pi_{item,clerk}(Sold)$.

Now, applying Step 3, and using the inverse W^{-1}, we can translate any query Q over D into a query $\overline{Q}$ over W, as indicated in the diagram of Figure 2.

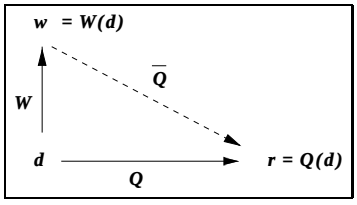

Figure 2. Query independence expressed as a commuting diagram.

This diagram shows the state d of D, the corresponding state $w = W(d)$ of the warehouse, and the result $r = Q(d)$ of evaluating query Q in state d. In order to express the query Q in terms of a query $\overline{Q}$ over w it is sufficient to substitute $W^{-1}(W(d))$ for d, which results in the following translation:

$$Q(d) = Q(W^{-1}(W(d))) = Q(W^{-1}(w)).$$

Using this translation we can now define $\overline{Q} = Q \circ W^{-1}$, the desired query over w. For example, the query

$$Q = \pi_{age}(\sigma_{item=computer}(Sale) \bowtie Emp)$$

over D (asking for the ages of clerks that have sold computers) can be answered by the following query over the warehouse:

$$\overline{Q} = \pi_{age}(\ \sigma_{item=computer}(\pi_{item,clerk}(Sold)) \bowtie (\pi_{clerk,age}(Sold) \cup C_1))$$

This translation is done automatically by simply substituting for *Sale* and *Emp* in Q the expressions provided by the inverse of the warehouse definition.

The following theorem shows the correctness of our approach:

Theorem 3.1 Let $D = \{R_1, \ldots, R_n\}$ be a set of relation schemata, and let V be a warehouse over D. Let C be a complement of V w.r.t. D. Then

a) $W = V \cup C$ is a query-independent warehouse, and

b) for every query Q over D, the query $\overline{Q}$ over W defined by $\overline{Q} = Q \circ W^{-1}$ satisfies the condition for query independence, i.e., $Q = \overline{Q} \circ W$. □

4 Update Independence

As we have explained in the Introduction, a warehouse is update-independent if no queries over base relations are required in order to keep the warehouse data consistent with the base data. The only information available to the warehouse are its views and the changes occurring in the base relations. Most warehouses are not update-independent but they can become so if additional (auxiliary) views are added and maintained at the warehouse level [18]. The purpose of this section is to study update independence and to show that there is a close relationship between query independence and update independence.

The problem of update independence (or self-maintainability, as it was called previously) was defined in [18] as follows: Consider a warehouse W over a set of base relation schemata D. Updates u are applied to the current state d of D in response to which views at the warehouse need to be maintained. We want to compute w', the update of the current warehouse state w (dashed arrow in Figure 3), using as little extra information as possible. If w' can be computed using only the materialized warehouse views and the database update u, then the warehouse W is *update-independent* or *self-maintainable*.

In Figure 3 this notion of update independence is sketched in terms of a commuting diagram. More formally, we have:

Definition 4.1 A warehouse W over a set of base relation schemata D is *update-independent* if the following condi-

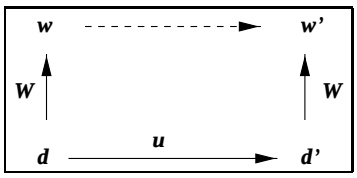

Figure 3. Update independence expressed as a commuting diagram.

tion holds: If an update u changes the database from state d to state d' then there exists a warehouse state w' s.t.

1. w' can be expressed in terms of warehouse relations and the update u only, and
2. $w' = W(d')$. □

We call Condition 2 of the above definition the *correctness criterion* for warehouse updates.

The solution that we propose for finding the correct warehouse state w', in terms of w and u only, can be summarized as follows:

1. Assuming V is a warehouse defined as a set of PSJ views over D with key constraints and inclusion dependencies, compute a complement C of V, and add its views to the warehouse to obtain a new warehouse $W = V \cup C$.
2. There is now a one-to-one mapping from database states to warehouse states (see previous section). Denote this mapping by W and its inverse by W^{-1}.
3. For every update u over D, define the new warehouse state w' in terms of its old state w by $w' = W \circ u \circ W^{-1}(w)$.
4. The new warehouse state satisfies the correctness criterion as stated in Theorem 4.1 below.

Theorem 4.1 Let $D = \{R_1, \ldots, R_n\}$ be a set of base relation schemata, and let $V = \{V_1, \ldots, V_k\}$ be a warehouse with a complement $C = \{C_1, \ldots, C_l\}$, both defined over D. Then

a) $W = V \cup C$ is an update-independent warehouse and

b) for every update u over D which changes the base relations from state d to d', the new warehouse state w' computed from the old state $w = W(d)$ by $w' = W(u(W^{-1}(w)))$ satisfies the correctness criterion for warehouse updates, i.e., $w' = W(d')$. □

By storing a complement at the warehouse level a (query and update) independent warehouse environment is established, which can be maintained using local information without accessing base data. Whenever the warehouse is notified about source updates, its new state has to be determined. Of course, due to performance reasons it is not feasible to recompute the warehouse views from scratch, even if all necessary information is available locally. Instead, it is far more attractive to derive incremental expressions for each kind of update and to evaluate them online, when source changes have to be integrated. For this purpose, incremental view maintenance algorithms, that were originally developed for view maintenance in centralized databases, are applicable in our setting, e.g. those in [4, 9], see [11] for an overview. All that has to be done to exploit those algorithms is to replace any reference to a base relation occurring in the maintenance expression by its inverse (Equation (4) of Theorem 2.2).

Example 4.1 Consider the scenario of Example 1.1 once more, and suppose that a set Δs of tuples is inserted into *Sale*. The following incremental maintenance expressions compute the new warehouse state in terms of Δs and the base relation Emp :

$$\begin{aligned} Sold' &= Sold \cup (\Delta s \bowtie Emp) \\ C_1' &= C_1 \setminus \pi_{clerk,age}(\Delta s \bowtie Emp) \\ C_2' &= C_2 \cup (\Delta s \setminus \pi_{item,clerk}(\Delta s \bowtie Emp)) \end{aligned}$$

In the above expressions, if we replace all references to Emp by their inverse, we obtain the following incremental maintenance expressions in terms of warehouse views only:

$$\begin{aligned} Sold' &= Sold \cup (\Delta s \bowtie (\pi_{clerk,age}(Sold) \cup C_1)) \\ C_1' &= C_1 \setminus \pi_{clerk,age}(\Delta s \bowtie (\pi_{clerk,age}(Sold) \cup C_1)) \\ C_2' &= C_2 \cup (\Delta s \setminus \pi_{item,clerk}(\Delta s \bowtie \\ &\qquad (\pi_{clerk,age}(Sold) \cup C_1))) \end{aligned}$$

Recall that the view *Sold* alone is not update-independent, and so (incremental) maintenance without auxiliary views requires queries involving base relations. In contrast, using a complement we are able to replace all references to base relations by warehouse views and to obtain the new warehouse state in an incremental and independent fashion. □

We end this section by noting that a query-independent warehouse is update-independent, whereas the converse is not true in general. That is, there are warehouses that are update-independent without being query-independent. For example, consider a warehouse $W = \sigma_\phi(R)$ defined over a single relation schema R using selection condition ϕ. Then W is obviously *not* query-independent (for non-trivial conditions ϕ). Yet W is update-independent: Every update on R can be translated to an update on W, *without* using a complement. Indeed, if we add to r a set of tuples Δr, we find:

$w' = \sigma_\phi(r \cup \Delta r) = \sigma_\phi(r) \cup \sigma_\phi(\Delta r) = w \cup \sigma_\phi(\Delta r)$
And if we delete Δr from r we find:
$w' = \sigma_\phi(r \setminus \Delta r) = \sigma_\phi(r) \setminus \sigma_\phi(\Delta r) = w \setminus \sigma_\phi(\Delta r)$
That is, insertions and deletions on r can be translated to insertions and deletions on the warehouse, based only on Δr and the warehouse definition $\sigma_\phi(R)$.

5 Putting Warehouse Complements in Perspective

In this section, we put together the results presented so far in the form of an algorithmic approach to warehouse specification.

Given a warehouse V (defined as a set of PSJ views over a set of base relation schemata D with key constraints and inclusion dependencies), our approach proceeds in a number of steps to determine a complement C of V, a set of algebraic expressions for computing the answers to queries over base data in terms of the warehouse and its complement, and a set of algebraic expressions for computing the changes of the warehouse and its complement in terms of the base relations and their changes.

Step 1: *Use Theorem 2.2 to compute*

1.1 a complement C of V (thus defining a new warehouse $W = V \cup C$);

1.2 the inverse W^{-1} of the warehouse definition (which is a set of algebraic expressions defining each base relation in terms of warehouse views).

Step 2: *Find the expressions for query translation:*
To answer a database query at the warehouse level, simply replace every reference to a base relation r by its inverse (from Step 1.2 above).

Step 3: *Find the maintenance expressions:*
To translate a database update at the warehouse level, use an incremental view maintenance algorithm and derive maintenance expressions, by simply replacing every reference to a base relation r by its inverse (from Step 1.2 above).

Several remarks are in order here, concerning the above approach. First, the main tool for rendering a given warehouse query- and update-independent is the notion of complement. Our approach delivers

a) auxiliary views if needed (in the form of a complement) and

b) inverse warehouse expressions (which can be used to answer database queries at the warehouse level, and to maintain the warehouse).

The warehouse user does not need to be aware of complementary views or query rewriting. At warehouse definition time (or, in a running warehouse environment, even later) all necessary expressions can be automatically derived from the base relation schemata and the view definitions. Furthermore, query rewriting for answering database queries and incremental view maintenance can be integrated in the warehousing environment and can be handled automatically as well.

Next, it is current practice to build warehousing environments on top of star schemata around fact and dimension tables which integrate information from various sources [5, 20]. If we assume all sources to be relational, then each fact table can be regarded as a PSJ view (or unions thereof) and maintained using our approach. For example, consider a business warehouse where parts from different suppliers are sold to customers according to their orders (similar to the one modeled in the TPC-D decision support benchmark [22]). This business could be distributed over several locations, each running its own operational database. Now, the warehouse maintains

a) dimension tables to store data on locations, customer, and supplier and

b) fact tables (including foreign keys from the dimension tables) for orders and sales which are extracted by PSJ queries from the sources and integrated by union.

Although views including union cannot be used for computing complements in general, the presence of foreign keys allows us to uniquely determine the origin of each tuple in a fact table by selecting on the dimension attributes. Thus, we can even exploit fact tables, that are integrated by union, for computing the warehouse complement. As a result, star schemata allow for an even wider applicability of our approach.

Finally, as mentioned in the Introduction, OLAP is a major application domain for data warehousing, where analysts execute complex queries involving aggregate views defined on fact tables. Although aggregate queries cannot be exploited when computing complements, they do *not* restrict the applicability of our approach either: The fact tables can be maintained as described above using PSJ views, whereas view maintenance algorithms for aggregate queries, e.g., [8, 12, 17], can be used to maintain materialized aggregate queries.

6 Concluding Remarks and Future Work

We have presented an approach for specifying warehouses that are self-maintainable w.r.t. queries *and* updates. The key idea behind our approach is setting up a one-to-one mapping from database states to warehouse states, so that the warehouse can recompute all base relations, if necessary. We have seen that this idea can be implemented by

adding a complement to the warehouse, and we have given an algorithm for computing complements for warehouses defined by PSJ expressions from databases containing key dependencies with referential integrity. Previous work has considered self-maintainability of warehouses w.r.t. updates only.

We remark that the complementary relations $\{C_1, \ldots, C_n\}$ given by Theorem 2.2 are *all* the information we need for warehouse independence. If the queries to base relations required for the computation of any specific C_i can be answered in reasonable time, then we do not need to maintain C_i at the warehouse; we simply store the expression for computing it. Otherwise, we have to maintain C_i at the warehouse.

We are currently pursuing the following lines of research. First, the computation of minimal complements needs to be studied in more depth: In this paper we have assumed that each view in the complement has the same set of attributes as some base relation. Relaxing the restrictions on the form of the complements may lead to smaller complements. Second, the relationship between query independence and update independence needs to be examined in more detail. We have seen that query independence implies update independence. However, we have also mentioned that update independence can be obtained without the use of a complement (in certain cases). So a relevant question here is to investigate cases in which update independence can be obtained with auxiliary information strictly smaller than a complement, and to determine the necessary amount as well as the degree of query independence that we obtain in such cases.

References

[1] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] F. Bancilhon, N. Spyratos, "Update Semantics of Relational Views," ACM TODS 1981, 6 (4), 557–575.

[3] J.A. Blakeley, N. Coburn, P. Larson, "Updating derived relations: Detecting irrelevant and autonomously computable updates," ACM TODS 1989, 14 (3), 369–400.

[4] J.A. Blakeley, P. Larson, F.W. Tompa, "Efficiently Updating Materialized Views," Proc. ACM SIGMOD 1986, 61–71.

[5] S. Chaudhuri, U. Dayal, "An Overview of Data Warehousing and OLAP Technologies," Proc. ACM SIGMOD 1997, 65–74.

[6] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim, "Optimizing Queries with Materialized Views," Proc. 11th ICDE 1995, 190–200.

[7] S.S. Cosmadakis, C.H. Papadimitriou, "Updates of Relational Views," JACM 31, 1984, 742–760.

[8] T. Griffin, L. Libkin, "Incremental Maintenance of Views with Duplicates," Proc. ACM SIGMOD 1995, 328–339.

[9] T. Griffin, L. Libkin, H. Trickey, "An Improved Algorithm for the Incremental Recomputation of Active Relational Expressions," IEEE TKDE 9, 1997, 508–511.

[10] A. Gupta, H.V. Jagadish, I.S. Mumick, "Data Integration using Self-Maintainable Views," Technical Memorandum, AT&T Bell Laboratories, November 1994.

[11] A. Gupta, I.S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications," IEEE Data Engineering Bulletin 1995, 18 (2), 3–18.

[12] A. Gupta, I.S. Mumick, V.S. Subrahmanian, "Maintaining Views Incrementally," Proc. ACM SIGMOD 1993, 157–167.

[13] P. Honeyman, "Extension Joins", Proc. 6th VLDB 1980, 239–244.

[14] N. Huyn, "Multiple-View Self-Maintenance in Data Warehousing Environments", Proc. 23rd VLDB 1997, 26–35.

[15] D. Laurent, J. Lechtenbörger, N. Spyratos, G. Vossen, "Complements for Data Warehouses," Technical Report, University of Münster, to appear.

[16] A. Levy, A. Mendelzon, Y. Sagiv, D. Srivastava, "Answering Queries using Views," Proc. 14th ACM PODS 1995, 95–104.

[17] I.S. Mumick, D. Quass, B.S. Mumick, "Maintenance of Data Cubes and Summary Tables in a Warehouse,", Proc. ACM SIGMOD 1997, 100–111.

[18] D. Quass, A. Gupta, I.S. Mumick, J. Widom, "Making Views Self-Maintainable for Data Warehousing," Proc. PDIS 1996.

[19] A. Rajaraman, Y. Sagiv, J.D. Ullman, "Answering Queries using templates with binding patterns," Proc. 14th ACM PODS 1995, 95–104.

[20] A. Sen, V.S. Jacob, eds., "Industrial-Strength Data Warehousing," CACM 41 (9) 1998, 28–69.

[21] H. Shu, "View Maintenance Using Conditional Tables," Proc. 5th DOOD 1997, Springer LNCS 1341, 67–84.

[22] Transaction Processing Council, "TPC Benchmark(tm) D (decision support), Standard Specification, Revision 1.3.1," San Jose, 1998, available on the web under URL http://www.tpc.org.

[23] J.D. Ullman, *Principles of Database and Knowledge-Base Systems*, Volume I, Computer Science Press, 1988.

[24] G. Vossen, *Data Models, Database Languages, and Database Management Systems*, Addison-Wesley, 1991.

[25] J. Widom, ed., *Special Issue on Materialized Views and Data Warehousing*, IEEE Data Engineering Bulletin 18 (2) 1995.

[26] J. Widom, "Research Problems in Data Warehousing," Proc. 4th CIKM 1995.

[27] Y. Zhuge, H. Garcia-Molina, J. Hammer, J. Widom, "View Maintenance in a Warehousing Environment," Proc. ACM SIGMOD 1995, 316–327.

[28] Y. Zhuge, H. Garcia-Molina, J.L. Wiener, "Multiple View Consistency for Data Warehousing," Proc. 13th ICDE 1997, 289–300.