Annotation propagation revisited for key preserving views

Citation for published version:
Cong, G, Fan, W & Geerts, F 2006, Annotation propagation revisited for key preserving views. in *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management, Arlington, Virginia, USA, November 6-11, 2006.* ACM, pp. 632-641. https://doi.org/10.1145/1183614.1183705

Digital Object Identifier (DOI):
10.1145/1183614.1183705

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Early version, also known as pre-print

Published In:
Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management, Arlington, Virginia, USA, November 6-11, 2006

Annotation Propagation Revisited for Key Preserving Views

Gao Cong
University of Edinburgh
gao.cong@ed.ac.uk

Wenfei Fan
University of Edinburgh
Bell Laboratories
wenfei@inf.ed.ac.uk

Floris Geerts
University of Edinburgh
Hasselt University
Transnational University of Limburg
fgeerts@inf.ed.ac.uk

Abstract

This paper revisits the analysis of annotation propagation from source databases to views defined in terms of conjunctive (SPJ) queries. Given a source database D, an SPJ query Q, the view $Q(D)$ and a tuple ΔV in the view, the *view* (resp. *source*) *side-effect* problem is to find a minimal set ΔD of tuples such that the deletion of ΔD from D results in the deletion of ΔV from $Q(D)$ while minimizing the side effects on the view (resp. the source). A third problem, referred to as the *annotation placement* problem, is to find a single base tuple ΔD such that annotation in a field of ΔD propagates to ΔV while minimizing the propagation to other fields in the view $Q(D)$. These are important for data provenance and the management of view updates. However important, these problems are unfortunately NP-hard for most subclasses of SPJ views [5].

To make the annotation propagation analysis feasible in practice, we propose a *key preserving* condition on SPJ views, which requires that the projection fields of an SPJ view Q retain a key of each base relation involved in Q. While this condition is less restrictive than other proposals [11, 14], it often simplifies the annotation propagation analysis. Indeed, for key-preserving SPJ views the annotation placement problem coincides with the view side-effect problem, and the view and source side-effect problems become tractable. In addition we generalize the setting of [5] by allowing ΔV to be a group of tuples to be deleted, and investigate the insertion of tuples to the view. We show that group updates make the analysis harder: these problems become NP-hard for several subclasses of SPJ views. We also show that for SPJ views the source and view side-effect problems are NP-hard for single-tuple insertion, but are tractable for some subclasses of SPJ for group insertions, in the presence or in the absence of the key preservation condition.

Categories and Subject Descriptors: F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous; H.2.4 [**Database Management**]: Systems — *Relational Databases*
General Terms: Theory, Algorithms
Keywords: Annotations, Provenance, View updates.

1. Introduction

It is common to find real-world data dirty [17]. To cope with this, corrections or annotations of errors are often added to the data by experts. This information is essential to the quality (accuracy and timeliness) of the data, and should be carried over along with the regular data when the data is migrated, transformed or integrated. With this comes the need for studying annotation propagation, *i.e.,* how annotations propagate through migration, transformation or integration processes. The analysis of annotation propagation has proved important in data provenance [5, 6] (aka. lineage [10, 9]) for tracing the origin of a piece of data, data cleaning [17] for improving data quality, and in security [19] for enforcing access control, among other things.

In many applications data migration, transformation or integration can be expressed as views defined in terms of conjunctive queries, *i.e.,* SPJ queries defined in terms of the selection, projection, join and renaming operators of the relational algebra. The analysis of annotation propagation can thus be formalized as follows: We consider *annotations* to be pieces of information associated with a location (tuple) in a relation. More formally, the triple (R, t, A) indicates that the annotation A is associated with tuple t in relation R. Given a source database D and an SPJ query Q, annotations are *propagated* to the view $V = Q(D)$ in the straightforward way (see [5] for more details). The associated problems addressed in this paper are the following: Given a tuple ΔV in the view,

- the *view side-effect* problem is to find a set ΔD of tuples in D such that $Q(D) - Q(D - \Delta D)$ is a *minimal* set containing ΔV and in addition, among all such source updates, ΔD is *minimal*; intuitively, this means that the deletion of ΔD from the source leads to the removal of ΔV from the view with minimal view side effect; in other words, ΔD indicates how the tuple ΔV gets into the view, and the problem is to identify a minimal set ΔD of locations (tuples) in the source such that annotations in those places propagate to a minimal number of tuples in the view including ΔV;

PCName	Conf
Joe	CIKM
John	CIKM
Tom	CIKM
John	SIGMOD

(a) Table PC($\underline{PCName, Conf}$)

Conf	Topic	# Paper
CIKM	IR	30
CIKM	DB	30
SIGMOD	DB	30

(b) Table Conf($\underline{Conf, Topic}, \#Paper$)

PCName	Topic
Joe	IR
Joe	DB
Tom	IR
Tom	DB
John	IR
John	DB

(c) View Q_1= $\pi_{PCName,Topic}(PC \bowtie Conf)$

PCName	Conf	Topic
Joe	CIKM	IR
Joe	CIKM	DB
Tom	CIKM	IR
Tom	CIKM	DB
John	CIKM	IR
John	CIKM	DB
John	SIGMOD	DB

(d) Key preserving view Q_2= $\pi_{PCName,Conf,Topic}(PC \bowtie Conf)$

Figure 1: Example of propagation problems

- the *source side-effect* problem is to find a *minimal set* ΔD of tuples in D such that $Q(D) - Q(D - \Delta D)$ contains ΔV; intuitively, it is to simply find a minimal set of locations (tuples) in the source such that the desired annotation in the view can be obtained by annotating in those places in the source, regardless of side effects on the view; as opposed to the previous problem, the source side-effect problem does not require $Q(D) - Q(D - \Delta D)$ to be minimal;

- the *annotation placement* problem is to find, given a field (location) in the tuple ΔV, a *single* tuple ΔD in D such that annotation in a field of ΔD propagates to a minimal number of tuples (locations) in the view including ΔV; in other words, when some error or other annotation is known for the tuple in ΔV, the problem is to find the corresponding tuple (location) in the source D to concretely annotate such that the annotation propagates to the view.

Example 1.1: To illustrate these decision problems, consider a database D consisting of two relations, PC($\underline{PCName, Conf}$) and Conf($\underline{Conf, Topic}, \#Paper$) (with keys underlined), and an SPJ query (view definition) Q_1= $\pi_{PCName,Topic}(PC \bowtie Conf)$. An instance of each relation and the view $Q_1(D)$ are shown in Fig. 1(a)-(c) (ignore Fig. 1(d) for now). Suppose that John is not a database researcher and thus the tuple (John, DB) in the view $Q_1(D)$ is an error, *i.e.,* ΔV = {(John, DB)}. We want to find locations (tuple fields) in the base relations of D to annotate the error information such that the annotations propagate to the fields in the view tuple ΔV via Q_1; in other words, we want to find tuples in D to remove such that their removal leads to the deletion of the error ΔV. The three decision problems described above impose different conditions on how to do this.

(1) View-side effect problem: There are multiple ways to remove tuples in D in order to delete ΔV from the view. Note that D tuples related to ΔV, *i.e.,* those tuples with matching values in ΔV, are (John, CIKM), (John, SIGMOD), (CIKM, DB, 30) and (SIGMOD, DB, 30). While removing certain combinations of these tuples leads to the removal of ΔV, we want to find a combination ΔD such that the removal of ΔD incurs minimal side effect on the view, *i.e.,* it deletes ΔV and a least number of other tuples from the view, and furthermore, ΔD contains the least number of tuples. In other words, we want to find and annotate ΔD such that all the fields of the view tuple ΔV will be annotated by propagation of the annotations in ΔD via Q_1. One solution is to remove {(John, CIKM), (John SIGMOD)} from the PC table, and the other is by removing (John, CIKM) from PC and (SIGMOD, DB, 30) from Conf. Note that none of the solutions is side-effect free: the first solution, for example, also results in the deletion of (John, IR) from the view.

(2) Source side-effect problem: The difference from (1) is that we do not care about the view side-effect when we search for a minimal set ΔD of tuples in D to delete. Thus in this case, removing {(SIGMOD, DB, 30), (CIKM, DB, 30)} from Conf is also a solution although it incurs more severe view side effects than the solutions given above.

(3) Annotation placement problem: Suppose that the information "John is not a database researcher" is to be annotated in the *PCName* field of ΔV. We want to find a single tuple ΔD in the database D to annotate such that the annotation propagates to the *PCName* field of ΔV and a least number of other fields in the view $Q_1(D)$. Here the solution is ΔD = (John, SIGMOD): by annotating the *PCName* field of ΔD we get the desired annotation in the view with zero side effect. □

Prior work. Although there has been a host of work on data provenance [4, 5, 6, 9, 10, 19], the complexity bounds for the decision problems associated with annotation propagation analysis are only studied in [5, 19], in which it is shown that the analysis is in general beyond reach in practice. Indeed, the view and source side-effect problems are NP-hard for views expressed in SPJ and in fact in its subclass PJ [5], and the annotation placement problem is NP-hard for PJ and SPJ views [19]. Although these problems are also of interest to the management of view updates, their complexity is not addressed in that line of work and the only complexity results known in the study of view updates concern finding minimal complements of views [8, 15], a problem quite different from the analysis of annotation propagation.

Contributions. To this end we identify a practical condition under which the analysis of annotation propagation becomes feasible. The condition, referred to as the *key-preservation* condition, requires that an SPJ view Q retains a key of every base relation involved in the definition of Q. In other words, the primary keys of all the base relations involved in Q are included as distinct attributes in the projection fields of Q. This is less restrictive than other proposals for restricting view definitions [11, 14], and many views for data transformation or integration found in practice can be naturally modified to be key preserving, by extending the projection-attribute list to include the primary keys.

We focus on fundamental issues in connection with key-preserving SPJ views: we give a full treatment of the decision problems associated with annotation propagation, and establish a variety of complexity results for these problems.

Our first contribution consists of complexity bounds for the analysis of annotation propagation for key-preserving SPJ views. These results tell us that the key preservation condition simplifies the annotation propagation problems studied in [5, 19]. We show that under the key preservation condition, the view side-effect problem and the annotation placement problem coincide, and moreover, the view and source side-effect problems (and thus the annotation placement problem) all become *tractable* for SPJ views.

Our second contribution is an investigation of the impact of *group updates* on the analysis of annotation propagation.

Here we allow the given view update ΔV to include multiple tuples to be deleted from the view, rather than a single tuple as stated above. In this setting the propagation analysis is to identify propagation of multiple annotations in the view. We show that group updates complicate the analysis: all the three problems become NP-hard for views defined in terms of join, *i.e.,* these problems are intractable for SJ, PJ and SPJ views while they remain tractable for SP views. To our knowledge these are among the first complexity results for group view updates.

Our third contribution consists of complexity results for the view and source side-effect problems when the given ΔV is a set of tuples to be *inserted* instead of deleted. The motivation for studying this is that one often wants to know, when new tuples along with annotations are inserted into the view, how the annotations should be propagated back to the source (aka. *feedback loop* [17]). We study these problems *both* in the presence *and* in the absence of the key preservation condition. We show that for PJ (and thus SPJ) views, the view and source side-effect problems are already NP-hard for single-tuple insertion, and these problems are in PTIME for SP and SJ views for group insertions, in the presence and in the absence of the key preservation condition. To our knowledge no previous work has established complexity results for these problems for view insertions.

Our main conclusions are: (a) key preservation simplifies the propagation analysis of annotations and view updates, to an extent; (b) group updates make our lives harder than a single-tuple update; and (c) view insertion does not behave as well as its deletion counterpart for key preserving views.

Taken together, these provide a dichotomy in the complexity of the analysis of annotation propagation for all subclasses of SPJ views, and for *single-tuple* and *group* view *insertions and deletions.* These complexity results are important not only for the propagation analysis of annotations; they are also useful for the study of classical view update problems, for which, to our knowledge, few complexity results have been established by previous work.

It should be mentioned that the key preservation condition was first studied in [7] for XML view updates. However, the decision problems investigated in [7] are quite different from the decision problems considered in this paper.

Organization. The remainder of the paper is organized as follows. Section 2 presents the key-preservation condition. Section 3 revisits the annotation propagation analysis of [5, 19] under the key preservation condition, and establishes complexity results for group deletions. Section 4 investigates these problems for view insertions. Related work is discussed in Section 5, followed by a conclusion in Section 6. All proofs of the complexity results are included in the paper.

2. Key Preservation

In this section we define the notion of key preservation and show that under this condition, the view side-effect problem and the annotation placement problem coincide.

SPJ **queries.** Let $\mathcal{R} = (R_1, \ldots, R_n)$ be a relational schema. An SPJ query on databases of $\mathcal{R}$ is an expression defined in terms of the selection (σ), projection (π), join ($\bowtie$) and renaming (δ) operators in the relational algebra, and with relation names $R_1, \ldots, R_n$ in $\mathcal{R}$ as well as constants. It is known that the class of satisfiable SPJ queries is equivalent to conjunctive queries as well as SPC queries defined in terms of the selection, projection and cross product ($\times$) operators (see, *e.g.,* [1]). Thus in the sequel we shall use SPJ and SPC interchangeably.

We also study various subclasses of SPJ, denoted by listing the operators supported: SP, SJ, and PJ (the renaming operator is included in all subclasses by default without listing it explicitly). For instance, PJ is the class of queries defined with the projection, join and renaming operators.

For example, the view given in Fig. 1(c) is a PJ view.

Key preservation. Consider $Q(R_1, \ldots, R_k)$, an SPJ query that takes the base relations $R_1, \ldots, R_k$ (repeats permitted) of $\mathcal{R}$ as input. From these base relation schemas and the definition of Q, it is straightforward to derive the schema of the output relation of Q, denoted by $\mathsf{schm}(Q)$.

We say that Q is *key-preserving* if all primary key attributes (with possible renaming) of each occurrence of the base relations involved in Q are included in the projection fields of Q.

Example 2.1: The query Q_1 (Fig. 1(c)) given in Example 1.1 can be extended such that it is key-preserving as follows: $Q_2 = \pi_{PCName,Conf,Topic}(PC \bowtie Conf)$. The corresponding view for $Q_2(D)$ is given in Fig. 1(d). □

Observe that queries without projection are always key-preserving.

We remark that key-preservation is far less restrictive than other conditions on view definitions proposed in earlier work [11, 14]. Indeed, these earlier proposals ask for joins to be defined on foreign keys, join attributes to be preserved in $\mathsf{schm}(Q)$, join to form a single tree, and/or for selection conditions not to include attribute comparison, etc.

The equivalence of the view side-effect problem and the annotation placement problem. For a key-preserving SPJ query $Q(R_1, \ldots, R_k)$, the two problems coincide. To see this, consider a source database D and the view $V = Q(D)$. For any tuple $t \in V$, and for each occurrence of each base relation R_i, t retains a key of the R_i relation. Hence one can identify a *unique* tuple t_i from each occurrence of the R_i relation, such that t in the view is constructed from these t_i's via Q. Thus as will be seen in Section 3, for the view side-effect problem, to delete t from V it suffices to remove a *single* t_i from some R_i relation. In other words, to remove a tuple ΔV from V it is always possible to find a *single* tuple to remove from the source. Equivalently, for the annotation placement problem, to annotate a single field in t, one can always identify a single t_i such that annotation at a field in t_i propagates to the field in t. This allows us to consider only the view side-effect and source side-effect problem in the sequel.

Example 2.2: The deletion analysis of Example 1.1 is simplified for the key-preserving view of Example 2.1. Consider the deletion $\Delta V = \{$(John, CIKM, DB)$\}$ from $Q_2(D)$. (1) View-side effect problem: Due to the key preservation property of Q_2, it is obvious that the deletion can be performed by deleting either (John, CIKM) from PC or (CIKM, DB, 30) from Conf. Here $\{$(John, CIKM)$\}$ is the minimal deletion with the minimal view side-effect. Note that the key preservation property also helps check the view side-effect by finding the occurrences of key values of deleted relation tuples in the view. (2) Source side-effect problem: Similar to (1), we can easily determine that the solution is either $\{$(John, CIKM)$\}$ or $\{$(CIKM, database, 30)$\}$. (3) Annotation placement problem: Under key preservation, the problem is the same as (1), and the solution is {(John, CIKM)}. □

Difference between insertions and deletions. To insert a tuple t into V, one can identify the key k_i of the tuple t_i that needs to be inserted into each occurrence of each R_i relation involved. As will be seen in Section 4, based on k_i one can either identify a tuple t_i already in the R_i relation with k_i as its key, or otherwise, construct a tuple t_i carrying k_i as its key and insert it into the R_i relation. Observe that while view-tuple deletion can always be carried out when side effect is allowed, in contrast, it is *not* always doable to insert a tuple into view in the presence of keys even if side effect is allowed, as illustrated below.

Example 2.3: Consider another key preserving view $Q_3 = PC \bowtie Conf$ in the setting of Example 1.1, and the insertion of the tuple (Kate, SIGMOD, DB, 35) into the view $Q_3(D)$. At first glance, it seems that this insertion can be carried out by inserting (Kate, SIGMOD) into table PC and (SIGMOD, DB, 35) into table Conf. However, in fact the insertion is not possible: the insertion (SIGMOD, DB, 35) has to be rejected since taken together with (SIGMOD, DB, 30) it violates the key in the relation Conf. In contrast, when modifications (i.e., a deletion followed by an insertion) are allowed, the insertion (Kate, SIGMOD, DB, 35) could be carried out by replacing (SIGMOD, DB, 30) with (SIGMOD, DB, 35) in table Conf. We do not consider modifications in this paper. □

3. Deletion Propagation

In this section we investigate the view and source side-effect problems for key-preserving SPJ views, in Sections 3.1 and 3.2, respectively, for single-tuple and group deletions.

3.1 The View Side-Effect Problem

Given a view deletion ΔV, the view side effect problem is to find a minimal set of source tuples to delete so that other view tuples (not in ΔV) deleted are minimized. The table below gives the complexity of the problem for various subclasses of SPJ queries for single-tuple or group deletions.

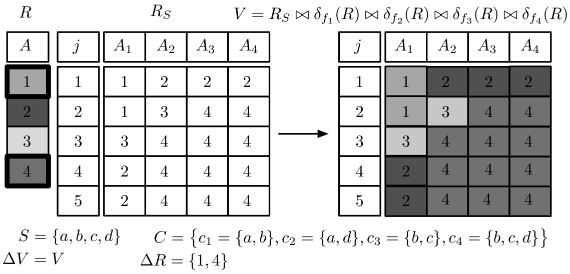
Query class	Complexity of view side-effect problem under key-preservation	
	single deletion	group deletions
SPJ (PJ, SJ)	PTIME	NP-hard
SP	PTIME	PTIME

It is known [5] that without key preservation the view side-effect problem for single deletion on a PJ view is NP-hard. In contrast, the problem becomes tractable for key preserving SPJ views. This shows that the key preservation condition simplifies the analysis of annotation propagation.

Theorem 3.1: *The view side-effect problem is in* PTIME *for single-tuple deletion for* SPJ *(and thus* PJ *and* SJ*) views under key preservation.* □

Proof: It suffices to give a proof for SPJ views. Let $\mathcal{R} = \{R_1,\ldots,R_n\}$ be a relational schema, Q a key-preserving SPJ query, D an instance of the schema $\mathcal{R}$, and ΔV consist of the single tuple t to be deleted from the view $Q(D)$.

Due to the key-preservation, we can associate with t (necessarily unique) tuples s_i in the base relations R_i appearing in Q, such that s_i and t have the same key for this relation.



R

A
1
2
3
4

R_S

j	A_1	A_2	A_3	A_4
1	1	2	2	2
2	1	3	4	4
3	3	4	4	4
4	2	4	4	4
5	2	4	4	4

$V = R_S \bowtie \delta_{f_1}(R) \bowtie \delta_{f_2}(R) \bowtie \delta_{f_3}(R) \bowtie \delta_{f_4}(R)$

j	A_1	A_2	A_3	A_4
1	1	2	2	2
2	1	3	4	4
3	3	4	4	4
4	2	4	4	4
5	2	4	4	4

$S = \{a, b, c, d\}$ $C = \{c_1 = \{a, b\}, c_2 = \{a, d\}, c_3 = \{b, c\}, c_4 = \{b, c, d\}\}$

$\Delta V = V$ $\Delta R = \{1, 4\}$

Figure 2: Illustration of the proof of Theorem 3.3.

In order to delete t from $Q(D)$ is suffices to delete a single such s_i from its base relation R_i. Any such deletion obviously results in a ΔD of minimal size since ΔD consists of a single tuple only, as illustrated in Example 2.2.

For ΔD to be a solution for the view side-effect problem, we need to find tuple s_i that leads to the minimal number of side-effects. Let S_i be the set of tuples in $Q(D) \setminus \{t\}$ carrying the key of s_i (in R_i). Note that computing S_i requires only a linear scan over the view $Q(D)$. Clearly, the size of S_i determines the number of side-effects obtained when choosing $\Delta D = \{s_i\}$. Let s be the tuple s_i such that its corresponding S_i is of minimal size. Then $\Delta D = \{s\}$ is a solution. It is clear that s can be found in PTIME. □

Theorem 3.2: *The view side-effect problem is in* PTIME *for group deletions for* SP *views under key preservation.* □

Proof: Let ΔV be a group deletion. It is easy to see that we can apply a simple modification of the algorithm given in the proof of Theorem 3.1 for each tuple in ΔV independently. Indeed, for each tuple $t \in \Delta V$ we have to delete a single distinct tuple s_t in the base relation appearing in the SP query Q. Let ΔD consist of the base relation tuples s_t for $t \in \Delta V$. Clearly, ΔD is of minimal size. Due to the key-preservation of Q, the deletion of ΔD from D will only delete the tuples in ΔV from the view $Q(D)$. Hence, ΔD is indeed a solution for the view side-effect problem. □

The problem, however, becomes NP-hard if we consider group deletions. This tells us that group updates may complicate the analysis of annotation propagation. It should be remarked that the complexity of group view deletions is not considered in [5, 19].

Theorem 3.3: *The view side-effect problem is* NP*-hard for group deletions for* SPJ*,* PJ *and* SJ *views under key preservation.* □

Proof: It suffices to show that the problem is NP-hard for views defined in terms of join only, by reduction from the minimal set cover problem. An instance of the minimal set cover problem consists of a collection C of subsets of a finite set S; it is to find a subset $C' \subseteq C$ such that every element in S belongs to at least one member of C' and moreover, $|C'|$ is minimal. This problem is NP-complete (cf. [12]).

Given S and C, we define an instance of the view side-effect problem. Let $S = \{x_i \mid i \in [1, n]\}$ and $C = \{c_j \mid j \in [1, k]\}$. We construct two base tables R and R_S, a join view and a group view deletion, as follows.

Source database. We define two base relations R and R_S.

- $R(A)$, where A is the key and is to hold a number in $[1, k]$.

Initially, $R(A)$ contains $k = |C|$ tuples $\{(1), (2), \ldots, (k)\}$ that represent the index of k subsets.

- $R_S(j, A_1, \ldots, A_k)$, where all the columns are the key. We encode each element in S with tuples in R_S as follows. For each x_i in S, let T_i be the collection of all the subsets in C that contain x_i. We assume w.l.o.g. that $T_i \neq \emptyset$ (otherwise there is no solution for the minimal set cover problem). Enumerate the elements of T_i as $(c_{i^1}, \ldots, c_{i^{n_i}})$. We generate a list of size k from T_i, $L_i = < i^1, \ldots, i^{n_i}, \ldots, i^{n_i} >$, by replacing c_{ij} with its index i^j and appending $(k - |T_i|)$ i^{n_i}'s at the end of the list (to make the size of the list to be k).

If $|S| > k$, then we generate $|S|$ tuples by adding one number in $[1, |S|]$ at the beginning of each list L_i. Otherwise, we generate $k+1$ tuples by adding one number in $[1, |S|]$ at the beginning of each list L_i and generate $k + 1 - |S|$ tuples by adding numbers $[|S| + 1, k + 1]$ to L_n. Thus table R_S initially contains $\ell = \max\{|S|, k + 1\}$ tuples.

Let the database instance D be the collection of all tuples defined above. The construction is illustrated in Fig. 2 for $S = \{a, b, c, d\}$ and $C = \{c_1 = \{a, b\}, c_2 = \{a, d\}, c_3 = \{b, c\}, c_4 = \{b, c, d\}\}$.

View. We define a view in terms of query $Q = R_S \bowtie \delta_{f_1}(R) \bowtie \cdots \bowtie \delta_{f_k}(R)$, where δ_{f_i} renames A in R to A_i. Initially, $Q(D)$ consists of ℓ view tuples, which are the same as those in the relation R_S.

Obviously, the view defined as above is key-preserving.

- View deletion. The group deletion ΔV is to remove all tuples in the view $Q(D)$.

The view side-effect problem is to find a smallest set of the tuples from R and R_S so that ΔV is deleted without side-effect. For example, suppose that we want to delete all tuples in the view V shown in Fig. 2. For each view tuple t, we indicate with colors which tuples (or c_i's) in R should be deleted in order to remove t from V. When all tuples are to be removed from V, i.e., $\Delta V = V$, then clearly deleting 1 and 4 from R achieves this goal (each tuple in V contains either 1 or 4). Hence, $\Delta R = \{1, 4\}$ and $C' = \{c_1, c_4\}$ is a minimal cover of S.

More formally, we next verify that the construction above is indeed a reduction from the minimum set cover problem. First suppose that C' is a minimal cover of S. We define ΔD such that it consists of deletion of tuples $\{(i_1), \ldots, (i_{|C'|})\}$ from R, where i_j is the index of subset $c_j \in C'$. In order to delete a tuple t in ΔV, we delete either $t[R_S]$ (its component in R_S) or one of its components in R. Since C' is a cover of S, at least one of components of t in R is in ΔD. Hence, it is clear that $Q(D - \Delta D) = Q(D) - \Delta V = \emptyset$. Furthermore, ΔD is minimal since (1) although deleting all the ℓ tuples from table R_S suffices to delete ΔV, it is not a minimal solution since $|C'| \leq k$ (and by construction, $\ell > k$), and (2) C' is a minimal cover of S. Conversely, suppose that ΔD is a solution to the view side-effect problem. Then as discussed above ΔD will be only composed of tuples in R. Let C' be the subset of C such that an element c_j of C is in C' if and only if ΔD involves deletion of the tuple (j) from relation R. To see that C' is a cover of S, note that $Q(D - \Delta D) = Q(D) - \Delta V = \emptyset$, and thus for each $x_i \in S$, some set c_{ij} is in C'. Moreover, C' is minimal since ΔD is minimal. □

3.2 The Source Side-Effect Problem

Given a view deletion ΔV, the source side-effect problem is to find a minimal set of source tuples to be deleted so that the view tuples in ΔV are deleted. Although the source side-effect problem relaxes the requirement of minimizing view side-effects in the view side-effect problem, unfortunately the problem does not become easier, and the complexity remains the same as its view side-effect counterpart. The table below gives the complexity of determining the minimum source deletions for various subclasses of SPJ queries for single-tuple or group deletions.

Query class	Complexity of source side effect problem under key-preservation	
	single deletion	group deletions
SPJ (SJ, PJ)	PTIME	NP-hard
SP	PTIME	PTIME

It has been shown in [5] that the source side-effect problem is already NP-hard for single deletion for PJ view. The problem for single deletion becomes polynomial-time solvable when the key preservation condition is imposed. This again verifies our observation that the key-preservation condition makes our lives easier.

Theorem 3.4: *The source side-effect problem is in* PTIME *for single-tuple deletion for* PJ, SJ *and* SPJ *views under key preservation.* □

Proof: It suffices to give a proof for SPJ views. We remark that the PTIME algorithm presented in the proof of Theorem 3.1 already gives a solution for the source side-effect problem. Indeed, it is observed there that the computed update ΔD is of minimal size. We can therefore use the same algorithm for the source side-effect problem, except that we do not have to perform the steps for selecting the update which minimizes the number of view side-effects. □

In fact the proof of Theorem 3.2 also works for the source side-effect problem. Hence, we have the following:

Theorem 3.5: *The source side-effect problem is in* PTIME *for group deletions for* SP *views under key preservation.* □

However, the problem for group deletions remains NP-hard, as its view side-effect counterpart. Again this problem has not been considered by previous work.

Theorem 3.6: *The source side-effect problem is* NP*-hard for group deletions for* PJ*,* SJ *and* SPJ *views under key preservation.* □

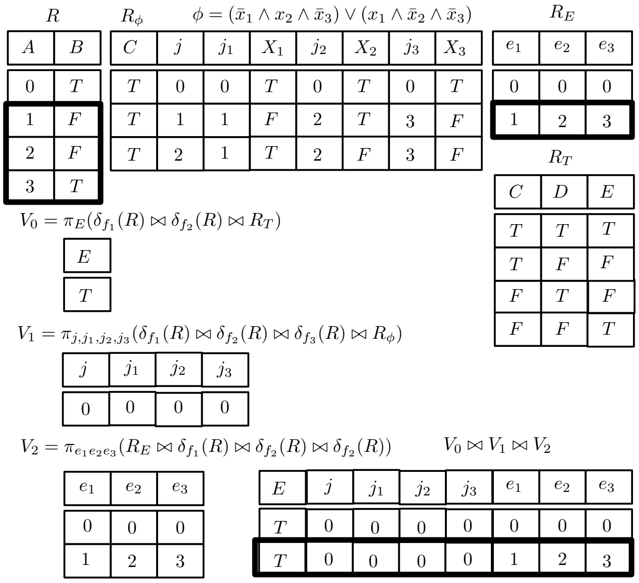
Proof: The proof of Theorem 3.3 is applicable here. □

4. Insertion Propagation

We next investigate the view and source side-effect problems for insertions, *i.e.,* when the view update ΔV consists of tuples to be inserted into the view. We study these two problems in Section 4.1 and 4.2 respectively, in the presence and in the absence of the key preservation condition, and for both single-tuple or group insertions.

4.1 The View Side-Effect Problem

We first study the view side-effect problem in the absence of key-preserving condition: given a source database D, a query Q, the view $V = Q(D)$ and a set ΔV of tuples, it is



R

A	B
0	T
1	F
2	F
3	T

R_ϕ $\quad \phi = (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3) \vee (x_1 \wedge \bar{x}_2 \wedge \bar{x}_3)$

C	j	j_1	X_1	j_2	X_2	j_3	X_3
T	0	0	T	0	T	0	T
T	1	1	F	2	T	3	F
T	2	1	T	2	F	3	F

R_E

e_1	e_2	e_3
0	0	0
1	2	3

R_T

C	D	E
T	T	T
T	F	F
F	T	F
F	F	T

$V_0 = \pi_E(\delta_{f_1}(R) \bowtie \delta_{f_2}(R) \bowtie R_T)$

E
T

$V_1 = \pi_{j,j_1,j_2,j_3}(\delta_{f_1}(R) \bowtie \delta_{f_2}(R) \bowtie \delta_{f_3}(R) \bowtie R_\phi)$

j	j_1	j_2	j_3
0	0	0	0

$V_2 = \pi_{e_1 e_2 e_3}(R_E \bowtie \delta_{f_1}(R) \bowtie \delta_{f_2}(R) \bowtie \delta_{f_2}(R))$

e_1	e_2	e_3
0	0	0
1	2	3

$V_0 \bowtie V_1 \bowtie V_2$

E	j	j_1	j_2	j_3	e_1	e_2	e_3
T	0	0	0	0	0	0	0
T	0	0	0	0	1	2	3

Figure 3: Illustration of the NP-hardness proof of Theorem 4.1.

to find a minimal set ΔD of tuples such that $Q(D) - Q(D \oplus \Delta D)$ contains ΔV and is minimal, *i.e.,* the insertion of ΔD into the source D gets ΔV into the view while incurring minimal side effect on the view. This problem turns out to be nontrivial: it is already intractable when Q is a PJ (and thus SPJ) view, even if ΔV consists of a single tuple.

Theorem 4.1: *The view side-effect problem is NP-hard for* PJ *views and single-tuple insertion, when the* PJ *views are not necessarily key preserving.* □

Proof: We prove the NP-hardness by reduction from the non-tautology problem. An instance of the latter problem is $\phi = C_1 \vee \cdots \vee C_n$, where all the variables in ϕ are $x_1, \ldots, x_k$, C_j is of the form $l_{j_1} \wedge l_{j_2} \wedge l_{j_3}$, and l_{i_j} is either x_s or $\bar{x}_s$, $s \in [1, k]$. The problem is to determine whether there is a truth assignment such that ϕ is false, *i.e.,* ϕ is not valid. This problem is known to be NP-complete (cf. [12]).

Given ϕ, we define a source database D, a PJ view Q, and a single tuple ΔV to be inserted into the view $V = Q(D)$, such that ϕ is not valid iff there exists a minimal ΔD that is *side-effect free*, *i.e.,* $Q(\Delta D \oplus D) = V \oplus \Delta V$.

Source D**.** The database consists of four base relations, R, R_ϕ, R_E, and R_T defined as follows.

- $R(A, B)$, where intuitively, A is to hold a number in $[1, k]$ encoding a variable, and B is a truth value (T or F). That is, $R(A, B)$ is to encode a truth assignment for ϕ. Initially $R(A, B)$ consists of a single tuple $(0, T)$.
- $R_\phi(C, j, j_1, X_1, j_2, X_2, j_3, X_3)$ in which for each $C_j = l_{j_1} \wedge l_{j_2} \wedge l_{j_3}$, there is a tuple $(T, j, l_{j_1}, X_1, l_{j_2}, X_2, l_{j_3}, X_3)$ in R_ϕ such that l_{j_i} is s if $l_{j_i} = x_s$ or $l_{j_i} = \bar{x}_s$, X_i is T if $l_{j_i} = x_s$, and X_i is F if $l_{j_i} = \bar{x}_s$. Each of these tuples codes a clause in ϕ. A special tuple $(T, 0, 0, T, 0, T, 0, T)$ is also in R_ϕ.
- $R_E(e_1, e_2, \ldots, e_k)$, in which e_i is to code i in $[1, k]$. Initially, R_E consists of a single special tuple $(0, \ldots, 0)$.
- $R_T(C, D, E)$ consisting of the four tuples (T, T, T), (T, F, F), (F, T, F), and (F, F, T). That is, the E-attribute is T if the other attributes are equal, and is F otherwise.

View. We define a PJ query $Q = V_0 \bowtie V_1 \bowtie V_2$ as follows:

- $V_0 = \pi_E(\delta_{f_1}(R) \bowtie \delta_{f_2}(R) \bowtie R_T)$, where δ_{f_1} renames B to C and δ_{f_2} renames B to D.
- $V_1 = \pi_{j,j_1,j_2,j_3}(\delta_{f_1}(R) \bowtie \delta_{f_2}(R_2) \bowtie \delta_{f_3}(R_3) \bowtie R_\phi)$, where δ_{f_i} renames A to j_i and B to X_i for $i = 1, 2, 3$. Intuitively, C holds if and only if one of the C_j 's is true.
- $V_2 = \pi_{e_1,e_2,\ldots,e_k}(R_E \bowtie \delta_{f_1}(R) \bowtie \cdots \bowtie \delta_{f_k}(R))$, where δ_{f_i} renames A to e_i for $i \in [1, k]$.

Initially $V = Q(D)$ has a single tuple $(T, 0, \ldots, 0)$.

View insert. We define ΔV to consist of a single tuple $(T, 0, 0, 0, 0, 1, \ldots, k)$ into V.

The construction above is illustrated in Fig. 3. In this figure, we have depicted the base relations R, R_ϕ, R_E and R_T as well as the intermediate view relations V_0, V_1 and V_2. The final view $V_0 \bowtie V_1 \bowtie V_2$ is shown at the bottom right. The tuple inserted in the view, as well as the tuples to be inserted in the base relations R and R_E are indicated by the bold rectangles. As we will show formally below, the key observation is that a zero side-effect update exists as long as V_1 only contains the initial tuple $(0, 0, 0, 0)$. This in its turn is equivalent to saying that ϕ is a non-tautology.

We next verify that there is a minimal, side-effect free ΔD iff ϕ is not a tautology. First, if ϕ is not a tautology, then there is a truth assignment μ such that ϕ is false, and thus C_j is false w.r.t. μ for all $j \in [1, n]$. We define ΔD based on μ as follows: for $i \in [1, k]$, we insert (i, T) into $R(A, B)$ iff $\mu(x_i) = T$, and insert (i, F) iff $\mu(x_i) = F$. We also insert $(1, \ldots, k)$ into R_E. Then obviously $Q(\Delta D \oplus D) = V \oplus \Delta V$. To see that ΔD is minimal, note that for any side-effect free ΔD, ΔD must contain k tuples of the form (i, X_i) to be inserted into R for $i \in [1, k]$, where X_i is either T or F, as well as a tuple $(1, \ldots, k)$ to be inserted into R_E. Thus the ΔD given above is already minimal.

Conversely, suppose that there is a minimal, side-effect free ΔD. Then again ΔD must insert $(1, \ldots, k)$ into R_E; in addition ΔD contains a *unique* tuple of the form (i, X) to be inserted into the base relation R for each $i \in [1, k]$, where X is either T or F. To see why (i, X) is unique, note that if for some i both (i, T) and (i, F) are in ΔD, then a tuple of the form $(F, 0, 0, 0, 0, 1, \ldots, k)$ would also be in $Q(D \oplus \Delta D)$ by the definition of Q (and in particular V_0 and R_T), a contradiction. Hence the instance of R is a valid truth assignment for ϕ. Since ΔD is side-effect free, V_1 will remain $(0, 0, 0, 0)$ after ΔD is inserted, *i.e.,* C_j will remain false for each $j \in [1, n]$. Thus ϕ is not a tautology. □

Worse, key preservation does not make our lives easier:

Theorem 4.2: *The view side-effect problem is NP-hard for key preserving* PJ *views and single-tuple insertion.* □

Proof: The proof is similar to that of Theorem 4.1, by reduction from the non-tautology problem.

Source D**.** The database consists of three base relations, R, R_ϕ and R_E. Here $R(A, B)$ and $R_E(e_1, e_2, \ldots, e_k)$ are the same as defined in the proof of Theorem 4.1, with A and $e_1, \ldots, e_k$ as the key of R and R_E, respectively. The relation R_ϕ is defined to be $R_\phi(j, j_1, X_1, j_2, X_2, j_3, X_3)$, in which j is the key, and $j, j_1, X_1, j_2, X_2, j_3, X_3$ are the same as given in the proof of Theorem 4.1.

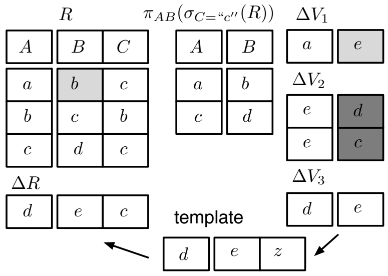


Figure 4: Illustration of the PTIME **algorithm in the proof of Theorem 4.3.**

View. We define a PJ query $Q = V_1 \times V_2$, where V_1 and V_2 are the same as given in the proof of Theorem 4.1. Initially $V = Q(D)$ has a single tuple $(0, \ldots, 0)$ (k+4 0's). It is easy to verify that Q is key preserving.

View insert. We define ΔV to consist of a single tuple $(0, 0, 0, 0, 1, \ldots, k)$ to be inserted into V.

We next verify that there is a minimal, side-effect free ΔD iff ϕ is not a tautology. When ϕ is not a tautology, a minimal, side-effect free ΔD can be constructed as described in the proof of Theorem 4.1. Conversely, suppose that there is a minimal, side-effect free ΔD. Then ΔD must insert $(1, \ldots, k)$ into R_E, and inserts a *unique* tuple of the form (i, X) into R for each $i \in [1, k]$, where X is either T or F. Note that if for some i both (i, T) and (i, F) are in ΔD, then these tuples violate the constraint that A is a key of R. Thus the instance of R is a truth assignment for ϕ. The rest of the argument is the same as that of Theorem 4.1. □

The good news is that the problem becomes tractable for SP and SJ views and for *group insertions*, in the presence and in the absence of key preservation.

Theorem 4.3: *The view side-effect problem is in* PTIME *for (a)* SP *views and (b)* SJ *views, for group insertions, no matter whether the views are key-preserving or not.* □

Proof: The proof is constructive. For each of the cases we provide a PTIME algorithm which either halts (indicating that no solution exists) or outputs a solution for the view side-effect problem. Note that some insertions on the view may not be doable, as demonstrated in Example 2.3.

Key-preserving SP **views.** Let D be a source database, Q a key-preserving *SP* query, and ΔV be a group update consisting of insertions only. We may assume that Q is of the form $\pi_{B_1,\ldots,B_n}(\sigma_C(\delta_f(R)))$ where R is one of the relations in the schema $\mathcal{R}$. If we denote by $A_1, \ldots, A_k$ the primary key attributes of R, w.l.o.g., then by the key-preservation of Q it is the case that $\{A_1, \ldots, A_k\} \subseteq \{B_1, \ldots, B_n\}$.

For each tuple $t \in \Delta V$, we define its *tuple template* $\hat{t} = (\vec{a}, \vec{b}, \vec{z})$, where $\vec{a} = t[A_1, \ldots, A_k]$, $\vec{b}$ consists of the constants in the remaining attributes in t, and finally, $\vec{z}$ consists of distinct variables for each remaining attribute in $\mathsf{schm}(R)$.

The PTIME algorithm for the view side-effect problem performs the following steps. We illustrate some of them in Fig. 4 for the base relation R (with A as its key), SP view $Q = \pi_{AB}(\sigma_{C=“c”}(R))$ and updates ΔV_1, ΔV_2 and ΔV_3.

First, we check whether ΔV contains different tuples with the same key attributes. If so, then clearly no solution for the problem exists, and the algorithm halts. This happens, e.g., for ΔV_2 in Fig. 4 (the gray color indicates the conflict – it is impossible to insert two distinct tuples with the same key e).

Otherwise, the algorithm continues by testing for each tuple $t \in \Delta V$, whether there already exists a tuple s in R with the same key attributes, i.e., $s[A_1, \ldots, A_k] = \vec{a}$. If this happens, then $\hat{t}$ should be equal to s. If one of the $\vec{b}$ attributes of t differs from those in s, then no solution exists and the algorithms halts. This happens e.g., for ΔV_1 in Fig. 4 (the gray color indicates the conflict).

Moreover, in order to get t inserted into the view, a necessary condition is that $\sigma_C(s)$ holds. If not, then no solution for the group update can be found and, again, the algorithm halts. Otherwise, we can safely remove all t from ΔV whose key already appears in R.

Finally, for each remaining tuple t in ΔV we need to instantiate the variables in its template $\hat{t}$. More specifically, we need to instantiate these variable such that the resulting tuple (this will be a tuple to be added to R) satisfies the selection condition C in Q. Because Q does not contain joins, we can treat each tuple in ΔV independently.

We recall that C is a conjunction of equalities of the form $x = y$, where x, y are either attributes or constants. By plugging in C the constants available in $\hat{t}$, *i.e.,* those in $\vec{a}$ and $\vec{b}$, we obtain a new conjunction C' (with possibly less variables). By constructing a dependency graph G between the constants and variables in C' and computing its transitive closure G', one can then easily check whether a desired instantiation of the variables exists. Indeed, if there exists an edge $(a, b) \in G'$ with a, b two different constants, then no instantiation exists. We say that C' is conflicting. Consequently, in this case no solution of the view side-effect problem exists and the algorithm halts. Otherwise, one assigns to all the variables in the same connected component in G' the same constant value (i.e., the value of the unique constant in this component, or an arbitrary one if the connected component consists of variables only). Variables not appearing in C can be instantiated arbitrarily. The resulting tuple is then added to ΔD.

The algorithm successfully computes a solution for the view side-effect problem if for each tuple in ΔV (modulo the ones whose key already appeared in R) a tuple is added to ΔD. In all other cases, no solution exists. For example, in Fig. 4 a solution for ΔV_3 exists. First, the tuple in ΔV_3 is expanded to a template (introducing the variable z), then this variable is instantiated using the condition $C = “c”$ of the selection predicate of Q.

We remark that in case a solution exists, ΔD computed by the above algorithm is of minimal size. Indeed, for each new key in ΔV, a single tuple with this key is added to ΔD. Since Q is key-preserving, this is the minimal number of tuples required for any solution. Moreover, it is easy to see that this solution is side-effect free, and hence is also minimal on the view side.

The algorithm runs clearly in polynomial time.

Arbitrary SP **views.** Let us now drop the key-preserving condition on the view Q. We use the same approach as in the key-preserving case, except that we do not have to check for conflicting keys. However, even in the absence of key-preservation, the update to the view cannot always be performed successfully. As we will see below, a necessary condition is that the tuples to be inserted in the view can be extended to tuples in the base relation satisfying the selection condition C.

Indeed, for each tuple t in ΔV to be inserted in the SP view Q, we create a tuple template $\hat{t} = (t, \vec{z})$ where $\vec{z}$ consists of variables for the attributes in $\mathsf{schm}(R) \setminus \{B_1, \ldots, B_m\}$.

We then proceed by checking for each template $\hat{t}$ whether there exists already tuples s in R such that (i) $\hat{t}$ and s agree on $\mathsf{schm}(Q)$; and (ii) $\sigma_C(s)$ holds. If there exists such a tuple s, then $\hat{t}$ is set to s, and we can safely remove t from ΔV to be inserted (it will automatically belong to the view). Otherwise, if $\sigma_C(s)$ does not hold or no such s exists, then we need to instantiate the variables $\vec{z}$ in $\hat{t}$ in such a way that for the resulting tuple t', $\sigma_C(t')$ holds. The tuples t' will make up the update ΔD to the database.

Testing whether such an instantiation exists can be done similarly as in the key-preserving case above. If this can be done successfully for each template, then ΔD will be a solution for the view side-effect problem. In fact, this solution does not introduce any side effects. Also, ΔD is minimal, because only the necessary tuples are inserted in D (we use existing tuples where possible). The algorithm runs clearly in polynomial time.

Key-preserving SJ **views.** We may assume that $Q = \sigma_C(\delta_{f_1}(R_1) \bowtie \cdots \bowtie \delta_{f_k}(R_k))$.

The PTIME algorithm consists of the following steps. Because Q does not contain projections, we can derive from each tuple t in ΔV and for each relation R_i ($i \in [1, k]$) in Q a tuple $\hat{t}_i = (\vec{a}_i, \vec{b}_i)$ over the attributes of R_i. We then check for each $t \in \Delta V$ whether $(\delta_{f_1}(\hat{t}_1) \bowtie \cdots \bowtie \delta_{f_k}(\hat{t}_k))$ satisfies the selection condition C. If not, then no solution exists and the algorithm halts. Otherwise, it continues.

Similar to the cases above, we check for each $\hat{t}_i$ whether there exists already an s_i in R_i having the same key $\vec{a}_i$. If this is the case, $\hat{t}_i$ should be equal to s_i. If there exists a $\hat{t}_i$ for which this does not hold, then no solution exists and the algorithm stops. Otherwise, the algorithm continues.

Denote by ΔR_i the set of tuple $\hat{t}_i$ for which no tuple in R_i exists with the same key. We check whether ΔR_i contains two different tuples having the same key. If such tuples exists, no solution can be found and the algorithm halts. Otherwise, we define ΔD to be $\{\Delta R_1, \ldots, \Delta R_k\}$.

We remark that ΔD is the minimal solution. Indeed, in each instance R_i, the same number of tuples as the number of new keys for R_i present in ΔV are inserted. This is the minimum requirement for any solution, so we cannot do it with less updates to D. Because we have no choice (due to the lack of projections) about which tuples to insert, the number of side-effects created is necessarily minimal. The algorithm clearly runs in polynomial time.

Arbitrary SJ **views.** Let us now drop the condition of key-preservation. Again, because Q does not have a projection, we can associate with each tuple t in ΔV and each relation R_i in Q, a unique tuple $\hat{t}_i$ over the attributes of R_i. We simply check if $\hat{t}_i$ already appears in the instance R_i. If not, then we add $\hat{t}_i$ to ΔR_i. Assuming that $\sigma_C(\delta_{f_1}(\hat{t}_1) \bowtie \cdots \bowtie \delta_{f_k}(\hat{t}_k))$ holds for each tuple t in ΔV, we define $\Delta D = \{\Delta R_1, \ldots, \Delta R_k\}$. For the same reasons as in the key-preserving case, this is a solution of the view side-effect problem. □

These results are summarized in Table below.

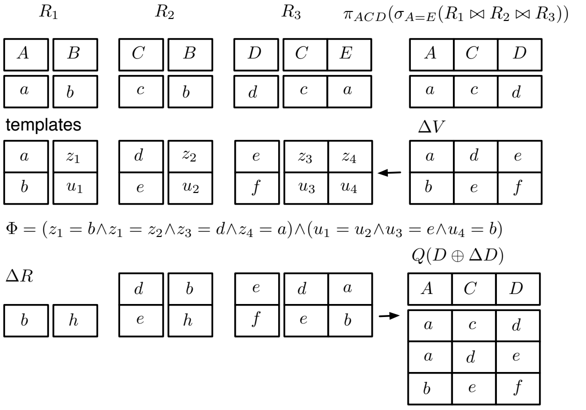


Figure 5: Illustration of the PTIME **algorithm in the proof of Theorem 4.4.**

Query class	Complexity of view side-effect problem	
	key-preservation	arbitrary
PJ, SPJ	NP-hard (1 tuple)	NP-hard (1 tuple)
SP	PTIME (group)	PTIME (group)
SJ	PTIME (group)	PTIME (group)

4.2 The Source Side-Effect Problem

Finally we study the source side-effect problem for insertions: given a source database D, a query Q, the view $V = Q(D)$ and a set ΔV of tuples, it is to find a minimal set ΔD of tuples such that $Q(D) - Q(D \oplus \Delta D)$ contains ΔV, *i.e.,* we want to find a *minimal* set of tuples to insert into the source such that the insertion will get ΔV into the view, regardless of side effects on the view. We study this problem for both key-preserving and general view queries Q in SPJ and its subclasses. The main results of this section are summarized in the Table below.

Query class	Complexity of source side-effect problem	
	key-preservation	arbitrary
PJ, SPJ	PTIME (group)	NP-hard (1 tuple)
SP	PTIME (group)	PTIME (group)
SJ	PTIME (group)	PTIME (group)

We first present the tractable results.

Theorem 4.4: *The source side-effect problem is in* PTIME *for (a)* SP *views and (b)* SJ *views, for group insertions, no matter whether the views are key-preserving or not. It is also in* PTIME *for (c) key-preserving* SPJ *views.* □

Proof: The proofs of cases (a) and (b) are similar to those of (a) and (b) for the view side-effect problem (see Theorem 4.3). To show (c), it requires a bit more effort (recall that the view side-effect problem for this case is intractable, by Theorem 4.2).

Arbitrary SP **and** SJ **views.** The update ΔD returned by the PTIME algorithm for the view side-effect problem is a solution of the source side-effect problem. Indeed, only the necessary tuples are inserted by those algorithms given for Theorem 4.3. As a result, ΔD is of minimal size.

Key-preserving SPJ **views.** Consider a key-preserving SPJ query $Q = \pi_{B_1,\ldots,B_m}\sigma_C(\delta_{f_1}(\hat{t}_1) \bowtie \cdots \bowtie \delta_{f_k}(\hat{t}_k))$.

As before, for each tuple t in ΔV and each relation R_i in Q, we associate a template $\hat{t}_i = (\vec{a}_i, \vec{b}_i, \vec{z}_i)$.

The PTIME algorithm first checks for incompatible templates: (i) there should be no two different templates with the same key; or (ii) templates $\hat{t}_i$ with the same key as an existing tuple s_i in R_i, but which differ in another attribute. As before, we use these existing tuples s_i (if they exist) to instantiate the variables in $\hat{t}_i$.

Fig. 5 illustrates the algorithm for the base relations R_1, R_2, R_3, with keys A, C and D, respectively, the key-preserving SPJ view $Q = \pi_{ACD}(\sigma_{A=E}(R_1 \bowtie R_2 \bowtie R_3))$, and view update ΔV. We also depict the templates for each tuple in ΔV.

If no conflicts are found, we define ΔR_i to be the set of templates $\hat{t}_i$ (note that some of then will have no variables anymore, which means that they are already in R_i).

It remains to instantiate the variables in the templates. For this, we proceed as follows: for each tuple t in ΔV we compute a conjunctive formula ϕ_t representing the selection and join condition to hold on $\hat{t}_1 \times \hat{t}_2 \times \cdots \times \hat{t}_k$ such that it will generate t in the view. The formula ϕ_t consists of a conjuncts of equations of the form $x = y$ where x and y are either variables or constants in $\hat{t}_i$, $i \in [1, k]$. We group together all conjunctions ϕ_t into a big conjunction $\Phi = \bigwedge_{t \in \Delta V} \phi_t$ and check (in a similar way as in case (a) of Theorem 4.2) whether there exists an instantiation of the variables which makes Φ true.

Since we are not concerned about the size of the side effects, we do not have to take into account constraints regarding existing constants in the database (this is in contrast with the NP-hardness proof in Theorem 4.1). Hence, if an instantiation exists, we can complete the templates into tuples which populate the update set ΔR_i. Finally, we define $\Delta D = \{\Delta R_1, \ldots, \Delta R_k\}$. For example, in Fig. 5 we show Φ and a possible instantiation of the variables. The updated view is shown on the bottom right. In this case no side-effects were created (while in general, side effect cannot be avoided).

We remark that ΔD is a solution and is also minimal. Indeed, at most a single tuple for each new key in tuples in ΔV is added, a necessary requirement for any solution. □

We next show that in the absence of the key preservation condition, the source side-effect problem becomes intractable for PJ (and thus SPJ) views and single-tuple insertion. Contrast this with Theorem 4.4 (b) and (c). Taken together, these tell us that the key preservation condition may also simplify the analysis of annotation propagation when view insertions are concerned.

Theorem 4.5: *The source side-effect problem is* NP*-hard for* PJ *views and single-tuple insertion, when the* PJ *views are not necessarily key preserving.* □

Proof: We prove the intractability by reduction from the minimal set cover problem (see the proof of Theorem 3.3 for the statement of this problem). It is known that this problem is NP-complete [12].

Given S and C, we define an instance of the source side-effect problem. Let $S = \{x_i \mid i \in [1, n]\}$ and $C = \{c_j \mid j \in [1, k]\}$. We construct a source database D, a PJ view Q, the view $V = Q(D)$, and a single tuple ΔV to be inserted into V. We show that we can find a minimal cover C' of S iff there exists a minimal set ΔD of tuples such that $Q(D \oplus \Delta D)$ contains ΔV.

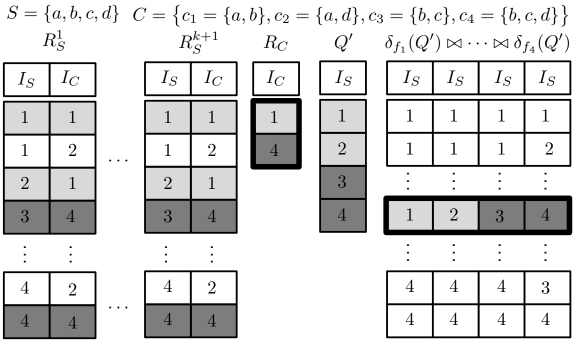

$S = \{a,b,c,d\}$ $C = \{c_1 = \{a,b\}, c_2 = \{a,d\}, c_3 = \{b,c\}, c_4 = \{b,c,d\}\}$
R_S^1
R_S^{k+1}
R_C
Q'
$\delta_{f_1}(Q') \bowtie \cdots \bowtie \delta_{f_4}(Q')$
I_S I_C
1 1
1 2
2 1
3 4
4 2
4 4
I_C
1
4
I_S
1
2
3
4
I_S I_S I_S I_S
1 1 1 1
1 1 1 2
1 2 3 4
4 4 4 3
4 4 4 4
...

Figure 6: Illustration of the NP-hardness proof of Theorem 4.5.

Source database. We define $k+2$ relations R_S^i, $i \in [1, k+1]$ and a relation R_C.

- $R_S^i(I_S, I_C)$, for $i \in [1, k+1]$, where I_S and I_C range over $[1, n]$ and $[1, k]$, respectively. Initially, (i, j) is in D iff $x_i \in c_j$, *i.e.,* (i, j) indicates whether or not the element x_i of S is in the subset c_j in the collection C. As will be seen shortly, we keep $k + 1$ copies of the $R_S^i(I_S, I_C)$ relation to prevent insertions into any of these relations.

- $R_C(I_C)$ is to hold the elements of C to be picked for covering S. In other words, $R_C(I_C)$ is to represent a cover C' (after it is picked) such that (j) is in D iff $c_j \in C'$, for $j \in [1, k]$. Initially R_C in D is empty, *i.e.,* no element of C is picked yet.

View. We define a PJ view $Q = \delta_{f_1}(Q') \bowtie \ldots \bowtie \delta_{f_n}(Q')$, where Q' is $\pi_{I_S}(R_S^1 \bowtie R_S^2 \bowtie \cdots R_S^{k+1} \bowtie R_C)$, and δ_{f_i} renames I_S to a distinct name I_S^i in order to conduct cross product (rather than natural join). A tuple in the view is a n-vector $(a_1, \ldots, a_n)$, where $a_i \in [1, n]$. Initially, $V = Q(D)$ is empty. Note that Q is not key preserving.

View insertion. The tuple ΔV is $(1, \ldots, n)$. It is to force a cover C' to be picked, *i.e.,* every element x_i in S is to be covered by some subset c_j in C'.

The reduction is illustrated in Fig. 6 for $S = \{a, b, c, d\}$ and $C = \{c_1 = \{a, b\}, c_2 = \{a, d\}, c_3 = \{b, c\}, c_4 = \{b, c, d\}\}$. The tuple inserted into the view and the tuples to be inserted into R_C are indicated by the bold rectangles. Tuples in R_C determine which sets in C is considered to be in a (minimal) cover of S. The colors represent the two elements in C, $c_1 = \{a, b\}$ and $c_4 = \{b, c, d\}$, selected by the insertion of (1) and (4) in R_C. It can be seen that the intermediate relation Q' contains all elements in S, which implies that $\{c_1, c_4\}$ form a cover of S. The insertion of these two elements in R_C is forced by the insertion of $(1, 2, 3, 4)$ in the view. As explained below, $k+1$ copies of R_S are needed to prevent an insertion in those base relations (as updates to one relation will cause an update in all $k + 1$).

More formally, we next show that this is indeed a reduction. First, assume that C' is a minimal cover of S. Then we construct source tuples ΔD such that (j) is inserted into $R_C(I_C)$ iff $c_j \in C'$. Obviously, $\Delta D \in Q(D \oplus \Delta D)$ since C' is a cover, and moreover, ΔD is minimal since C' is minimal. Note that, however, ΔD is not side-effect free: $Q(D \oplus \Delta D)$ contains all permutations of $(1, \ldots, n)$. But side effects are not the concern of the source side-effect problem.

Conversely, suppose that there is a minimal ΔD such that $\Delta D \in Q(D \oplus \Delta D)$. Note that ΔD consists of insertions to $R_C(I_C)$ only. Indeed, if one wants to insert tuples into

$R_S^i(I_S, I_C)$, for some $i \in [1, k+1]$, in order to add a tuple to the view, the same insertions always have to be performed to all $k+1$ source relations $R_S^i(I_S, I_C)$. Obviously, the minimal solution always consists of maximal k updates.

Given the minimal update ΔD to $R_C(I_C)$, we define a set C' such that c_j is in C' iff (j) is in $R_C(I_C)$ in ΔD. Since $(1, \ldots, n)$ is in $Q(D \oplus \Delta D)$, from the definition of Q it follows that ΔD consists of (j)'s such that for any $i \in [1, k]$, there is $(i, j) \in R_S(I_S, I_C)$. Thus C' is a cover of S. In addition, C' is a minimal cover since ΔD is minimal. That is, C' is a minimal cover of S. □

5. Related Work

To our knowledge, the only known complexity results for the analysis of annotation propagation were given in [5, 19]. We have remarked in Section 1 on the connection between our work and [5, 19]. In particular, key preservation, group updates and propagation of view insertions were not considered in [5, 19]. On relational view updates, surprisingly few complexity bounds are known; in fact the only tractability and intractability results we are aware of were established in [2, 8, 15], for finding a minimal view complement for relational views, a problem very different from ours.

There has also been work on the modelling and managing of provenance information [10, 20, 4, 3]. Except for [10], no complexity results were given. In [10], a key-preserving condition was also considered. It was shown there that the condition simplifies the computation of lineage. However, views of [10] are defined in terms of generic mapping functions, which are quite different from SPJ views studied in this paper. As a consequence their complexity results do not apply to the decision problems considered in this paper and vice versa.

An algorithm was provided in [9] for translating view deletions to base relations with zero side-effects, based on data lineage. This algorithm performs an exhaustive search over all candidate solutions, leading to an exponential time complexity. In contrast, with our key-preservation condition, the computation of data lineage can be simplified and the view side-effect free deletion problem is PTIME resolvable.

There has been a host of work on relational view updates (*e.g.,* [8, 11, 14, 15, 3]). Algorithms were provided in [11] for translating restricted view updates to base-table updates without side effects in the presence of certain functional dependencies. An algorithm was developed in [14] to translate (with side effects) a class of SPJ view updates to base relations, with the following restrictions: base tables may only be joined on keys and must satisfy foreign keys; a join view corresponds to a single tree where each node refers to a relation; join attributes must be preserved; and comparisons between two attributes are not allowed in selection conditions. As remarked in Section 2, our key preservation condition is less restrictive than those in [11, 14]. More recently in [3], a bi-directional query language was proposed, which imposes conditions on the operators in the language such that arbitrary changes to views can be carried out. The conditions are more restrictive than the key preservation condition studied in this paper.

Commercial database systems [13, 16, 18] allow updates on very restricted views, while allowing users to specify updates manually with the INSTEAD OF triggers. For example, for views to be deletable IBM DB2 [13] restricts the FROM clause to reference only one base table.

6. Conclusion

We have re-investigated the propagation analysis of annotations under the key preservation condition. We have shown that for key-preserving SPJ views, the view and source side-effect problems are in PTIME as opposed to NP-hard in the absence of the condition [5, 19]. We have also investigated the impact of *group* updates on the complexity of the propagation analysis, and shown that group updates complicate the analysis: for group deletions the view and source side-effect problems become NP-hard for key-preserving SPJ views. In addition, we have established the first complexity results for the analysis of *view insertions* for SPJ views, both in the presence and in the absence of the key preservation condition. These provide a complete picture of the complexity of the propagation analysis of annotations. These results are not only important for data provenance but are also useful for view-update processing.

We are currently studying approximation (heuristic) algorithms for conducting the propagation analysis of annotations when the associated problems are intractable. We also plan to identify other practical conditions on view definitions such that the analysis can be performed efficiently.

Acknowledgment. Wenfei Fan is supported in part by EPSRC GR/S63205/01, GR/T27433/01 and BBSRC BB/D006473/1. Floris Geerts is a postdoctoral researcher of the FWO Vlaanderen and is supported in part by EPSRC GR/S63205/01.

7. References

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

[2] F. Bancilhon and N. Spyratos. Update semantics of relational views *TODS*, 6(4):557–575, 1981.

[3] A. Bohannon, B. Pierce, and J. A. Vaughan. Relational lenses: A language for updateable views. In *PODS*, 2006.

[4] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD*, 2006.

[5] P. Buneman, S. Khanna, and W. Tan. On propagation of deletions and annotations through views. In *PODS*, 2002.

[6] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[7] B. Choi, G. Cong, W. Fan, and S. Viglas. Updating recursive XML views of relations. Submitted for publication, 2006.

[8] S. S. Cosmadakis and C. H. Papadimitriou. Updates of relational views. In *PODS*, 1983.

[9] Y. Cui and J. Widom. Run-time translation of view tuple deletions using data lineage. *Technical Report, Standford University*, 2001.

[10] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *TODS*, 25(2), 2000.

[11] U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *TODS*, 7(3), 1982.

[12] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman and Co., 1979.

[13] IBM. *IBM DB2 Universal Database SQL Reference. http://www.ibm.com/software/data/db2/* .

[14] A. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In *PODS*, 1985.

[15] J. Lechtenborger and G. Vossen. On the computation of relational view complements. *TODS*, 28(2):175–208, 2003.

[16] Oracle. *SQL Reference. http://www.oracle.com/ technology/documentation/database10g.html.*

[17] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4), 2000.

[18] SQL server. *MSDN Library. http://msdn.microsoft.com/library.*

[19] W. C. Tan. Containment of relational queries with annotation propagation. In *DBPL*, 2003.

[20] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.