Algorithms for Translating View Updates to Database Updates for Views Involving Selections, Projections, and Joins

Arthur M. Keller
Stanford University
(Extended Abstract)

ABSTRACT. We consider the problem of updating databases through views composed of selections, projections, and joins of a series of Boyce-Codd Normal Form relations. This involves translating updates expressed against the view to updates expressed against the database. We present five criteria that these translations must satisfy. For each type of view update (insert, delete, replace), we provide a list of templates for translation into database updates that satisfy the five criteria. We show that there cannot be any other translations that satisfy the five criteria.

KEYWORDS. Relational databases, database theory, view update.

CR CATEGORIES. H.2.1, H.1.1, E.4.

1 Introduction

The problem of updating databases through views is an important practical problem that has attracted much interest [Bancilhon 79, 81, Carlson 79, Davidson 81, Dayal 78, 79, 82, Furtado 79, Kaplan 81, Keller 82, 84, Masunaga 83]. The user specifies queries to be executed against the database view; these queries are translated to queries against the underlying database through query modification [Stonebraker 75]. However, in current practice, updates must be specified against the underlying database rather than against the view, because the problem of updating relational databases through views is inherently ambiguous.

This work was supported in part by contract N39-82-G-0250 (the Knowledge Base Management Systems Project, Prof. Gio Wiederhold, Principal Investigator) from the Defense Advanced Research Projects Agency and by contract AFOSR-80-0212 (Universal Relations, Prof. Jeff Ullman, Principal Investigator) from the Air Force Office of Scientific Research, both of the United States Department of Defense. The views and conclusions contained in this document are those of the authors and should not be interpreted as representative of the official policies of DARPA or the US Government.

Author's address: Computer Science Department, Stanford University, Stanford, CA 94305-2085.

Since the view is only an uninstantiated window onto the database, any updates specified against the database view must be translated into updates against the underlying database. The updated database state then induces a new view state, and it is desirable that the new view state correspond to performing the user-specified update directly on the original view state as far as possible. This is described by the following diagram.

$$\begin{tikzcd}
V(DB) \arrow[r, "U"{name=A}] & U(V(DB)) \overset{?}{=} V(DB') \\
DB \arrow[u, "V"] \arrow[r, "T(U)"{name=B}] & T(U)(DB) = DB' \arrow[u, "V"']
\arrow[Rightarrow, from=A, to=B, "T"]
\end{tikzcd}$$

The user specifies update U against the view of the database, $V(DB)$. The view update translator T supplies the database update $T(U)$, which results in DB' when applied to the database. The new view state is $V(DB')$. This translation has *no side effects in the view* if $V(DB') = U(V(DB))$, that is, if the view has changed precisely in accordance with the user's request. There are update translators that do not have side effects in the view for views that involve only selections and projections. There are some updates for views involving joins that cannot be translated without side effects in the view. Therefore, in this paper, only views that involve joins may have update translators with side effects in the view.

Given a view definition, the question of choosing a view update translator arises. This requires understanding the ways in which individual view update requests may be satisfied by database updates. Any particular view update request may result in a view state that does not correspond to any database state. Such a view update request may not be translated without relaxing the constraint precluding view side effects.* Otherwise, the update request is rejected by the view update translator. If we are lucky, there will be precisely one way to perform the database update that results in the desired view update. Since the view is many-to-one, the new view state may correspond to many database states. Of these database states, we would like to choose

* In certain cases, we have shown that it is quite reasonable to relax this constraint in a limited manner [Keller 82].

one that is "as close as possible" under some measure to the original database state. That is, we would like to minimize the effect of the view update on the database.

2 View Update Translation

We need to define a few terms to explain the process of translation of view updates into database updates [Ullman 82, Maier 83]. A domain is a (finite) set. A relation schema is an ordered (or tagged) set of domains and a set of constraints that tuples in the relation must satisfy. A functional dependency or key dependency is an example of such a constraint. A tuple is an ordered (or tagged) set of values, each one from its respective domain. The extension of a relation is the set of tuples in the relation. A database schema is a set of relation schemata indexed by relation name. A database extension is a set of relation extensions, one for each relation in the database schema.

A database view definition is a mapping whose domain is the set of all relation extensions for a given database schema. The range of a database view definition is also a set of relation extensions for a schema specific to the view definition. The mapping from the domain database to each relation in the range of the view is defined by a type of database query. The view extension is the extension of the database which is the range of the view for a particular extension of the database which is the domain of the view.

The operations on databases and views are deletion, insertion, and replacement. A deletion is the removal of a single tuple from a relation. An insertion is the addition of a single tuple into a relation. A replacement is the combination of a deletion and an insertion into the same relation performed as a single atomic action that does not require an intermediate consistent state between the deletion and insertion steps. An update is a deletion, an insertion, or a replacement.

A database update may be directly applied against the database, provided it satisfies the constraints on the database. A view update is merely an update that is described against the view, but it must be translated into a sequence of database updates in order for it to be executed. There may be several candidate sequences of database updates corresponding to one view update. We call these sequences of database updates the *translations* of the view update request.

We say that a translation is *valid* if it performs the view update as requested. For updates through select and project views, we will require that the new view extension be precisely the result of performing the view update on the old view extension, were the view to be an ordinary relation. For updates through views that include joins, it may not be possible to perform the view update without additional changes to the view [Keller 82]. These *view side effects* are as a result of functional dependencies that require that changes in the view tuples requested are consistent with the remainder of the database.

Requiring that a translation be valid is not sufficient for our purposes—it is only a first step. We define 5 additional criteria (in Section 3) we require the translations to satisfy. We use the criteria to obtain only the simplest (or minimal) view update translations.

A view update translator is a mapping from view update requests into translations of these view update requests. A translator takes the user's view update requests and translate them into database update requests, which will then be processed by the database system. Thus, a view update translation facility is a useful adjunct to a database system which has a view definition facility. The lack of a view update translation facility means that users must specifying updates directly against the database rather than through views.

We shall provide several (classes of) view update translators. We do not claim that these are the only view update translators possible. Rather, we show that, for each view update request, these view update translators generate all possible translations of that view update request into sequences of database update requests that satisfy our 5 criteria.

The translators we define are generators of the complete set of view update translators that generate translations that satisfy our 5 criteria. So that the view update translator may choose among these alternatives at view update time, we propose that the database adminstrator (DBA) provide additional semantics during view definition time. These additional semantics permit the view update facility to choose one of these translators. The derivation of these semantics from the "business model" [Keller 85a] is beyond the scope of this paper.

3 Criteria for View Update Translation

A translation of a particular view update request is characterized by three sets consisting of the insertions, deletions, and replacements applied to the underlying database. The insertions and deletions are each described by a set containing the affected tuples. The replacements are described by a set of ordered pairs of old and new tuples. These sets all contain the exact tuples, specifying all attributes. We will consider two translations equivalent if they have the same effect on the database. In practice, the equivalence can result from converting a pair of an insertion and a deletion into a replacement, or from swapping the replacement tuples from a pair of replace operations. Formally, let the set of removed tuples be the union of the set of deleted tuples and the set of replaced tuples; similarly, the set of added tuples is the union of the set of inserted tuples and the set of replacement tuples. Then two translations are equivalent if their respective added and removed sets are equal. Recall that the translations are valid when the implement the request exactly (without view side effects).

All of the candidate update translations are to satisfy the following 5 criteria (in addition to being valid).

1. "No database side effects." The only database tuples affected have keys that match their respective values in the tuples mentioned in the view update request. (This is part of the rectangle rule [Dayal 78, 79, 82]). Note that this requires the key of each relation affected to appear in the view. In particular, this means that if the key of a tuple changes, the old and new keys must appear in the respective positions of the view update request.

2. "Only 'one step' changes." Each database tuple is affected by at most one step of the translation for any single view update request. Specifically, a translation cannot replace an inserted tuple, or delete a replaced tuple, or replace a tuple twice in succession. (This rule implies that there is no ordering imposed on the individual database updates that translate a view update.)

3. "Minimal change: no unnecessary changes." There is no valid translation that implements the request by performing only a proper subset of the database requests. (Note that we are concerned with the set of database operations; a deletion is not simpler than a replacement that replaces the same tuple since the replacement is a single request.)

4. "Minimal change: replacements cannot be simplified." Consider two (alternative) database replacement requests where both specify replacing the same tuple. A database replacement that does not involve changing the key is simpler than one where the key changes. A database replacement that changes a proper subset of the attributes changed by another database replacement is simpler. Comparing the attributes of the replaced tuple with those of the replacement tuple, a replacement request that makes some changes is simpler than one that makes those same changes in addition to others.

5. "Minimal change: no delete-insert pairs." We do not allow candidate translations to include both deletions and insertions on any one relation, as they may be converted into replacements, which we consider simpler. Thus candidate translations may contain either deletions or insertions for any relation, but not both, in addition to replacements. A translation may, however, contain an insertion into one relation and a deletion from another relation.

Let us consider the implications of these five criteria. Criterion 1 requires that any change to the database affect only tuples that are relevant to the view update. This requirement is not as stringent as maintaining a constant complement [Bancilhon 81]. Both requirements are intended to eliminate unintended effects on other users. The constant complement method takes a fixed notion of some other user and prevents any actions through our view from affecting the other user, which is contrary to the purpose of a shared database and precludes some reasonable updates or update translators.

Criterion 2 eliminates two types of anomalies. We do not want a tuple to be replaced by two separate tuples; it would have disappeared after the first of these replacements. We also do not want a tuple to undergo multiple separate changes, as these could be combined into a single change. For example, we do not want a tuple to be replaced only to be deleted in its new form. Rather, the original tuple should be deleted.

Criteria 3, 4, and 5 require that the updates be minimal. This takes three forms: we do not do any unnecessary operations, the operations we do perform are the simplest ones possible, and we never do in two steps what can be done in one. The only operation we can simplify is a replacement operation, and it is simplified by not changing the key, or by changing fewer attributes. We consider a one-step replacement operation simpler than a two-step deletion-insertion pair. This allows us to get at the essence of the changes necessary.

The purpose of our five criteria are to permit all possible changes, but only in their simplest forms. If changing a particular attribute value is sufficient, we want to consider that in preference to changing that attribute in addition to others. It is certainly possible to translate a view update request by performing additional changes to the database, but there are endless possibilities for such elaboration. If we are to achieve our goal of characterizing the possibilities, we must restrict ourselves to the simplest ones, which capture the essence of the changes in the more elaborate translations.

Based on the definitions of added and removed sets above, one translation is at least as simple as another if its added and removed sets are subsets of those of the other translation.

THEOREM. For given view update request, for every valid translation, there is (at least one) translation at least as simple that satisfies the 5 criteria.*

THEOREM. The five criteria are independent.

4 Views Consisting of Selections and Projections

We will first consider views consisting of selections and projections of a single Boyce-Codd Normal Form relation. We will then consider the composition of join views and selection and projection views.

We shall deal with select and project views on a single relation with a single key dependency. Let R be the set of attributes in the relation $\mathbf{R}$, and let K be the set of attributes in the key. We shall assume that the functional dependency $K \rightarrow R$ is the *only* consistency constraint on $\mathbf{R}$. Observe that since the relation may only have a key constraint, the database has already undergone normalization.

Let us first consider the selection condition. The selection condition is a conjunction of terms of the form $A \in s$ (or equivalently, $A \notin e$), where s (and e) is a set of constants in the domain of A. (Note that $s \cup e$ is equal to the domain of A.) We call the values in s *selecting values*, and the values in e *excluding values*. For non-selecting attributes the set of selecting values is the entire domain and the set of excluding values is the empty set. We call the attributes appearing in the selection condition *selecting attributes*. If the selection condition is "true" (i.e., an empty conjunction), the set of selecting attributes is empty, the sets of selecting values are the entire domains, and the sets of excluding values are empty. This type of selection condition allows attributes to be treated independently in view updates.

We call the attributes appearing in the view the *projected attributes*, while those not appearing in the view are *projected out*. We require that all the attributes of the key must appear in the view (none may be projected out). Any or all of the selecting attributes may be projected out, except for those in the key. This means that the key of the database is the key of the view.

4-1 Example

Let us consider the relation EMP which contains each employee's number, name, location, and whether the employee is a member of the company baseball team. The company has two locations: New York and San Francisco. Baseball team members must be employees.

* The proofs of the theorems in this paper are contained in the dissertation [Keller 85b].

The personnel manager, Susan, in New York has the following view definition:

View P:
```
Select *
From EMP
Where Location="New York"
```

She requests the deletion of employee #17 from her view. A reasonable translation of this request is to delete the employee record from the underlying database. Thus, we have translated a view deletion into a database deletion. If the employee was a member of the baseball team, he has been removed from that also.

The baseball team manager, Frank, has the following view definition:

View B:
```
Select *
From EMP
Where Baseball="Yes"
```

He requests deletion of employee #14 from his view. It is unreasonable to delete the employee tuple from the underlying database (unless you believe that baseball is all-important). A reasonable translation of this view deletion request is to replace the Baseball attribute of the underlying database tuple with a "No." Thus, we have translated a view deletion into a database replacement.

One might argue that the Frank's view deletion request should have been a replacement. However, this would mean requesting the replacement of a tuple in the view with a view tuple that did not appear in the view. Then Frank's request would not be valid in the view, as the replacement tuple could not possibly be a view tuple. In addition, Frank would have to make a distinction between deletion and replacement that he could not discern by looking at the effects through his view.

It is possible to translate the Susan's request by moving employee #17 to California. We doubt that the California manager would be pleased by such an implementation. Rather, such a request should be issued by someone authorized to access the entire relation (as a replacement request): someone who can see the effects of that request.

We see that a view deletion request is sometimes translated into a database deletion request best and at other times into a database replacement request. As we shall see, similar alternatives arise for insertion and replacement. We suggest that additional semantics be used to choose among the various alternatives, but such semantics are beyond the scope of this paper.

4-2 Translation of Update Requests

We will consider the general case of translating single tuple update requests on a select and project view into updates to the underlying database. We will deal with single tuple insertions first. We will follow that with single tuple deletions. Finally, we will deal with single tuple replacements.

4-3 Translation of Insertion Requests

The request is to insert a single, fully-specified view tuple. The new view tuple must be a valid view tuple—it must satisfy the selection condition—and not conflict with any existing view tuple. That is, there must not already be a tuple in the view with the key of the view tuple to be inserted. The extend-insert algorithms are subroutines of Algorithms classes I-1 and I-2, which are the algorithms that translate view insertions.

ALGORITHM CLASS EXTEND-INSERT: The new database tuple is formed by taking the attributes from the new view tuple as supplied. For remaining attributes, the values are chosen arbitrarily from their respective sets of selecting values (which is the domain for non-selecting attributes). Each combination of values represents a different algorithm from this class. There is a unique extend-insert algorithm iff each attribute projected out (appearing in the database but not in the view) has set of selecting values that is a singleton (has only one element).

ALGORITHM CLASS I-1: If the new view tuple does not conflict with (have the same key as) any existing *database* tuple, then insert the tuple obtained by one of the extend-insert algorithms, else reject the update request. There is an algorithm in class I-1 for each extend-insert algorithm.

ALGORITHM CLASS I-2: If the new view tuple has a key matching that of an existing database tuple, then change the attributes in the database tuple to match the new view tuple and change all attributes in the database tuple with excluding values to arbitrary selecting values. There is an algorithm in class I-2 for every combination of one selecting value from zero or more selecting attributes other than the key.

THEOREM. The set of update translations that satisfy the 5 criteria for candidate update translations for individual view insertions are precisely those in algorithm classes I-1 and I-2.

Let us consider when these translations may be used. There may be several translations in algorithm class I-1 that may be applicable at the same time (although only one should be chosen). Similarly for algorithm class I-2. However, the translations in algorithm class I-1 apply to a disjoint set of database states from the translations in algorithm class I-2. In particular, a database state has at least one valid translation from algorithm class I-1 or from algorithm class I-2 *but not both*. Note that there are translators which are formed by combinations of translations of algorithms in classes I-1 and I-2, including those which can translate all legal view update requests.

4-4 Translation of Deletion Requests

The request is to delete a single, fully-specified view tuple. The deleted view tuple must currently be in the view.

ALGORITHM CLASS D-1: Delete the database tuple whose key matches that of the view tuple to be deleted. There is only one algorithm in class D-1 for each view update request.

ALGORITHM CLASS D-2: Replace the database tuple whose key matches that of the view tuple to be deleted, changing one non-key selecting attribute to an arbitrary excluded value. There is an algorithm in class D-2 for every non-key excluding value. Of course, there is no algorithm in class D-2 if the selection condition is "true" (i.e., there is no select clause) or if the set of selecting attributes is a subset of the key.

THEOREM. The set of update translations that satisfy the 5 criteria for candidate update translations for individual view deletions are precisely those in algorithm classes D-1 and D-2.

Let us consider when these translations may be used. The single algorithm in class D-1 is always applicable. The algorithms in class D-2 are applicable when they exist. One can consider algorithm class D-1 to be the inverse of algorithm class I-1, and algorithm class D-2 to be the inverse of algorithm class I-2. We note that these inverse are not perfect, which is why the insertion and deletion of the same tuple (or vice versa) is not necessary a no-op. Furthermore, there is no analog for deletion of an translator that combines algorithms from classes I-1 and I-2.

4-5 Translation of Replacement Requests

The request is to replace a single, fully-specified view tuple with another. The replaced view tuple must currently be in the view, and the replacement view tuple must currently not be in the view. Both tuples must

Delete / Insert	Algorithm class D-1 Delete replaced tuple	Algorithm class D-2 Replace (in database) replaced (view) tuple
Algorithm class I-1 No conflict for replacement view tuple	Algorithm class R-2 One replacement	Algorithm class R-3 Replace old view tuple Insert new view tuple
Algorithm class I-2 There is a database tuple whose key matches the replacement view tuple	Algorithm class R-4 Delete old view tuple Replace new view tuple	Algorithm class R-5 Replace both old and new view tuples

satisfy the selection condition. It must be possible to do the replacement in the view; that is, if there is a tuple in the original view whose key matches that of the replacement tuple, it must be the replaced tuple.

ALGORITHM EXTEND-REPLACE: Replace the database tuple changing the attributes appearing in the view to match those in the new view tuple. When used in algorithm classes R-1 and R-2, this will mean changing only those attributes that change in the view tuple replacement. This is similar to algorithm class I-2, except that the replaced database tuple *does* appear in the view. There is only one extend-replace algorithm.

ALGORITHM CLASS R-1: If the key does not change in the view tuple replacement, then perform algorithm extend-replace changing only those attributes that change in the view tuple replacement. Otherwise, reject the update request.

Note that if the key does not change, there is no possibility of a conflict with any tuple not appearing in the view.

Algorithms R-2 through R-5 handle the case where the key changes in the view update request, and are summarized in the chart above.

ALGORITHM CLASS R-2: Perform algorithm extend-replace if the key changes in the view tuple replacement and there is no tuple in the database whose key matches that of the replacement view tuple. Otherwise, reject the update request.

Algorithm class R-2 will not allow changes to database tuples not appearing in the view.

ALGORITHM CLASS R-3: If the key changes in the view tuple replacement and there *is* a tuple in the database whose key matches that of the replacement view tuple, then perform an algorithm of class I-2 (on the new view tuple) and delete the database tuple whose key matches that of the replaced view tuple. Otherwise, reject the update request.

Algorithm class R-3 changes a database tuple that does not appear in the view (because otherwise the view tuple replacement is not valid). If we consider the view replacement as a view deletion and a view insertion, then the dichotomy between Algorithm classes I-1 and I-2 parallels that between Algorithm classes R-2 and R-3 (respectively).

Algorithm classes R-2 and R-3 assume that no tuple remains in the database with a key matching that of the replaced view tuple. However, it is possible to replace (in the database) the replaced view tuple with a tuple with a matching key that does not satisfy the selection criteria. This parallels the dichotomy of Algorithm classes D-1 and D-2 for deletion. We obtain two algorithm classes using the same conditions of Algorithm classes R-2 and R-3, respectively.

ALGORITHM CLASS R-4: If the key changes in the view tuple replacement and there is no tuple in the database whose key matches that of the replacement view tuple, perform an algorithm from class D-2 (for the replaced view tuple) and an algorithm from class I-1 (for the replacement view tuple). (That is, the replaced view tuple will be changed to not appear in the view and the replacement view tuple will be inserted.) Otherwise, reject the update request.

ALGORITHM CLASS R-5: If the key changes in the view tuple replacement and there *is* a tuple in the database whose key matches that of the replacement view tuple, then perform an algorithm from class D-2 (for the replaced view tuple) and an algorithm from class I-2 (for the replacement view tuple). (That is, the replaced view tuple will be changed to not appear in the view and the replacement view tuple will be obtained by replacing some database tuple that did not appear in the view.) Otherwise, reject the update request.

Algorithm class R-1 is the only one possible when the replacement does not change the key. When the view replacement does change the key, we have two options for handling the replaced view tuple corresponding to Algorithm classes D-1 and D-2, and two situations involving the replacement view tuple corresponding to Algorithm classes I-1 and I-2. The four replacement algorithm classes are described by the table.

THEOREM. The set of update translations that satisfy the five criteria for candidate update translations for individual view replacements are precisely those that are generated by Algorithm classes R-1, R-2, R-3, R-4, and R-5.

5 Views Consisting of Selections, Projections, and Joins

We will now consider views defined using joins as well as selections and projections. We define a query to be in *Select-Project-Join Normal Form* (or *SPJNF*) when it does the selections first, the projections next, and the joins last. Note in particular that this implies that the join attributes must appear in the view.

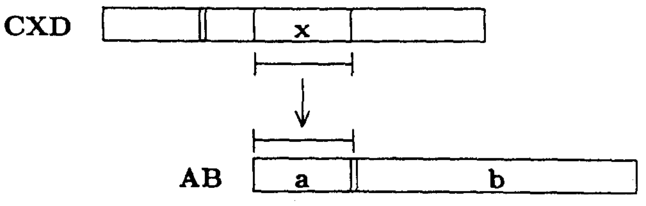
THEOREM. Any relational query where no projection removes a join attribute and the selection conditions are conjunctions of the form "attribute in set" can be converted into an equivalent (results in the same answer) relational query that is in SPJNF.

A view in SPJNF is the composition of a view consisting of joins with some number of select and project views (each on the individual relations), any of which may be the identity. We shall show how to update through the join view and then prove that composing the views works as expected.

5-1 Requirements for Joins

There are two requirements for joins. First, each join must be an extension join with an inclusion dependency. Second, the combination of joins must have a particular pattern.

Each join must be a reference connection. A reference connection [El-Masri 79, 80, Wiederhold 83] is the combination of an extension join [Honeyman 80] and an inclusion dependency [Casanova 82]. In an extension join, the join attributes are the key of one of the relations. In the figure, the join attributes are **X** in relation **CXD** and **A** in relation **AB**. Notice that **A** is



Reference Connection

the key for relation **AB**. In addition, we require that for each value **X** in **CXD**, there is a corresponding tuple in **AB** with a matching **A** value.

We can obtain a *query graph* by constructing a graph where each node corresponds to a relation in the view and each edge corresponds to a join in the view definition [Finkelstein 82]. The edges have directions as shown in the figure (in the many-to-one direction). We shall require that the join views correspond to a query graph that is a tree where all edges are directed away from a single root and each node refers to a unique relation.* Note that the key of the root is the key of the entire view.

We will call the class of views in SPJNF where the joins satisfy the two requirements of this section (reference connection and rooted tree), the projections do not remove any key, and the selection conditions are conjunctions of the form $A \in s$ where s is a set of constants in the domain for attribute A.

5-2 Updating Join Views

In this section, we consider the problem of updating join views. Each relation of the join view corresponds to an underlying relation of the database or a select and project view of such a relation. Of course, the SP view could be the identity view (i.e., no selection or projection). Let us first consider how to delete a tuple from the join view.

ALGORITHM CLASS SPJ-D: Delete the tuple from the root relation (or SP view) only using one of the algorithm of classes D-1 or D-2.

THEOREM. Algorithm class SPJ-D is the only algorithm that satisfies the five criteria that deletes a tuple from join views of the type specified when the SP views are identity views.

We next consider inserting a tuple into the join view. This involves inserting the various projections of the new join view tuple into the individual relations.

* We can relax this constraint to allow rooted DAGs if we relax the five criteria somewhat.

ALGORITHM CLASS SPJ-I: Take the projections of the join view to the attributes listed in each SP view. On each projection (or SP view) there are three cases:

CASE 1: The projection exists in the SP view in the exact projected form. If this is the root SP view, reject the update as it violates an FD in the view. Otherwise, we need do nothing with this SP view.

CASE 2: The projection does not match the key of any tuple in the SP view. Perform an SP view insertion using the projection of the new join view tuple.

CASE 3: There is already a tuple in the SP view with a key matching that of the projection, but the other values do not match. Replace (in the SP view) the existing SP view tuple by the projection of the new join view tuple. We may reject the update request if we do not wish to perform a replacement in the SP view.

If any of the SP view operations fail, the entire view update request fails and is undone.

THEOREM. Algorithm class SPJ-I is the only algorithm that satisfies the five criteria that inserts a tuple from join views of the type specified when the SP views are identity views.

We next consider replacing an individual tuple in the join view.

ALGORITHM CLASS SPJ-R: Perform a recursive (preorder [Knuth 73]) search on query graph tree. (We shall ignore the retracing steps that occur when leaf nodes are reached.) We are initially in State R at root relation.

STATE R (replacing): Compare projection (to this SP view) of old join view tuple with new join view tuple.

CASE R-1: Projections match exactly. Move to next relation down. Go to State R.

CASE R-2: Projections differ but keys match. Perform SP view replacement. Move to next relation down. Go to State I.

CASE R-3: Projections differ and keys differ. This can only happen in root. Perform SP view replacement. Move to next relation down. Go to State I.

STATE I (inserting): Compare projection (to this SP view) of old view tuple with new view tuple.

CASE I-1: Keys match. Go to State R (staying in this relation).

CASE I-2: Keys differ, new key not in SP view. Insert tuple into SP view. Move to next relation down. Go to State I.

CASE I-3: Keys differ, new projection in SP view. Move to next relation down. Go to State I.

CASE I-4: Keys differ, new key in SP view but conflicting data. Perform SP view replacement. Move to next relation down. Go to State I.

THEOREM. Algorithm class SPJ-R is the only algorithm that satisfies the five criteria that replaces a tuple from join views of the type specified when the SP views are identity views.

5-3 Combining Joins Views with Select and Project Views

We need to combine select and project view algorithms with join view algorithms to get select, project, and join view algorithms. Fortunately, the natural composition works correctly.

For a given select, project, and join view, the set of view update translations (translators) is the obtained from cartesian product of the sets of the view update translations (translators) for each select and project view. We use the one of the algorithms SPJ-D, SPJ-I, and SPJ-R as appropriate. Each algorithm describes how to use select and project view algorithms. This notion is captured in the following theorems.

LEMMA. Let U_1 and U_2 be a set of view update requests on the select and project views in sets V_1 and V_2 respectively, where there is at most one view update request on each view and each underlying relation is referenced in only one of the views V_1 or V_2 but not both. Let T_1 (T_2) contain one translation for each view update request in U_1 (U_2). Let the sets T_1 and T_2 each collectively satisfy the five criteria for view update translation. Then $T = T_1 \cup T_2$ collectively satisfies the five criteria for the view update requests $U = U_1 \cup U_2$ on the views $V = V_1 \cup V_2$.

THEOREM. Algorithm class SPJ-D are the only algorithms that satisfies the five criteria that deletes a tuple from select, project, and join views of the type specified.

THEOREM. Algorithm class SPJ-I are the only algorithms that satisfies the five criteria that inserts a tuple from select, project, and join views of the type specified.

THEOREM. Algorithm class SPJ-R are the only algorithms that satisfies the five criteria that replaces a tuple from select, project, and join views of the type specified.

6 Conclusion

We have devised five criteria for acceptable view update translations. We have enumerated a complete list of translators that satisfy these five criteria for a large class of select, project, and join views on Boyce-Codd Normal Form relations. Our techniques take into account the possibility that an object the user has requested to be deleted should actually be transformed into an object the user does not know about, and the possibility that an object the user wants inserted may refer to an existing object the user has just become aware of. Thus an object can be deleted by "destroying" it or converting it into another, unrecognizable object.

With a complete list of alternative translations, we have circumscribed the search space for a translator for view updates (into database updates). Additional semantics are needed to choose the desired translator. Collecting, coding, and using such additional semantics are beyond the scope of this paper.

We handle a large class of select, project, and join views on Boyce-Codd Normal Form relations. That is, there is a single consistency constraint on each relation: a key dependency (or functional dependency). The selection condition is the (possibly empty) conjunction of terms, each of the form *attribute* $\in$ *set*. The projection may remove any attributes mentioned in the selection condition, except that the key of the relation must appear in the view. The views are described in Select-Project-Join Normal Form, which requires that all the join attributes appear in the view, the joins are extension joins with inclusion dependencies, and the joins can be represented as a tree in a directed query graph.

7 References

[Bancilhon 79] F. Bancilhon, "Supporting View Updates in Relational Data Bases," in *Data Base Architecture*, Bracci and Nijssen, eds., North Holland, June 1979.

[Bancilhon 81] F. Bancilhon and N. Spyratos, "Update Semantics and Relational Views," *ACM Trans. on Database Systems*, **6**:4, December 1981.

[Carlson 79] C. Robert Carlson and Adarsh K. Arora, "The Updatability of Relational Views Based on Functional Dependencies," *Third International Computer Software and Applications Conference*, IEEE Computer Society, Chicago, IL, November 1979.

[Casanova 82] Marco Casanova, Ronald Fagin, and Christos Papadimitriou, "Inclusion Dependencies and Their Interaction with Functional Dependencies," *Proc. of the ACM Symp. on Princ. of Database Systems*, Los Angeles, March 1982.

[Davidson 81] Jim Davidson and S. Jerrold Kaplan, "Natural Language Access to Databases: Interpretation of Update Requests," *Proc. 7th Int. Joint Conf. on Artificial Intelligence*, Vancouver, B.C., August 1981.

[Dayal 78] U. Dayal and P. A. Bernstein, "On the Updatability of Relational Views," *Proc. Fourth VLDB Conf.*, IEEE Computer Society, Berlin, West Germany, October 1978.

[Dayal 79] Umeshwar Dayal, *Schema-Mapping Problems in Database systems*, Aiken Computation Laboratory, Harvard University, TR-11-79, Ph.D. dissertation, August 1979.

[Dayal 82] U. Dayal and P. A. Bernstein, "On the Correct Translation of Update Operations on Relational Views," *ACM Trans. on Database Systems*, **7**:3, September 1982.

[El-Masri 79] Ramez El-Masri and Gio Wiederhold, "Data Models Integration using the Structural Model," *Proc. of the 1979 SIGMOD Conference*, ACM SIGMOD, Boston, June 1979.

[El-Masri 80] Ramez El-Masri, *On the Design, Use, and Integration of Data Models*, Ph.D. dissertation, Stanford University, 1980.

[Finkelstein 82] Sheldon Finkelstein, "Common Expression Analysis in Database Applications," *Proc. Int. Conf. on Management of Data*, ACM SIGMOD, Orlando, FL, June 1982.

[Furtado 79] A. L. Furtado, K. C. Sevcik, and C. S. DOS Santos, "Permitting Updates Through Views of Data Bases," *Inform. Systems*, **4**:4, Pergamon Press, Great Britain, 1979.

[Kaplan 81] S. Jerrold Kaplan and Jim Davidson, "Interpreting Natural Language Database Updates," *Proc. 19th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, June 1981.

[Honeyman 80] Peter Honeyman, "Extension Joins," *Proc. Int. Conf. on Very Large Data Bases*, Montreal, 1980.

[Knuth 73] Donald E. Knuth, *The Art of Computer Programming, Volume 1, Fundamental Algorithms*, Addison-Wesley, Reading, MA, second edition, 1973.

[Keller 82] Arthur M. Keller, "Updates to Relational Databases Through Views Involving Joins," in *Improving Database Usability and Responsiveness*, Peter Scheuermann, ed., Academic Press, New York, 1982.

[Keller 84] Arthur M. Keller and Jeffrey D. Ullman, "On Complementary and Independent Mappings on Databases," *1984 ACM SIGMOD Int. Conf. on Management of Data*, Boston, June 1984.

[Keller 85a] Arthur M. Keller, "Choosing a View Update Translator by Dialog at View Definition Time," submitted for publication.

[Keller 85b] Arthur M. Keller, *Updating Relational Databases Through Views*, Stanford University, Computer Science Dept., Ph.D. dissertation, 1985.

[Maier 83] D. Maier, *Theory of Relational Databases*, Computer Science Press, Rockville, MD. 1983.

[Masunaga 83] Y. Masunaga, "A Relational Database View Update Translation Mechanism," IBM, San Jose Reserach Laboratory, Report RJ3742, 1983.

[Stonebraker 75] Michael Stonebraker, "Implementation of Integrity Constraints and Views by Query Modification," *Proc. of the 1975 SIGMOD Conference*, ACM SIGMOD, San Jose, June 1975.

[Ullman 82] Jeffrey D. Ullman, *Principles of Database Systems*. Computer Science Press, Potomac, MD, second edition. 1982.

[Wiederhold 83] Gio Wiederhold. *Database Design*, McGraw-Hill, Second edition, 1983.