Run-Time Translation of View Tuple Deletions Using Data Lineage[*]

Yingwei Cui and Jennifer Widom

Computer Science Department, Stanford University
{cyw, widom}@db.stanford.edu

Abstract

To support first-class views in a database system, we must not only allow users to query and browse the database through views, but also allow database updates through views: the well-known *view update* problem. Although the view update problem has been studied extensively, we take a fresh approach based on *data lineage* to provide improved results for translating deletions against virtual or materialized views into deletions against the underlying database. Our fully automatic algorithm finds a translation that is guaranteed to be *exact* (side-effect free) whenever an exact translation exists, using only the view definition at compile-time and the base data at view-update time.

1 Introduction

Data management through user-defined *views* is an important feature for shared databases. To support first-class views, we must not only allow users to query and browse the database through views, but also allow database updates through views. A number of problems arise when updating a database through views, yielding the well-known and well-studied *view update* problem [BS81, Cle78, DB82, KU84, LS91, Mas84, Shu96, Sto75, Tom94]. The typical steps involved in a view update are:

(1) The user requests an update U_V to a view.
(2) Some process, referred to as *view update translation*, takes U_V and produces an update U_D to the underlying *base data*.
(3) U_D is applied to the base database. If the view is *materialized* (as opposed to *virtual*), then the view is modified to reflect the base data update U_D.

In this paper, we consider the specific problem of translating deletions against view tuples (referred to as *view deletions*) into deletions against base tables (referred to as *base deletions*).

Many previous approaches, e.g., [Cle78, Kel86, LS91, RS79], require participation from the view definer and/or the view updater in specifying view update translations. In this paper, we use techniques based on *data lineage* [CWW00] to devise a fully automatic algorithm that translates view deletions using only the view definition at compile-time and the base data at view-update time.

Referring back to the steps outlined above for view update, when the base data update U_D is applied in step (3) the view is modified, either logically or physically, as a result. Let U'_V denote the view modifications induced by database update U_D. U'_V should certainly contain the user's requested view update U_V from

[*]This work was supported by the National Science Foundation under grant IIS-9811947.

step (1), in which case U_D is called a *correct* view update translation. Even better, U'_V should be identical to U_V, in which case U_D is called an *exact* view update translation. Surprisingly, most previous work on the view update problem does not consider exactness, e.g., [Mas84, LS91], or guarantees exactness only for a restricted class of select-project-join (SPJ) views, e.g., [DB82, Kel85]. Unlike these algorithms, our run-time translation algorithm finds a translation for any view deletion against any SPJ view that is guaranteed to be exact whenever an exact translation exists, and data lineage helps us achieve this result.

There may be more than one translation for a view update—even more than one exact translation—leading to an inherent *ambiguity* in the view update translation process that has been the focus of much previous work, e.g., [BS81, Kel86, LS91]. We do not explore the ambiguity issue in this paper. Instead, our goal is to find, as fast as possible and without view definer or user intervention, an exact translation that updates a small number of base tuples. Also we do not consider constraints on the base tables in our work: we assume that all deletions on base tables are valid. Some discussion on how base table constraints can affect the view update translation process can be found in [DB82, Kel85]. Finally, we do not consider insertions or modifications to views. This paper focuses specifically on how data lineage can be used to improve the view update translation process. After some exploratory work it is our belief that the current state of the art in data lineage, e.g., [CWW00], is applicable only to view deletions, and by extension to a portion of the view modification problem (complemented by a solution to the insertion "half" of the problem). For a complete view update translation package, our deletion algorithm could be combined with any previous algorithm for translating view insertions, e.g., [DB82], although as noted earlier these algorithms do not guarantee exactness.

In summary, this paper focuses on the automation and exactness aspects of the view update translation process. We present a fully automatic run-time algorithm for translating deletions against SPJ views into deletions against the underlying database. Our algorithm is based on data lineage techniques, and it yields translations that are guaranteed to be exact whenever an exact translation exists.

1.1 Paper Outline

The remainder of the paper proceeds as follows. In the rest of this section we survey related work and then present a running example to illustrate the view update problem and our solutions. In Section 2 we formalize the view update problem for deletions, and in Section 3 we provide necessary background on data lineage. Section 4 formalizes the relationship between data lineage and view deletions, which is used as the basis for our translation algorithm for single-tuple deletions presented in Section 5. Sections 6 and 7 extend our algorithm to handle sets of deletions and deletions specified by a selection condition. Empirical results from our implementation are presented in Section 8, and Section 9 concludes the paper.

1.2 Related Work

One of the biggest differences between our approach to the view update problem and most previous approaches is that we determine view update translations at view-update time instead of at view-definition time.

UserGroup

user	group
john	sale
lisa	mkt
lisa	eng1
lisa	eng6
joe	eng1
joe	eng2
mary	eng2
mary	eng4
ted	eng4
ted	eng5

GroupAccess

group	file
sale	f1
mkt	f2
eng1	f3
eng1	f4
eng2	f3
eng2	f4
eng4	f5
eng5	f5
eng5	f6
eng6	f3

Figure 1: Base tables

V

user	file
lisa	f3
lisa	f4
joe	f3
joe	f4
mary	f3
mary	f4
mary	f5
ted	f5
ted	f6

Figure 2: View contents

Consequently, other approaches may miss exact view update translations when they exist, because those approaches cannot take all data-dependent information into account.

Some previous approaches ask the view definer to specify, as part of a view definition, the set of permitted view update operations together with their translations, e.g., [Cle78, RS79]. A somewhat more automated approach is for the system to enumerate different view update translations at view definition time and let the view definer choose the preferable ones, e.g., [Kel86]. Another approach, proposed in [LS91], translates view updates using user-provided information at view-definition time as well as at view-update time. All of these approaches require input from the view definer or view updater, unlike our approach which can be fully automatic, allowing exact view update translations without any additional effort at view definition or update time.

A fully automatic approach is suggested in [Mas84], which provides ten general rules for translating view deletions and insertions on select, project, and join views. The concept of *view complements* is introduced in [BS81] to supply extra semantic information for selecting a view update translation. Neither of these approaches guarantees translation exactness.

An algorithm proposed in [DB82] guarantees exact translations in restricted cases by requiring certain functional dependencies on the base data. A concept of *source-tuple* is defined in [DB82], similar in spirit but with a different definition than data lineage. [Kel85] also proposed an algorithm that guarantees exact translations for a very restricted classes of SPJ views: to be updatable a view must include all key and join attributes of the base tables, and the base tables must join on the key attributes and satisfy foreign key constraints. Our approach makes no restrictions on the SPJ views we consider.

1.3 Running Example

We will use the following example throughout the paper to illustrate the view update problem and our solutions. Consider a simple user access control database with two base tables: $\texttt{UserGroup}(\texttt{user}, \texttt{group})$ and $\texttt{GroupAccess}(\texttt{group}, \texttt{file})$. The `UserGroup` table contains the groups that each user belongs to, and the `GroupAccess` table lists the files accessible by each group. Figure 1 shows a small example database.

Example #	View deletions	Translation
1	delete $\langle \texttt{ted}, \texttt{f5} \rangle$ from V	delete $\langle \texttt{ted}, \texttt{eng4} \rangle$ from $\texttt{UserGroup}$ delete $\langle \texttt{eng5}, \texttt{f5} \rangle$ from $\texttt{GroupAccess}$
2	delete $\langle \texttt{lisa}, \texttt{f3} \rangle$ from V	delete $\langle \texttt{lisa}, \texttt{eng6} \rangle$ from $\texttt{UserGroup}$ delete $\langle \texttt{eng1}, \texttt{f3} \rangle$ from $\texttt{GroupAccess}$
3	delete $\langle \texttt{joe}, \texttt{f3} \rangle$ from V	delete $\langle \texttt{joe}, \texttt{eng2} \rangle$ from $\texttt{UserGroup}$ delete $\langle \texttt{eng1}, \texttt{f3} \rangle$ from $\texttt{GroupAccess}$
4	delete $\langle \texttt{joe}, \texttt{f4} \rangle$ from V	delete $\langle \texttt{joe}, \texttt{eng1} \rangle$, $\langle \texttt{joe}, \texttt{eng2} \rangle$ from $\texttt{UserGroup}$
5	delete $\langle \texttt{lisa}, \texttt{f4} \rangle$, $\langle \texttt{joe}, \texttt{f4} \rangle$ from V	delete $\langle \texttt{eng1}, \texttt{f4} \rangle$ from $\texttt{GroupAccess}$
6	delete from V where $\texttt{user} = '\texttt{ted}'$	delete $\langle \texttt{ted}, \texttt{eng4} \rangle$, $\langle \texttt{ted}, \texttt{eng5} \rangle$ from $\texttt{UserGroup}$ delete $\langle \texttt{eng5}, \texttt{f5} \rangle$, $\langle \texttt{eng5}, \texttt{f6} \rangle$ from $\texttt{GroupAccess}$

Table 1: Example view deletions and translations

We consider an SPJ view V defined in relational algebra as follows:

$$V = \pi_{\texttt{user},\texttt{file}}(\sigma_{\texttt{group}='\texttt{eng}\%'}(\texttt{UserGroup} \bowtie \texttt{GroupAccess}))$$

V contains information about all files accessible by users in the "eng" (engineering) groups; its contents over our sample base data are shown in Figure 2. We consider six example view deletions, listed in Table 1. All translations shown in the table are exact except for Example 4, which has no exact translation. The remainder of this paper will show how we can use data lineage techniques to find the translations shown in this example, and more generally to translate any deletions against any SPJ view.

2 The View Update Problem for Deletions

We now formally define the view update translation problem for deletions. In this paper we consider only select-project-join (SPJ) views, so every view V we consider can be written as

$$V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$$

where $R_1, \ldots, R_n$ are base tables, C is a boolean condition (containing both join and local selection predicates), and A is a list of projected attributes. We represent deletions on a base table R as ∇R and deletions on a database $D = R_1, \ldots, R_n$ as $\nabla D = \nabla R_1, \ldots, \nabla R_n$. We assume set semantics (no duplicates) and no base table constraints throughout the paper. Base table constraints are discussed in Section 1 and duplicates in Section 9.

Given a view V, we use $-t$ to denote a request to delete a single view tuple $t \in V$. (In Section 6 we generalize to view update requests that are sets of deleted tuples, as in Example 5 of Table 1, and in Section 7 we handle view update requests specified by a selection condition, as in Example 6 of Table 1.) Note that we use "V" generically throughout the paper to represent the name of a view, its definition, and sometimes its (virtual or actual) contents. We now formalize the concept of a *view deletion translation*.

Definition 2.1 (View Deletion Translation) Given a view V and a deletion request $-t$ where $t \in V$, we say that database deletion ∇D is a *translation for* $-t$ if ∇D causes the deletion of t from V when applied to database D. More formally, let V' be the new view based on the updated database $D - \nabla D$, and let $\nabla V = V - V'$ be the actual deleted view tuples, which we call the *view deletions induced by* ∇D. We say that ∇D is a translation for $-t$ if $\{t\} \subseteq \nabla V$. □

Let ∇D be a translation for deletion $-t$ on view V and let ∇V be the view deletions induced by ∇D as specified in Definition 2.1. If $\nabla V = \{t\}$ then ∇D is an *exact* translation for $-t$. Otherwise, ∇D is *inexact* and causes *side-effect* (or *extra deletions*) $E = \nabla V - \{t\}$. Concrete examples will be seen in Section 4.

For SPJ views, when we perform a deletion ∇D on the base data, the changes to the view induced by ∇D are always deletions, never insertions or modifications. However, when we translate a deletion request $-t$ into updates on the base data, it is not necessary to produce only deletions. For example, in translating $-t$ we might choose to delete some existing base tuples, then insert some new ones to compensate for view side-effects caused by the deletions. In this paper, we do not consider the compensation approach, focusing only on the pure deletions-to-deletions translation problem. In considering the more general problem we found that cases where view deletions benefit from translations encompassing all types of base data updates are very rare, although to be fair such cases do exist. If these cases turn out to be more prevalent than we think, we will consider the more general translation algorithm as future work.

3 Data Lineage

The concept of *data lineage* was introduced primarily to enable users to trace derived data items back to their sources, especially in a transformational or data warehousing setting [CWW00, CW01]. Data lineage is closely related to the view update problem, specifically to the translation of view deletions. As we will see, using data lineage as a basis for translating view deletions introduces a fresh approach with improved results.

We briefly review the definition of data lineage and introduce a new concept of *exclusive lineage*. The actual computation of lineage and exclusive lineage will be presented in Section 5. In this paper we only present the aspects of data lineage that are relevant to the view update problem addressed. For further motivation, details, and full treatment of relational data lineage see [CWW00, CW00].

Given a view V over base database $D = R_1, \ldots, R_n$ and a tuple $t \in V$, informally t's *lineage in* D is $\langle R_1^*, \ldots, R_n^* \rangle$, where $R_i^* \subseteq R_i$ ($i = 1..n$) contains exactly those tuples in R_i that are used to derive t. We call R_i^* the ith *branch* of t's lineage. Although we have studied data lineage for general relational views including aggregation and set operators in [CWW00], in this paper we are focusing on SPJ views. The following formal definition of data lineage is equivalent to the lineage definition in [CWW00] when specialized to SPJ views.

Definition 3.1 Consider an SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$ and a view tuple $t \in V$. A tuple $t_i \in R_i$ belongs to t's lineage branch R_i^* if and only if:

$$\{t\} \subseteq \pi_A(\sigma_C(R_1 \times \cdots \times R_{i-1} \times \{t_i\} \times R_{i+1} \times \cdots \times R_n))$$

We denote t's entire lineage in D as $lineage(t, V, D) = \langle R_1^*, \ldots, R_n^* \rangle$. □

Example 3.2 Recall view V from Section 1.3, containing user file access control information. Given tuple $t = \langle \texttt{ted}, \texttt{f5} \rangle$ in V, based on Definition 3.1 t's lineage is:

$$\begin{aligned} lineage(t, V, D) = \langle &\texttt{UserGroup}^* = \{\langle \texttt{ted}, \texttt{eng4} \rangle, \langle \texttt{ted}, \texttt{eng5} \rangle\}, \\ &\texttt{GroupAccess}^* = \{\langle \texttt{eng4}, \texttt{f5} \rangle, \langle \texttt{eng5}, \texttt{f5} \rangle\}\rangle \end{aligned}$$ □

The *exclusive lineage* of a view tuple t is the set of base tuples that contribute to t and only to t. Formally:

Definition 3.3 Consider an SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$ and a view tuple $t \in V$. A tuple $t_i \in R_i$ belongs to t's exclusive lineage branch R_i^{**} if and only if:

$$\{t\} = \pi_A(\sigma_C(R_1 \times \cdots \times R_{i-1} \times \{t_i\} \times R_{i+1} \times \cdots \times R_n))$$

(The difference from Definition 3.1 is simply = instead of $\subseteq$.) We denote t's entire exclusive lineage in D as $elineage(t, V, D) = \langle R_1^{**}, \ldots, R_n^{**} \rangle$. Note that $R_i^{**} \subseteq R_i^*$, $i = 1..n$. □

Example 3.4 Again, consider view V from Section 1.3 and tuple $t = \langle \texttt{ted}, \texttt{f5} \rangle$ in V. Based on Definition 3.3, t's exclusive lineage is:

$$elineage(t, V, D) = \langle \texttt{UserGroup}^{**} = \{\langle \texttt{ted}, \texttt{eng4} \rangle\},\ \texttt{GroupAccess}^{**} = \{\langle \texttt{eng5}, \texttt{f5} \rangle\}\rangle$$

Tuples $\langle \texttt{ted}, \texttt{eng5} \rangle$ and $\langle \texttt{eng4}, \texttt{f5} \rangle$ from t's lineage (Example 3.2) are not in t's exclusive lineage, because $\langle \texttt{ted}, \texttt{eng5} \rangle$ also is in the lineage of view tuple $\langle \texttt{ted}, \texttt{f6} \rangle$, and $\langle \texttt{eng4}, \texttt{f5} \rangle$ also is in the lineage of view tuple $\langle \texttt{bob}, \texttt{f5} \rangle$. □

4 Relationship Between Data Lineage and View Deletions

As mentioned earlier, data lineage is closely related to the view update problem, specifically to translating view deletions. In this section, we formalize the relationship between data lineage and view deletions, to lay the groundwork for our deletion translation algorithm to be specified in Section 5. We continue to focus on single-tuple deletions. Recall that we extend our results to deletions of a view tuple set and deletions specified by a selection condition in Sections 6 and 7, respectively.

Consider an SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$ over database D and the deletion $-t$ of a tuple $t \in V$. Ideally, deleting t's exclusive lineage (Definition 3.3) from D induces the deletion of t from V, as in the following example.

Example 4.1 Consider Example 1 from Table 1, which deletes tuple $t = \langle \texttt{ted}, \texttt{f5} \rangle$ from V. Example 3.4 showed that t's exclusive lineage is:

$$elineage(t, V, D) = \langle \texttt{UserGroup}^{**} = \{\langle \texttt{ted}, \texttt{eng4} \rangle\},\ \texttt{GroupAccess}^{**} = \{\langle \texttt{eng5}, \texttt{f5} \rangle\}\rangle$$

If we delete these tuples from the base tables, t will be deleted from V. Therefore, $\nabla elineage(t, V, D)$ is a translation for $-t$. □

An important property of a translation that deletes the exclusive lineage of the deleted view tuple, as in Example 4.1, is that the translation is guaranteed to be exact. By Definition 3.3 of exclusive lineage, none of the tuples in $R_1^{**}, \ldots, R_n^{**}$ contribute to any view tuples other than t, so translation $\nabla D^{**} = \nabla R_1^{**}, \ldots, \nabla R_n^{**}$ can induce only the deletion of t. Unfortunately, ∇D^{**} is not always a translation, even if $-t$ has an exact translation, as shown by the following example.

Example 4.2 Consider Example 2 from Table 1, which deletes tuple $t = \langle \texttt{lisa}, \texttt{f3} \rangle$ from V. Based on Definition 3.3, t's exclusive lineage is:

$$elineage(t, V, D) = \langle \texttt{UserGroup}^{**} = \{\langle \texttt{lisa}, \texttt{eng6} \rangle\},\ \texttt{GroupAccess}^{**} = \varnothing \rangle$$

Deleting $elineage(t, V, D)$ does not induce the deletion of t. Thus, $\nabla elineage(t, V, D)$ is not a translation for $-t$. However, $-t$ does have an exact translation that deletes tuple $\langle \texttt{lisa}, \texttt{eng6} \rangle$ from `UserGroup` and $\langle \texttt{eng1}, \texttt{f3} \rangle$ from `GroupAccess`. □

When the exclusive lineage does not provide us with a translation, we can instead consider deleting one branch R_i^* of t's lineage. (We also could consider deleting all branches, but each branch individually always is sufficient to induce the deletion of t.) Unlike ∇D^{**}, any ∇R_i^* is a translation, but it may not be exact even when there is an exact translation, as shown by the following example.

Example 4.3 Consider again Example 1 from Table 1, which deletes tuple $t = \langle \texttt{ted}, \texttt{f5} \rangle$ from view V. Example 3.2 showed that t's lineage is:

$$\begin{aligned} lineage(t, V, D) = \langle & \texttt{UserGroup}^* = \{\langle \texttt{ted}, \texttt{eng4} \rangle, \langle \texttt{ted}, \texttt{eng5} \rangle\}, \\ & \texttt{GroupAccess}^* = \{\langle \texttt{eng4}, \texttt{f5} \rangle, \langle \texttt{eng5}, \texttt{f5} \rangle\} \rangle \end{aligned}$$

Deleting either lineage branch is a translation for $-t$, but neither translation is exact: $\nabla\texttt{UserGroup}^*$ also induces the deletion of view tuple $\langle \texttt{ted}, \texttt{f6} \rangle$, while $\nabla\texttt{GroupAccess}^*$ also induces the deletion of view tuple $\langle \texttt{mary}, \texttt{f5} \rangle$. Example 4.1 illustrated an exact translation for $-t$. □

So far we have seen that ∇D^{**} may provide an exact translation or may not be a translation at all, and ∇R_i^* (for any $i \in 1..n$) provides a translation but it may not be exact. In neither case are we guaranteed to be provided with an exact translation even if one exists, and nor is it the case that when ∇D^{**} fails ∇R_i^* succeeds, or vice-versa. Thus, if neither ∇D^{**} nor ∇R_i^* $(i = 1..n)$ provides an exact translation, the brute-force approach is to enumerate all possible subsets of all base relations, checking if deleting those subsets is an exact translation. Fortunately we can reduce the search space considerably using the following theorem, which tells us that if there is an exact translation for $-t$, then there is an exact translation that is contained in t's lineage. We will further reduce the search space with pruning techniques in our algorithm presented in Section 5.

Theorem 4.4 Consider an SPJ view V over database D and a tuple $t \in V$. If deletion $-t$ has an exact translation, then $-t$ has an exact translation ∇D such that $\nabla D \subseteq \mathit{lineage}(t, V, D)$.[1]

Proof: See Appendix A.1. □

5 View Tuple Deletion Algorithm

Based on our observations in Section 4, we specify an algorithm DELETE(t, V, D) in Figure 3 that translates deletion $-t$ on view V into deletions ∇D on base database D. The algorithm performs the deletions ∇D on the base database. We assume that if V is materialized (or if there are other materialized views over D), then a view maintenance algorithm will detect the deletions and modify any materialized views accordingly. Our algorithm can be modified easily to return ∇D and/or ∇V (the deletions on V induced by ∇D; note $\nabla V = \{t\}$ in the case of an exact translation) if desired. DELETE finds an exact translation for $-t$ whenever one exists. In cases when $-t$ has no exact translation, DELETE uses an inexact translation, with some attempt to minimize the side-effect. The algorithm proceeds as follows.

In line 1 of Figure 3, we compute t's lineage using the *tracing query* from [CWW00]. In the tracing query, $A = t$ is shorthand for equating all attributes in list A to their values in tuple t, and operator $\mathit{Split}_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(R) = \langle \pi_{\mathbf{R}_1}(R), \ldots, \pi_{\mathbf{R}_n}(R) \rangle$ projects table R on the schema of R_i for $i = 1..n$. We then compute t's exclusive lineage D^{**} based on Definition 3.3 (lines 2–5), and delete it from D (line 6). Computing the exclusive lineage requires one n-way join for each tuple in t's lineage, but lineage is typically very small, and each join is expected to be cheap since one branch of the join is simply the lineage tuple $\{t_i\}$. If deleting t's exclusive lineage induces the deletion of t from V (line 7), then we are done, as in Example 4.1. We know that t has not been deleted if it is still derivable from its remaining lineage: $t \in \pi_A(\sigma_C((R_1^* - R_1^{**}) \times \cdots \times (R_n^* - R_n^{**})))$.

If deleting the exclusive lineage did not successfully delete t, then we try the next step discussed in Section 4, which is to consider deleting the remainder of some lineage branch R_i^*. Before exploring this option, we recompute t's lineage in our smaller D, i.e., in the D that remains after deleting D^{**} (line 8). Any tuple t_i that is eliminated from R_i^* in the recomputed lineage no longer contributes to t: the tuples it formerly joined with to produce t must have been deleted as part of the exclusive lineage.

In lines 9–14 we consider each remaining lineage branch R_i^* in turn. If ∇R_i^* is an exact translation, then we are done. During the iteration, we keep track of the lineage branch with smallest side-effect, in case we eventually fail to find an exact translation at all. To determine the side-effect of ∇R_i^*, we first compute all potential extra deletions E using the query $\pi_A(\sigma_C(R_1 \times \cdots \times R_i^* \times \cdots \times R_n)) - \{t\}$ (line 11). We then remove from E tuples that are still derivable after deleting R_i^* (line 12). The remaining E is the actual side-effect induced by ∇R_i^*. In terms of efficiency, line 11 performs a join where one branch (R_i^*) is expected to be very small. Line 12 appears to be expensive—nearly a recomputation of the entire view—however a good

[1]Note that we abuse the subset symbol "$\subseteq$" to operate on lists of tables: $\nabla D \subseteq \mathit{lineage}(t, V, D)$ is shorthand for $\nabla R_1 \subseteq R_1^*, \ldots, \nabla R_n \subseteq R_n^*$.

```
**procedure** DELETE($t, V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n)), D$)
**// Compute $t$'s lineage:**
(1) $D^* = \langle R_1^*, \ldots, R_n^* \rangle \leftarrow Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(\sigma_{C \wedge A=t}(R_1 \times \cdots \times R_n))$;

**// Compute and delete $t$'s exclusive lineage:**
(2) $D^{**} = \langle R_1^{**}, \ldots, R_n^{**} \rangle \leftarrow \langle \varnothing, \ldots, \varnothing \rangle$;
(3) **for** $i = 1..n$ **do**
(4)     **for each** $t_i \in R_i^*$ **do**
(5)         **if** $\pi_A(\sigma_C(R_1 \times \cdots \times \{t_i\} \times \cdots \times R_n)) = \{t\}$ **then** $R_i^{**} \leftarrow R_i^{**} \cup \{t_i\}$;
(6) $D \leftarrow D - D^{**}$;

**// Check if $t$ is deleted:**
(7) **if** $t \notin \pi_A(\sigma_C((R_1^* - R_1^{**}) \times \cdots \times (R_n^* - R_n^{**})))$ **then return**;

**// Recompute $t$'s lineage in smaller $D$:**
(8) $D^* = \langle R_1^*, \ldots, R_n^* \rangle \leftarrow Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(\sigma_{C \wedge A=t}(R_1 \times \cdots \times R_n))$;

**// Find a lineage branch with zero or smallest side-effect:**
(9) $minE \leftarrow \infty$;
(10) **for** $i = 1..n$ **do**
(11)     $E \leftarrow \pi_A(\sigma_C(R_1 \times \cdots \times R_i^* \times \cdots \times R_n)) - \{t\}$;
(12)     $E \leftarrow E - \pi_A(\sigma_C(R_1 \times \cdots \times (R_i - R_i^*) \times \cdots \times R_n))$
(13)     **if** $E = \varnothing$ **then** $R_i \leftarrow R_i - R_i^*$; **return**;
(14)     **if** $|E| < minE$ **then** $m \leftarrow i$; $minE \leftarrow |E|$;

**// Enumerate subsets of $t$'s lineage with limited size:**
(15) $k \leftarrow |\sigma_C(R_1^* \times \cdots \times R_n^*)|$;
(16) $S \leftarrow$ all subsets of $D^*$ that contain at most $k$ tuples;
(17) **for** each $D' = \langle R_1', \ldots, R_n' \rangle \in S$ **do**
(18)     **if** $t \in \pi_A(\sigma_C((R_1^* - R_1') \times \cdots \times (R_n^* - R_n')))$
(19)     **then** prune all subsets of $D'$ from $S$;
(20)     **else** $E \leftarrow \bigcup_{i=1..n} \pi_A(\sigma_C(R_1 \times \cdots \times R_i' \times \cdots \times R_n)) - \{t\}$;
(21)         $E \leftarrow E - \pi_A(\sigma_C((R_1 - R_1') \times \cdots \times (R_n - R_n')))$;
(22)         **if** $E = \varnothing$ **then** $D \leftarrow D - D'$; **return**;
(23)         **else** prune all supersets of $D'$ from $S$;

**// Delete the lineage branch $R_m^*$ with smallest side-effect:**
(24) $R_m \leftarrow R_m - R_m^*$; **return**;
```

Figure 3: Translating the deletion of a view tuple t

optimizer can effectively introduce a semijoin with E into the right-hand operand of the difference to keep the join small. If we find a translation with no side-effect, i.e., $E = \varnothing$, then we are done (line 13). Otherwise, line 14 determines if we are seeing the smallest side-effect so far and, if so, records the side-effect size and current branch in $minE$ and m, respectively.

Example 5.1 Consider Example 2 from Table 1, which deletes tuple $t = \langle \texttt{lisa}, \texttt{f3} \rangle$ from V. In procedure DELETE, we compute t's lineage and exclusive lineage, and obtain:

$$\begin{aligned} lineage(t, V, D) = \langle & \texttt{UserGroup}^* = \{\langle \texttt{lisa}, \texttt{eng1} \rangle, \langle \texttt{lisa}, \texttt{eng6} \rangle\}, \\ & \texttt{GroupAccess}^* = \{\langle \texttt{eng1}, \texttt{f3} \rangle, \langle \texttt{eng6}, \texttt{f3} \rangle\}\rangle \\ elineage(t, V, D) = \langle & \texttt{UserGroup}^{**} = \{\langle \texttt{lisa}, \texttt{eng6} \rangle\},\ \texttt{GroupAccess}^{**} = \varnothing \rangle \end{aligned}$$

We first delete t's exclusive lineage from the base database D (lines 1–6), which does not induce the deletion of t (line 7). We then recompute t's lineage in the updated D (line 8) and obtain:

$$lineage(t, V, D) = \langle \texttt{UserGroup}^* = \{\langle \texttt{lisa}, \texttt{eng1} \rangle\},\ \texttt{GroupAccess}^* = \{\langle \texttt{eng1}, \texttt{f3} \rangle\}\rangle$$

Deleting the lineage branch $\texttt{UserGroup}^*$ has the side-effect of deleting $\langle \texttt{lisa}, \texttt{f4} \rangle$ as well as $\langle \texttt{lisa}, \texttt{f3} \rangle$. However, deleting the lineage branch $\texttt{GroupAccess}^*$ has no side-effect, so we further delete $\texttt{GroupAccess}^*$. The entire exact translation is to delete tuple $\langle \texttt{lisa}, \texttt{eng6} \rangle$ from UserGroup and $\langle \texttt{eng1}, \texttt{f3} \rangle$ from GroupAccess. $\square$

If t does not have a lineage branch whose deletion is side-effect free, we next look for a combination of tuples from different lineage branches whose deletion forms an exact translation for $-t$ (lines 15–23). We look for an exact translation by enumerating subsets of t's lineage, based on Theorem 4.4. Letting $x = |R_1^*| + |R_2^*| + \ \ldots\ + |R_n^*|$, we must enumerate and check up to 2^x possible translations. Although x tends to be relatively small, in some cases we can further bound the size of the search space based on the number of derivations of tuple t, as shown by the following theorem.

Theorem 5.2 Consider an SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$ over database D and a tuple $t \in V$. Let $lineage(t, V, D) = \langle R_1^*, \ldots, R_n^* \rangle$, and let $k = |\sigma_C(R_1^* \times \cdots \times R_n^*)|$. k is the number of different ways of deriving tuple t. If $-t$ has an exact translation, then it has an exact translation $\nabla D' = \nabla R_1', \ldots, \nabla R_n'$ such that $R_i' \subseteq R_i^*$ for $i = 1..n$, and $|R_1'| + \cdots + |R_n'| \leq k$.

Proof: A formal proof is given in Appendix A.2. Informally, to delete t we need only delete one tuple from each of t's k derivations. If we have an exact translation for t, it must delete at least one tuple from each of t's derivations. We can eliminate from the exact translation all but one tuple from each derivation, and we still have an exact translation.[2] $\square$

Lines 15–17 compute the bound k from Theorem 5.2 and initiate the subset enumeration. For each candidate translation $\nabla D'$, line 18 checks if $\nabla D'$ is indeed a translation, by checking if t is still derivable after deleting D'. If so, then $\nabla D'$ is not a translation, but it does give us information that we can use to further prune the search space: If deleting D' does not delete t, then nor can deleting any subset of D' (line 19). (This

[2]This proof might lead us to consider an alternate approach: Instead of considering all lineage subsets up to size k, we compute the derivations of t and then consider all combinations of one tuple from each derivation. It turns out that this enumeration can actually be more expensive than the one we are using, because without expensive bookkeeping it might end up considering the same subset multiple times.

pruning technique seems to suggest we should enumerate larger subsets first, but in the other branch of the **if** statement we introduce a pruning technique that eliminates supersets.)

Lines 20–22 check if the candidate translation $\nabla D'$ is an exact translation, by checking if it has any side-effect. The procedure is similar but not identical to lines 11–13. We first compute all potential extra deletions E by joining each R_i' to the remaining relations (line 20). We then remove from E tuples that are still derivable after deleting D' (line 21). The remaining E is the actual side-effect induced by $\nabla D'$.[3] If E is empty, then we have found an exact translation, so we perform the deletion and are done (line 22). Otherwise, line 23 performs additional pruning: If deleting D' introduces a side-effect, then so will deleting any superset of D'.

Example 5.3 Consider Example 3 from Table 1, which deletes tuple $t = \langle \texttt{joe}, \texttt{f3} \rangle$ from V. In DELETE, we compute:

$$\begin{aligned} lineage(t, V, D) &= \langle \texttt{UserGroup}^* = \{\langle \texttt{joe}, \texttt{eng1}\rangle, \langle \texttt{joe}, \texttt{eng2}\rangle\}, \\ &\qquad \texttt{GroupAccess}^* = \{\langle \texttt{eng1}, \texttt{f3}\rangle, \langle \texttt{eng2}, \texttt{f3}\rangle\}\rangle \\ elineage(t, V, D) &= \langle \texttt{UserGroup}^{**} = \varnothing,\ \texttt{GroupAccess}^{**} = \varnothing\rangle \end{aligned}$$

We first delete t's exclusive lineage, which does not induce the deletion of t, and then we recompute t's lineage. (Obviously we can add an **if** statement in the algorithm to skip these deletion and recomputation steps when the exclusive lineage is empty, as in this example.) Deleting either remaining lineage branch of t will cause a side-effect, so we proceed to enumerate subsets of t's lineage. The number of t's derivations is $k = |\sigma_{\texttt{group}='\texttt{eng}\%'}(\texttt{UserGroup}^* \bowtie \texttt{GroupAccess}^*)| = 2$, so we only need to consider lineage subsets containing one or two tuples. Enumerating these subsets, we find an exact translation $\nabla D'$ for $-t$ where $D' = \langle \texttt{UserGroup}' = \{\langle \texttt{joe}, \texttt{eng2}\rangle\}, \texttt{GroupAccess}' = \{\langle \texttt{eng1}, \texttt{f3}\rangle\}\rangle$. □

Let us consider the complexity of lines 15–23. The number of iterations in the **for** loop is bounded by $|D^*|^k$, where k is number of t's derivations. In general, both $|D^*|$ and k are small, and we use pruning techniques to further reduce the number of iterations. Within each iteration, the complexity of line 18 is similar to that of line 7, the complexity of line 20 is n times the complexity of line 11 (recall our view is an n-way join), and the complexity of line 21 is similar to line 12. Empirical results are reported in Section 8.

As the final step of the algorithm (line 24), if we cannot find a lineage subset D' whose deletion causes exactly the deletion of t, then by Theorem 4.4 $-t$ has no exact translation. In this case, we delete the lineage branch R_m^* with the smallest side-effect of all lineage branches, which we saved in line 14.

Example 5.4 Consider Example 4 from Table 1, which deletes tuple $t = \langle \texttt{joe}, \texttt{f4} \rangle$ from V. In DELETE, we compute:

$$\begin{aligned} lineage(t, V, D) &= \langle \texttt{UserGroup}^* = \{\langle \texttt{joe}, \texttt{eng1}\rangle, \langle \texttt{joe}, \texttt{eng2}\rangle\}, \\ &\qquad \texttt{GroupAccess}^* = \{\langle \texttt{eng1}, \texttt{f4}\rangle, \langle \texttt{eng2}, \texttt{f4}\rangle\}\rangle \end{aligned}$$

[3]Note that this computation is similar to *incremental view maintenance*, which determines the effect on a view of base table modifications [BDD$^+$98, GM95].

$elineage(t, V, D) = \langle \texttt{UserGroup}^{**} = \varnothing,\ \texttt{GroupAccess}^{**} = \varnothing \rangle$

Through lines 1–23 in the DELETE algorithm, we discover that $-t$ has no exact translation: The number of t's derivations is $k = 2$, and deleting any subset of t's lineage with no more than two tuples either does not cause the deletion of t or causes deletions in addition to t. Thus, we delete $\texttt{UserGroup}^*$, which has the smaller side-effect of the two lineage branches. □

To our knowledge, all previous view update algorithms either do not guarantee exact translations for deletions even when they exist, e.g., [Mas84, LS91], or they guarantee exactness for a restricted class of views satisfying certain functional dependencies or foreign key constraints, e.g., [DB82, Kel86]. Algorithm DELETE is the first to guarantee exact deletion translations whenever they exist for general SPJ views.

6 Deleting Sets of View Tuples

So far we have considered translating the deletion of a single tuple t from view V. In general, users may want to request the deletion of a set of tuples $T \subseteq V$, specified either by value or by a selection condition. In this section we extend our algorithm to translate a view deletion request $-T$, while in the next section we consider condition-based deletions.

We might be tempted to translate view deletion $-T$ ($T \subseteq V$) by translating deletion $-t$ for each $t \in T$ using algorithm DELETE from Section 5. In addition to being inefficient, this approach is too conservative: it is possible that there is an exact translation for $-T$ even when there is no exact translation for some tuples $t \in T$. This case occurs when extra deletions from an inexact translation for $-t$, $t \in T$, are not actually a side-effect, because the extra deletions are contained in T. Thus, we must translate $-T$ as a unit.

First let us generalize Definition 2.1 of a view deletion translation:

Definition 6.1 (View Deletion Translation for $-T$) Consider a view V, a deletion request $-T$ where $T \subseteq V$, and a database deletion ∇D. Let V' be the new view based on the updated database $D - \nabla D$, and let $\nabla V = V - V'$ be the actual deleted view tuples. We say that ∇D is a *translation* for $-T$ if $T \subseteq \nabla V$. If $T = \nabla V$ then ∇D is an *exact* translation, otherwise ∇D causes *side-effect* $E = \nabla V - T$. □

We also must extend our definitions for lineage and exclusive lineage to apply to sets of tuples. The lineage of a view tuple set $T \subseteq V$ combines the lineage of each tuple $t \in T$:

$$lineage(T, V, D) = \bigcup_{t \in T} lineage(t, V, D)$$

We can compute T's lineage according to this definition with one query [CWW00]: $Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(\sigma_C(R_1 \times \cdots \times R_n) \ltimes T)$. (Recall that the operator $Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(R) = \langle \pi_{\mathbf{R}_1}(R), \ldots, \pi_{\mathbf{R}_n}(R) \rangle$ projects table R on the schema of R_i for $i = 1..n$.) A tuple $t_i \in R_i$ belongs to T's exclusive lineage R_i^{**} if and only if:

$$\pi_A(\sigma_C(R_1 \times \cdots \times R_{i-1} \times \{t_i\} \times R_{i+1} \times \cdots \times R_n)) \subseteq T$$

Based on the above definitions, the observations we made on the relationship between data lineage and single-tuple deletions in Section 4 all carry over directly to the deletion of a view tuple set. Furthermore, Theorem 5.2 in Section 5 can be extended to cover $-T$ as follows.

Theorem 6.2 Consider an SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$ over database D and a tuple set $T \subseteq V$. Let $lineage(T, V, D) = \langle R_1^*, \ldots, R_n^* \rangle$. If $-T$ has an exact translation, then it has an exact translation that deletes at most k tuples in T's lineage, where $k = |\sigma_C(R_1^* \times \cdots \times R_n^*) \ltimes T|$ is the total number of derivations of all tuples $t \in T$.

Proof: A formal proof is given in Appendix A.3. The proof is very similar to the proof of Theorem 5.2, with the only notable difference being the computation of k. Here, we reduce the size of k after joining all lineage branches by performing a semijoin with T, because joining lineage tuples that result from two different view tuples t_1 and t_2 in T may result in view tuples t_3 outside of T. We do not want to accidentally count derivations of t_3 in our bound k. □

Thus, we need only small modifications to our DELETE algorithm from Section 5 to translate the deletion of a view subset T. Figure 4 shows the modified algorithm DELETE-SET with all changes underlined. DELETE-SET finds a translation for $-T$ that is guaranteed to be exact whenever an exact translation exists.

Example 6.3 Consider Example 5 from Table 1, which deletes a tuple set $T = \{\langle \texttt{joe}, \texttt{f4} \rangle, \langle \texttt{lisa}, \texttt{f4} \rangle\}$ from V. In DELETE-SET, we compute T's lineage and exclusive lineage as:

$$\begin{aligned} lineage(T, V, D) = \langle & \texttt{UserGroup}^* = \{\langle \texttt{joe}, \texttt{eng1} \rangle, \langle \texttt{lisa}, \texttt{eng1} \rangle\}, \\ & \texttt{GroupAccess}^* = \{\langle \texttt{eng1}, \texttt{f4} \rangle\}\rangle \\ elineage(T, V, D) = \langle & \texttt{UserGroup}^{**} = \varnothing,\ \texttt{GroupAccess}^{**} = \{\langle \texttt{eng1}, \texttt{eng4} \rangle\}\rangle \end{aligned}$$

Deleting T's exclusive lineage induces the deletion of T. Thus, we find an exact translation $\nabla elineage(T, V, D)$ for $-T$.

Now consider what would have happened had we translated the deletion of the two tuples $\langle \texttt{joe}, \texttt{f4} \rangle$ and $\langle \texttt{lisa}, \texttt{f4} \rangle$ separately, using our original algorithm DELETE. Suppose we translate deletion $t = \langle \texttt{joe}, \texttt{f4} \rangle$ first. In Example 5.4 we showed that deleting $\langle \texttt{joe}, \texttt{f4} \rangle$ has no exact translation, so the algorithm chooses to delete lineage branch $\texttt{UserGroup}^* = \{\langle \texttt{joe}, \texttt{eng1} \rangle, \langle \texttt{joe}, \texttt{eng2} \rangle\}$. The side-effect $E = \{\langle \texttt{joe}, \texttt{f3} \rangle\}$ of this translation is not contained in the original view request $T = \{\langle \texttt{joe}, \texttt{f4} \rangle, \langle \texttt{lisa}, \texttt{f4} \rangle\}$, so regardless of the translation chosen for tuple deletion $t = \langle \texttt{lisa}, \texttt{f4} \rangle$, the final translation for $-T$ will be inexact. □

7 Deleting View Tuples Specified By Selection Conditions

Consider as usual our SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$. Suppose that instead of specifying an actual view tuple or set of tuples to be deleted, the user performs a condition-based deletion as in SQL: "`DELETE FROM` V `WHERE` C'" where C' can be any selection condition, similar to the conditions C used in view definitions. A naive approach is to first compute the tuple set $T = \sigma_{C'}(V)$, then translate $-T$ using

```
**procedure** DELETE-SET($\underline{\underline{T}}$, $V = \pi_A(\sigma_C(R1 \times \cdots \times R_n)), D$)
**// Compute $\underline{\underline{T}}$'s lineage:**
(1) $D^* = \langle R_1^*, \ldots, R_n^* \rangle \leftarrow Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(\sigma_C(R_1 \times \cdots \times R_n) \underline{\underline{\ltimes T}})$;

**// Compute and delete $\underline{\underline{T}}$'s exclusive lineage:**
(2) $D^{**} = \langle R_1^{**}, \ldots, R_n^{**} \rangle \leftarrow \langle \varnothing, \ldots, \varnothing \rangle$;
(3) **for** $i = 1..n$ **do**
(4)     **for** each $t_i \in R_i^*$ **do**
(5)         **if** $\pi_A(\sigma_C(R_1 \times \cdots \times \{t_i\} \times \cdots \times R_n)) \underline{\underline{\subseteq T}}$ **then** $R_i^{**} \leftarrow R_i^{**} \cup \{t_i\}$;
(6) $D \leftarrow D - D^{**}$;

**// Check if $\underline{\underline{T}}$ is deleted:**
(7) **if** $\underline{\underline{T \cap}}\, \pi_A(\sigma_C((R_1^* - R_1^{**}) \times \cdots \times (R_n^* - R_n^{**}))) \underline{\underline{= \varnothing}}$ **then return**;

**// Recompute $\underline{\underline{T}}$'s lineage in smaller $D$:**
(8) $D^* = \langle R_1^*, \ldots, R_n^* \rangle \leftarrow Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(\sigma_C(R_1 \times \cdots \times R_n) \underline{\underline{\ltimes T}})$;

**// Find a lineage branch with zero or smallest side-effect:**
(9) $minE \leftarrow \infty$;
(10) **for** $i = 1..n$ **do**
(11)     $E \leftarrow \pi_A(\sigma_C(R_1 \times \cdots \times R_i^* \times \cdots \times R_n)) \underline{\underline{- T}}$;
(12)     $E \leftarrow E - \pi_A(\sigma_C(R_1 \times \cdots \times (R_i - R_i^*) \times \cdots \times R_n))$
(13)     **if** $E = \varnothing$ **then** $R_i \leftarrow R_i - R_i^*$; **return**;
(14)     **if** $|E| < minE$ **then** $m \leftarrow i$; $minE \leftarrow |E|$;

**// Enumerate subsets of $\underline{\underline{T}}$'s lineage with limited size:**
(15) $k \leftarrow |\sigma_C(R_1^* \times \cdots \times R_n^*) \underline{\underline{\ltimes T}}|$;
(16) $S \leftarrow$ all subsets of $D^*$ that contain at most $k$ tuples;
(17) **for** each $D' = \langle R_1', \ldots, R_n' \rangle \in S$ **do**
(18)     **if** $\underline{\underline{T \cap}}\, \pi_A(\sigma_C((R_1^* - R_1') \times \cdots \times (R_n^* - R_n'))) \underline{\underline{\neq \varnothing}}$
(19)     **then** prune all subsets of $D'$ from $S$;
(20)     **else** $E \leftarrow \bigcup_{i=1..n} \pi_A(\sigma_C(R_1 \times \cdots \times R_i' \times \cdots \times R_n)) \underline{\underline{- T}}$;
(21)         $E \leftarrow E - \pi_A(\sigma_C((R_1 - R_1') \times \cdots \times (R_n - R_n')))$;
(22)         **if** $E = \varnothing$ **then** $D \leftarrow D - D'$; **return**;
(23)         **else** prune all supersets of $D'$ from $S$;

**// Delete the lineage branch $R_m^*$ with smallest side-effect:**
(24) $R_m \leftarrow R_m - R_m^*$; **return**;
```

Figure 4: Translating the deletion of a view subset T

algorithm DELETE-SET from Section 6. However, the computation of $T = \sigma_{C'}(V)$ can be expensive, particularly if V is a virtual view. Fortunately, instead of computing T, we can incorporate the selection condition C' into our translation algorithm. The new algorithm DELETE-COND is specified in Figure 5, with the changes from DELETE-SET underlined.

In lines 1 and 8, to compute the lineage of $T = \sigma_{C'}(V)$ according to V, we use query $Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(\sigma_{C \wedge C'}(R_1 \times \cdots \times R_n))$, instead of performing a semijoin with T as in DELETE-SET. In line 5, we test whether a tuple $t_i \in R_i^*$ belongs to the exclusive lineage of $T = \sigma_{C'}(V)$ by testing whether $\pi_A(\sigma_{C \wedge \neg C'}(R_1 \times \cdots \times \{t_i\} \times \cdots \times R_n)) = \varnothing$, because this condition is satisfied if and only if t_i does

```
**procedure** DELETE-COND($\underline{\underline{C'}}, V = \pi_A(\sigma_C(R1 \times \cdots \times R_n)), D$)
**// Compute $\underline{\underline{T = \sigma_{C'}(V)}}$'s lineage:**
(1) $D^* = \langle R_1^* \ldots R_n^* \rangle \leftarrow Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(\sigma_{C\underline{\underline{\wedge C'}}}(R_1 \times \cdots \times R_n))$;

**// Compute and delete $T$'s exclusive lineage:**
(2) $D^{**} = \langle R_1^{**}, \ldots, R_n^{**} \rangle \leftarrow \langle \varnothing, \ldots, \varnothing \rangle$;
(3) **for** $i = 1..n$ **do**
(4)       **for** each $t_i \in R_i^*$ **do**
(5)             **if** $\pi_A(\sigma_{C\underline{\underline{\wedge \neg C'}}}(R_1 \times \cdots \times \{t_i\} \times \cdots \times R_n)) = \varnothing$ **then** $R_i^{**} \leftarrow R_i^{**} \cup \{t_i\}$;
(6) $D \leftarrow D - D^{**}$;

**// Check if $T$ is deleted:**
(7) **if** $\pi_A(\sigma_{C\underline{\underline{\wedge C'}}}((R_1^* - R_1^{**}) \times \cdots \times (R_n^* - R_n^{**}))) = \varnothing$ **then return**;

**// Recompute $T$'s lineage in smaller $D$:**
(8) $D^* = \langle R_1^* \ldots R_n^* \rangle \leftarrow Split_{\mathbf{R}_1,\ldots,\mathbf{R}_n}(\sigma_{C\underline{\underline{\wedge C'}}}(R_1 \times \cdots \times R_n))$;

**// Find a lineage branch with zero or smallest side-effect:**
(9) $minE \leftarrow \infty$;
(10) **for** $i = 1..n$ **do**
(11)       $E \leftarrow \pi_A(\sigma_{C\underline{\underline{\wedge \neg C'}}}(R_1 \times \cdots \times R_i^* \times \cdots \times R_n))$;
(12)       $E \leftarrow E - \pi_A(\sigma_C(R_1 \times \cdots \times (R_i - R_i^*) \times \cdots \times R_n))$
(13)       **if** $E = \varnothing$ **then** $R_i \leftarrow R_i - R_i^*$; **return**;
(14)       **if** $|E| < minE$ **then** $m \leftarrow i$; $minE \leftarrow |E|$;

**// Enumerate subsets of $T$'s lineage with limited size:**
(15) $k \leftarrow |\sigma_{C\underline{\underline{\wedge C'}}}(R_1^* \times \cdots \times R_n^*)|$;
(16) $S \leftarrow$ all subsets of $D^*$ that contain at most $k$ tuples;
(17) **for** each $D' = \langle R_1', \ldots, R_n' \rangle \in S$ **do**
(18)       **if** $\pi_A(\sigma_{C\underline{\underline{\wedge C'}}}((R_1^* - R_1') \times \cdots \times (R_n^* - R_n'))) \neq \varnothing$
(19)       **then** prune all subsets of $D'$ from $S$;
(20)       **else** $E \leftarrow \bigcup_{i=1..n} \pi_A(\sigma_{C\underline{\underline{\wedge \neg C'}}}(R_1 \times \cdots \times R_i' \times \cdots \times R_n))$;
(21)             $E \leftarrow E - \pi_A(\sigma_C((R_1 - R_1') \times \cdots \times (R_n - R_n')))$;
(22)             **if** $E = \varnothing$ **then** $D \leftarrow D - D'$; **return**;
(23)             **else** prune all supersets of $D'$ from $S$;

**// Delete the lineage branch $R_m^*$ with smallest side-effect:**
(24) $R_m \leftarrow R_m - R_m^*$; **return**;
```

Figure 5: Translating view deletions based on a selection condition C'

not contribute to any view tuple that fails condition C'. We can similarly compute the potential side-effect of a base deletion ∇R_i^* using $\pi_A(\sigma_{C\wedge\neg C'}(R_1 \times \cdots \times R_i^* \times \cdots \times R_n))$ in lines 11 and 20. Finally, in lines 7 and 18, we test whether deleting base tuples $D' = \langle R_1', \ldots, R_n' \rangle$ is a translation by testing whether $\pi_A(\sigma_{C\wedge C'}((R_1^* - R_1') \times \cdots \times (R_n^* - R_n'))) = \varnothing$.

Example 7.1 Consider Example 6 from Table 1, which deletes all tuples with $\texttt{user} = \texttt{'ted'}$ from view V. Let $\mathbf{U}$ and $\mathbf{G}$ denote the schemas of base tables `UserGroup` and `GroupAccess`, respectively. In DELETE-COND, we compute the lineage of $T = \sigma_{\texttt{user='ted'}}(V)$ as follows:

`UserGroup`*

user	group
ted	eng4
ted	eng5

`GroupAccess`*

group	file
eng4	f5
eng5	f5
eng5	f6

`UserGroup`**

user	group
ted	eng4
ted	eng5

`GroupAccess`**

group	file
eng5	f5
eng5	f6

Figure 6: $T = \sigma_{\texttt{user}='\texttt{ted}'}(V)$, its lineage and exclusive lineage

$$\langle \texttt{UserGroup}^*, \texttt{GroupAccess}^* \rangle \;=\; Split_{\mathbf{U},\mathbf{G}}(\sigma_{\texttt{group}='\texttt{eng}\%' \wedge \texttt{user}='\texttt{ted}'}(\texttt{UserGroup} \bowtie \texttt{GroupAccess}))$$

The result is shown in Figure 6. We then compute T's exclusive lineage and obtain $D^{**} = \langle \texttt{UserGroup}^{**}$, $\texttt{GroupAccess}^{**} \rangle$ in Figure 6. ∇D^{**} is a translation, and therefore is an exact translation for the condition-based view deletion. □

8 Empirical Results

We have implemented our view deletion translation algorithm using stored procedures in a standard commercial relational DBMS. In this section, we present some preliminary performance results that explore the following two questions:

1. Roughly how often is there an exact translation for a view deletion $-t$ that deletes the exclusive lineage of t? That deletes a lineage branch of t? That deletes a subset of t's lineage? How often is there no exact translation at all?

2. How does our algorithm perform as we scale up the size of the base data?

We use synthetic data for the experiments based on the user access control database in our example from Section 1.3. Recall that the two base tables are $\texttt{UserGroup}(\texttt{user}, \texttt{group})$ and $\texttt{GroupAccess}(\texttt{group}, \texttt{file})$. The view we consider is $V = \pi_{\texttt{user},\texttt{file}}(\texttt{UserGroup} \bowtie \texttt{GroupAccess})$. (We omit the selection condition from our original example view.) Contents of the base tables are generated using the parameters listed in Table 2. There are *#groups* groups. The users in each group and the files that each group has access to are picked randomly from a fixed set of users and files with sizes *#users* and *#files*, respectively. The base data size depends on *#groups*, as well as parameters *users-per-group* (*UG* for short) and *files-per-group* (*FG* for short). The actual number of users for a given group is selected from a Gaussian distribution centered at *UG* with a standard deviation of $UG/2$, and similarly for the number of files each group has access to. For our experiments we always set $UG = FG$, and we refer to this parameter as *density*. When this number is low, the join is very selective, while when it is high, the join is heavily intertwined.

Our experiments focus on the single-tuple deletion case, i.e., each request is to delete a single view tuple. All experiments were conducted on a dedicated Windows NT machine with a Pentium II processor.

Parameter	*Description*
#groups	total number of groups
#users	maximum number of users
#files	maximum number of files
users-per-group (UG)	average number of users that belong to each group
files-per-group (FG)	average number of files that each group has access to

Table 2: Data generation parameters

8.1 Four Cases of Translations

Given a view deletion request $-t$, there are four possible outcomes for the translation according to algorithm DELETE:

1. $-t$ has an exact translation ∇D^{**} that deletes t's exclusive lineage D^{**}. In this case, our algorithm finds the exact translation ∇D^{**} and returns at line 7 of Figure 3.

2. Case (1) fails, but $-t$ has an exact translation ∇R_i^* that deletes a lineage branch R_i^* of $-t$. In this case, our algorithm finds an exact translation ∇R_i^* and returns at line 13. The number of iterations of lines 10–14 will vary from deletion to deletion.

3. Cases (1) and (2) both fail, but $-t$ has an exact translation. In this case, our algorithm finds an exact translation ∇D by deleting a subset of t's lineage. It returns at line 22, and the number of iterations of lines 17–23 will vary from deletion to deletion.

4. $-t$ has no exact translations, so the algorithm returns at line 24.

In our experiment, we study how often each of the above four cases occurs under different base data scenarios. Specifically, when generating the base data, we fix *#groups*, *#users*, and *#files* at 50, and we increase the density (*UG* and *FG*) from 1 to 30. For each base data scenario, we issue a deletion request $-t$ for each tuple $t \in V$, and count the number of translations that belong to each of the four cases. Figure 7 presents the percentage of each case among all four cases.

The result shows that for very sparse base data, case (1) happens frequently: most deletions $-t$ have an exact translation that deletes t's exclusive lineage. At the extreme, when the density is 1, each group g has exactly one user u and one file f, so all lineage tuples of any view tuple $t = \langle u, f \rangle$ are exclusive. Therefore deleting the entire exclusive lineage is always an exact translation. As the density increases, case (4) occurs more often. The intuition is that as more tuples are joined with each other, deleting any lineage tuple of t is likely to affect other view tuples, so $-t$ often does not have an exact translation. However, as the density further increases, cases (2) and (3) start to occur more frequently. Now, because each view tuple often has several derivations, although deleting a branch or subset of t's lineage may affect some derivations of another view tuple t', t' often can be derived in another way, so $-t$ still has an exact translation. Finally, when the base data is very dense, case (2) completely takes over.

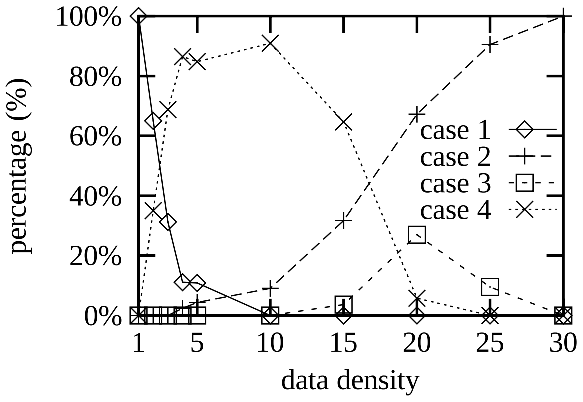

Figure 7: Four cases of translations

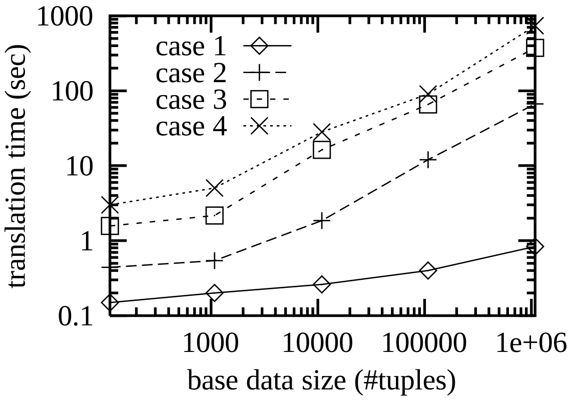

Figure 8: Translation time

8.2 Algorithm Scalability

In this experiment, we study the scalability of our algorithm to the size of the base data. Specifically, we fix the density at 8 and increase *#groups* from 10 to 100,000. (We also set *#users* = *#files* = *#groups*, but these values do not affect the base data size.) We continue to consider the four translation cases from Section 8.1 separately, and we delete view tuples one at a time until we have several representative instances of each case. Figure 8 shows the average time for the translation in each case (in seconds) as the total size of the base data increases. From the results plotted on log-scale x and y axes, we see that the translation time for each case grows at worst approximately linearly in the size of the base data.

9 Conclusions and Future Work

We developed a fully automatic algorithm for translating deletions against any select-project-join view into deletions against the underlying database. Deletions can be specified as a single tuple, a set of tuples, or as a selection condition over the view. Our algorithm can be parameterized by the view definition at compile-time, and by the view update request (and base data) at run-time. By taking data-dependent information into account at run-time and using techniques based on data lineage, our algorithm finds translations that are guaranteed to be free of side-effects (*exact*) whenever such translations exist. No previous algorithms we know of can make the same claim. Empirical results show that our algorithm often finds exact translations very quickly, and that it scales at worst linearly in the database size.

There are numerous avenues for extending our work:

- This paper considers select-project-join (SPJ) views only. More general relational views (including, e.g., aggregation, set union, difference) pose additional challenges to the view update problem. Our work on data lineage does cover a very general class of relational views [CWW00], and we hope to use it as a basis for more general view update translation algorithms. Going one step further, we also addressed the data lineage problem for "views" defined by general *transformation graphs* [CW01], which appears to pose an even more challenging view update problem.

- We considered set semantics in this paper. In [CWW00] we developed a theory and algorithms for data lineage in the presence of duplicates (multisets), in both the base tables and the views. Our approach to duplicates in [CWW00] can, we believe, lead to an effective treatment of duplicates in our view deletion translation algorithm.

- We have addressed the deletion-to-deletion translation problem only. Even extending to consider base insertions or modifications when translating view deletions (the "compensation" idea discussed in Section 1) is a significant step, particularly if we wish to retain our exactness guarantee. We also might extend our algorithm to provide translations for view insertions or modifications, although as discussed in Section 1 it appears that data lineage techniques may not be applicable in these cases.

- Our algorithm does not attempt to produce any kind of *minimal* translation, e.g., one that deletes the fewest base tuples. Using cardinality as a minimality metric, modifying our algorithm is not difficult, but keeping it efficient is a challenge. We also do not take constraints on base data (or on base data updates) into account—again the challenge may be one primarily of efficiency. Finally, our current approach considers each view in isolation. When there are multiple views, an update translation could induce side-effects on views other than the view being updated, and it might be interesting to take such "side-side-effects" into account.

Acknowledgements

We are particularly grateful to Yujie Cao for her thorough implementation of our initial view update algorithms. We also thank Jeff Ullman for several useful discussions.

References

[BDD$^+$98] R. Bello, K. Dias, A. Downing, J. Feena, W. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in Oracle. In *Proc. of the Twenty-Fourth International Conference on Very Large Databases*, pages 659–664, New York City, New York, August 1998.

[BS81] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transaction on Database Systems*, 6(4):557–575, 1981.

[Cle78] E.K. Clemons. *An external schema facility to support data base updates*. Academic Press, 1978.

[CW00] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proc. of the Sixteenth International Conference on Data Engineering*, pages 367–378, San Diego, California, February 2000.

[CW01] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *Proc. of the Twenty-Seventh International Conference on Very Large Data Bases*, Rome, Italy, September 2001. To appear.

[CWW00] Y. Cui, J. Widom, and J.L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems*, 25(2):179–227, 2000.

[DB82] U. Dayal and P.A. Bernstein. On the correct translation of update operations on relational views. *ACM Transaction on Database Systems*, 8(3):381–416, 1982.

[GM95] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized View and Data Warehousing*, 18(2):3–19, June 1995.

[Kel85] A.M. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In *Proc. of the Fourth International Conference on Principle of Database Systems*, pages 467–476, Stockholm, Sweden, March 1985.

[Kel86] A.M. Keller. Choosing a view update translator by dialog at view definition time. In *Proc. of the Twelfth International Conference on Very Large Data Bases*, pages 467–476, Kyoto, Japan, August 1986.

[KU84] A.M. Keller and J.D. Ullman. On complementary and independent mappings of databases. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 143–148, Boston, Massachusetts, June 1984.

[LS91] J.A. Larson and A.P. Sheth. Updating relational views using knowledge at view definition and view update time. *Information Systems*, 16(2):145–168, 1991.

[Mas84] Y. Masunaga. A relational database view update translation mechanism. In *Proc. of the Tenth International Conference on Very Large Data Bases*, pages 309–320, Singapore, August 1984.

[RS79] L. Rowe and K.A. Schoens. Data abstractions, views, and updates in Rigel. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 71–81, Boston, Massachusetts, May 1979.

[Shu96] H. Shu. Formulating view update translation as constraint satisfaction. Technical report, University of Karlstad, Sweden, November 1996. http://www-b.informatik.uni-hannover.de/∼hs/.

[Sto75] M. Stonebraker. Implementation of integrity constraints and views by query modification. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 65–78, San Jose, California, May 1975.

[Tom94] A. Tomasic. Determining correct view update translations via query containment. In *Proc. of the Workshop on Deductive Databases and Logic Programming*, pages 75–83, Santa Margherita Ligure, Italy, June 1994.

A Proofs

A.1 Proof for Theorem 4.4

Theorem 4.4 Consider an SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$ over database D and a tuple $t \in V$. If deletion $-t$ has an exact translation, then $-t$ has an exact translation ∇D such that $\nabla D \subseteq lineage(t, V, D)$. $\square$

Proof: The proof relies on the following two facts, which are obvious enough that we omit detailed justification:

- Let $\langle R_1^*, \ldots, R_n^* \rangle = lineage(t, V, D)$. $R_i^* \neq \varnothing$ for $i = 1..n$.
- There exist tuples $t_1 \in R_1, \ldots, t_n \in R_n$ such that $\pi_A(\sigma_C(\{t_1\} \times \cdots \times \{t_n\})) = \{t\}$.

Suppose $-t$ has an exact translation $\nabla D = \nabla R_1, \ldots, \nabla R_n$ where $\nabla R_i \subseteq R_i$, $i = 1..n$. We prove that $\nabla D' = \nabla R'_1, \ldots, \nabla R'_n$ where $\nabla R'_i = \nabla R_i \cap R_i^*$, $i = 1..n$, also is an exact translation. First, we prove that $\nabla D'$ is a translation, i.e., $t \notin V'$ where V' is the new view based on the updated database $D - \nabla D'$. Suppose for the sake of a contradiction that $t \in V'$. Then, we know there exist $t_i \in R_i - \nabla R'_i$ for $i = 1..n$ such that $\pi_A(\sigma_C(\{t_1\} \times \cdots \times \{t_n\})) = \{t\}$. Based on Definition 3.1 of lineage, $t_i \in R_i^*$, $i = 1..n$. Since $t_i \in R_i - \nabla R'_i$, we know that $t_i \notin \nabla R'_i = \nabla R_i \cap R_i^*$. Therefore since $t_i \in R_i^*$, $t_i \notin \nabla R_i$, and thus $t_i \in R_i - \nabla R_i$. Therefore $t \in \pi_A(\sigma_C((R_1 - \nabla R_1) \times \cdots \times (R_n - \nabla R_n)))$, which contradicts the fact that ∇D is a translation for $-t$. Thus, $t \notin V'$ and $\nabla D'$ is a translation for $-t$. Because ∇D is an exact translation and $\nabla D' \subseteq \nabla D$, by the definition of an exact translation $\nabla D'$ also is exact.

A.2 Proof for Theorem 5.2

Theorem 5.2 Consider an SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$ over database D and a tuple $t \in V$. Let $lineage(t, V, D) = \langle R_1^*, \ldots, R_n^* \rangle$, and let $k = |\sigma_C(R_1^* \times \cdots \times R_n^*)|$. If $-t$ has an exact translation, then it has an exact translation $\nabla D' = \nabla R'_1, \ldots, \nabla R'_n$ such that $R'_i \subseteq R_i^*$ for $i = 1..n$, and $|R'_1| + \cdots + |R'_n| \leq k$. $\square$

Proof: The proof relies on the following fact, which is obvious enough that we omit detailed justification:

- For any $t_1 \in R_1^*, \ldots, t_n \in R_n^*$, $\pi_A(\sigma_C(\{t_1\} \times \cdots \times \{t_n\})) = \{t\}$ or $\pi_A(\sigma_C(\{t_1\} \times \cdots \times \{t_n\})) = \varnothing$. The number of distinct combinations such that $\pi_A(\sigma_C(\{t_1\} \times \cdots \times \{t_n\})) = \{t\}$ is k.

Let the following list enumerate the k distinct combinations of $t_1 \in R_1^*, \ldots, t_n \in R_n^*$ such that $\pi_A(\sigma_C(\{t_1\} \times \cdots \times \{t_n\})) = \{t\}$.

(1) $t_1^1,\ t_2^1, \ldots, t_n^1$
(2) $t_1^2,\ t_2^2, \ldots, t_n^2$
...

(k) $t_1^k,\ t_2^k, \ldots, t_n^k$

We know $-t$ has an exact translation $\nabla D'' = \nabla R_1'', \ldots, \nabla R_n''$, and without loss of generality (by Theorem 4.4), assume $R_i'' \subseteq R_i^*$, $i = 1..n$. We need to prove that there is an exact translation $\nabla D' = \nabla R_1', \ldots, \nabla R_n'$ such that $|R_1'| + \cdots + |R_n'| \leq k$. For each combination $j = 1..k$ in the list above, there exists at least one t_i^j, $i \in 1..n$, such that $t_i^j \in \nabla R_i''$, because otherwise t would remain in the view after deleting D''. Let $D' = \bigcup_{j=1..k}\{t_i^j\}$ which is of size k. $\nabla D'$ is a translation, because it deletes one tuple from each of the k combinations that produce t. Furthermore, it is an exact translation because it is a subset of $\nabla D''$, which is an exact translation.

A.3 Proof for Theorem 6.2

Theorem 6.2 Consider an SPJ view $V = \pi_A(\sigma_C(R_1 \times \cdots \times R_n))$ over database D and a tuple set $T \subseteq V$. Let $lineage(T, V, D) = \langle R_1^*, \ldots, R_n^* \rangle$. If $-T$ has an exact translation, then it has an exact translation that deletes at most k tuples in T's lineage, where $k = |\sigma_C(R_1^* \times \cdots \times R_n^*) \ltimes T|$ is the total number of derivations of all tuples $t \in T$. □

Proof: The proof relies on the following fact, which is obvious enough that we omit detailed justification:

- Consider all combinations $t_1 \in R_1^*, \ldots, t_n \in R_n^*$. The number of distinct combinations such that $\pi_A(\sigma_C(\{t_1\} \times \cdots \times \{t_n\})) = \{t\}$ and $t \in T$ is k.

Let the following list enumerate the k distinct combinations of $t_1 \in R_1^*, \ldots, t_n \in R_n^*$ such that $\pi_A(\sigma_C(\{t_1\} \times \cdots \times \{t_n\})) = \{t\}$ and $t \in T$:

(1) $t_1^1,\ t_2^1, \ldots, t_n^1$
(2) $t_1^2,\ t_2^2, \ldots, t_n^2$
...
(k) $t_1^k,\ t_2^k, \ldots, t_n^k$

We know T has an exact translation $\nabla D'' = \nabla R_1'', \ldots, \nabla R_n''$, and without loss of generality (by the obvious set generalization of Theorem 4.4), assume $R_i'' \subseteq R_i^*$, $i = 1..n$. We need to prove that there is an exact translation $\nabla D' = \nabla R_1', \ldots, \nabla R_n'$ such that $|R_1'| + \cdots + |R_n'| \leq k$. For each combination $j = 1..k$ in the list above, there exists at least one t_i^j, $i \in 1..n$, such that $t_i^j \in \nabla R_i''$, because otherwise some $t \in T$ would remain in the view after deleting D''. Let $D' = \bigcup_{j=1..k}\{t_i^j\}$ which is of size k. $\nabla D'$ is a translation, because it deletes one tuple from each of the k combinations that produce all of the t's in T. Furthermore, it is an exact translation because it is a subset of $\nabla D''$, which is an exact translation.