Choosing a View Update Translator by Dialog at View Definition Time

Arthur M. Keller
University of Texas at Austin

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. Previously, we enumerated all translations of view updates into database updates that satisfy five criteria. This enumeration shows that the problem of translating view updates to database updates is inherently ambiguous. We give examples of structurally similar views that should have different translations because of the real world semantics. We propose that these semantics be obtained at view definition time. We show how this can be done through a structured dialog with the database administrator to choose a view update translator at view definition time. The questions asked during this dialog are based on the view definition, database structural schema information, and the answers to earlier questions in the dialog. Based on these questions a specific translator is chosen. Using this translator, user-specified view updates can be translated into database updates without the need for any disambiguating dialog. However, dialog with the user may be desired to confirm that the (view) side effects resulting from the user's view update request are acceptable.

KEYWORDS. Relational databases, database theory, view update.

1 Introduction

We know how to answer queries expressed against views, but we do not completely understand how to handle updates expressed against views. These queries are translated to queries against the underlying database through query modification [Stonebraker 75]. However, because view updates can be ambiguous, updates must currently be specified against the underlying database rather than against the view. Many researchers have considered the problem of translating updates expressed against views into updates expressed against the underlying database [Bancilhon 81, Brosda 85, Carlson 79, Clemons 78, Cosmadakis 84, Davidson 83, Dayal 82, Furtado 79, 85, Hegner 84, Kaplan 81, Keller 82, 84, 85a, 85b, 86, Masunaga 83, Medeiros 85, Rowe 79, Salveter 84, Sevcik 78, Tuchermann 83].

Since in the common model of relational databases [ANSI 82], 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 as far as possible. This is described by the following diagram.

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

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

This work was started while the author was at the Computer Science Department of Stanford University. This work was supported in part by contract N39-84-C-0211 (the Knowledge Base Management Systems Project, Prof. Gio Wiederhold, PI) from the Defense Advanced Research Projects Agency and by contract AFOSR-80-0212 (Universal Relations, Prof. Jeff Ullman, PI) from the Air Force Office of Scientific Research, both of the United States Department of Defense, and by the Computer Sciences Research and Development Fund of The University of Texas at Austin. 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, the US Government, or the State of Texas.

Author's address: The University of Texas at Austin, Department of Computer Sciences, Austin, TX 78712-1188.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

changed precisely in accordance with the user's request. No side effects are necessary to translate updates expressed against select and project views. In some cases, updates expressed against views that involve joins cannot be translated unless some side effects are permitted.

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 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.

For a large class of select, project, join views, there is an enumeration of all translations of view updates into database updates [Keller 85a]. This enumeration shows that the problem of translating view update to database updates is inherently ambiguous. In Section 3, we illustrate this with two views that are structurally similar but whose semantics require different translations of view updates. We propose that semantics be obtained at view definition time to choose a translator that selects a translation for each view update request (or, alternatively, rejects the update request). We show how these semantics can be obtained and this translator chosen at view definition time through a dialog with the database administrator based on the view definition and structural schema information about 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 an 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 into 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. The corresponding underlying tuples to a view tuple are the database tuples with keys matching those appearing in the view tuple. The side effects occur as a result of view tuples sharing corresponding underlying tuples that undergo updates.

Requiring that a translation be valid is not sufficient for our purposes—it is only a first step. We have defined 5 additional criteria we require the translations to satisfy [Keller 85a]. These criteria proscribe database side effects, multiple changes to the same database tuple, unnecessary database changes, replacements that can be simplified, and delete-insert pairs on the same relation. We use the criteria to obtain only the simplest (or minimal) view update translations.

Because of space limitations, most background material has been omitted from this paper. Explanations of the need for and use of semantics can be found in [Keller 86], and more details on the criteria and on the space of algorithms can be found in [Keller 85a].

3 Dialog at View Definition Time

We propose that the semantics necessary for disambiguating view update translation be obtained at view definition time. The semantics are used to choose a view update translator. Once a translator is chosen, users may specify updates through the view, which the translator converts into database updates without any disambiguating dialog.

In the discussion that follows, we will assume that the view is defined by a database administrator (DBA) who will also provide the necessary semantics to choose a translator. While this is the simplest case for the use of a view definition facility, it is clear that this system could be used by any user with the wherewithal to define a view, either for the user's own use or for the use of other, perhaps less knowledgeable users. We regard the effort of collecting the semantics at view definition time to be amortized by utilizing them for many view updates.

The candidate translators can be organized into a tree, where each node of the tree represents a decision to be made. The semantics are merely the sequence of decisions made by the DBA in a walk of this tree guided by the view definition facility. The view definition facility presents questions to the DBA, each time supplying several options, based on the view definition, the database schema, and the answers to the previous questions. Note that the tree of translators is different from the query graph representing the view. Furthermore, the tree of translators is merely a pedagogical device; it does not actually exist within the view definition facility.

Choosing a translator does not use any information about the transactions that will be performed against the view. This is because the translator chosen will take as input individual view tuple updates and translate them into sets of database updates. Any information that would be contained in the nature of the transactions performed that is useful for determining how to translate the update is captured at the view definition dialog. Since the set of transactions is not necessarily available at view definition time, does not contain all the information needed for choosing a view update translator, and at best provides information that is already provided by the dialog, we have chosen to use the dialog instead. The dialog is described in subsequent subsections.

3-1 Theoretical Comments

For the class of views we handle, we have previously enumerated all possible *translations* of single view tuple updates into sequences of database updates that satisfy our five criteria [Keller 85a]. By selecting a translation for each specific view update request, we obtain a *translator* of view updates. The set of possible translations for each update request characterizes the set of possible translators.

Not all translators in this set are reasonable. We immediately reject those that make varying decisions based on extraneous information, such as the phase of the moon. It is acceptable to choose between different translators depending on such extraneous information, but each translator should make decisions solely based on the state of the database and the particular view update requested.

Other translators may make choices among the possible translations by treating particular domain values specially even though the view definition does not require it. For example, a translator may choose one translation for updates involving employees whose last name begins with 'A' through 'L' and another for employees whose last name begins with 'M' through 'Z'. Such translators are allowable but implausible.

We would like to define a *basis set* of translators from which all possible translations can be generated using transformation rules. The first transformation rule allows combination of two translators: Given translators T_1 and T_2 and predicate p dependent solely on the database state and view update request, there is a translator T_3 such that $T_3 =$ if p then T_1 else T_2. The second transformation rule acknowledges the fact that it is acceptable to reject a view update request. One formulation of this rule is: Given translator T_1 and predicate p dependent solely on the database state and view update request, there is a translator T_3 such that $T_3 =$ if p then T_1 else *reject request*. An alternative formulation of this rule is: There exists a translator T_0 that rejects all view update requests. Both alternative formulations of the second rule are equivalent when we adopt the first rule.

For the purpose of this discussion, we consider the positive decision to reject a view update request to be a translation. This is required by the second transformation rule above. However, the user or program that made the view update request must be notified that the request has been rejected, as would be required if an ordinary database update request were rejected.

There are many alternative basis sets that generate the set of all possible translators. We will choose one that most closely matches the decisions made by the update algorithms. We will not attempt to formally define this particular notion. We call the basis set we define $\mathcal{T}_0$. In addition, where it is convenient, we ask certain questions that result in additional translators that can be obtained using the second transformation rule (first formulation). These additional questions are marked '*' in the algorithms in the following sections. These additional translators, while not in the basis set, are useful in practice when the user has only limited authorization for updates. Thus, the set of translators actually obtained by our dialogs belong to a set, $\mathcal{T}_1$, that properly contains $\mathcal{T}_0$. Note also that $\mathcal{T}_1$ is also a basis set for the set of all possible translators that satisfy our five criteria, although not a minimal one. On the other hand $\mathcal{T}_0$ is minimal in the sense that any proper subset is no longer a basis set.

4 Dialog at View Definition: Deletion

Using the query graph [Finkelstein 82] that defines the view, we consider the selections and projections applied to the root relation. This is because deleting from a select, project, and join view can be accomplished by deleting from the select and project view corresponding to the root relation [Keller 85a]. If there is no selection on the root relation, there is no alternative to deleting the projection of the view tuple from the underlying relation (*i.e.*, the tuple with the same key as the view tuple). If there is a selection on the root relation, we may alternatively replace the corresponding underlying tuple in the root relation by changing a selecting attribute to an excluding value.

```
ALGORITHM DBA-D:
Ask: Are view tuple deletions permitted? *
If not, exit
If there is no selection on the root relation, or all
      selecting attributes of the root relation are part of
      the key
      Then deletion of a view tuple is done by deleting
            the corresponding root database tuple; exit
Ask: Should deletion of a view tuple result in deletion
      of the corresponding root database tuple (as in
      New York manager example) or its replacement
      (as in baseball team manager example)?
If deletion, then deletion of a view tuple is done by
      deleting the corresponding root database tuple;
      exit
Ask: Which of the selecting attributes in the root
      relation that are not part of the key (supply the
      list of them) is to be replaced?
If that attribute has more than one excluding value
      Then ask: Which excluding value (supply list)
            should be used for that attribute?
Deletion of a view tuple is done by replacing the
      corresponding root database tuple changing
      the specified selecting attribute to the specified
      excluding value
```

The attributes that have only one excluding value can be highlighted, since they are more likely to be desired. The view update translation chosen does not have any side effects in the view and only affects one database tuple.

5 Dialog at View Definition: Insertion

Inserting into a selection, project, and join (SPJ) view involves ensuring that the projections of the view tuple appear in each of the relations so that they may be joined together to form the view tuple. The SPJ view can be decomposed into a join view of a series of select and project (SP) views formed by taking the selections and projections of the query graph using them to define a view on each relation [Keller 85a]. The following algorithm decomposes a SPJ view tuple insertion into a series of operations on these SP views [Keller 85a].

INSERT INTO SPJ VIEW: 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 view 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.

```
ALGORITHM DBA-I:
Ask: Are view tuple insertions permitted? *
If not, exit
For each relation in the view (using a pre-order traversal
      of query graph)
      Ask: Are modifications permitted to this relation? *
```

```
If not, continue loop with next relation (When view
     insertions require changes to this relation, they
     will be rejected.)
Ask: Can a new tuple be inserted into this rela-
     tion? *
If not, view insertions that require insertion of a
     tuple into this relation are rejected
For each attribute in this relation that does not
     appear in the view
     If it is not a selecting attribute
          Then ask: Which domain value should be
               used for this attribute?
          Else if there is only one selecting value
               for this attribute
               Then use that one
               Else ask: Which selecting value
                    should be used for this attribute
(When inserting a tuple into the database, we take
     the projection of the view tuple to this relation
     and extend it with these values chosen.)
Ask: Can a view tuple insertion result in a change
     to a database tuple that does not satisfy the
     selection condition (as arises in the baseball
     team manager example)? *
If not, we reject view insertions that would require
     it
If so, we change the corresponding database tuple
     (key matches values in view tuple) so that its
     values match those of the inserted view tuple.
     If there are selecting attributes in this relation
          that do not appear in the view, we will
          have to change every excluding value in
          the corresponding database tuple to a
          selecting value.
          If we have obtained a list of values above
               Then use those here too
               Else ask: Choose a selecting value to
                    use for each selecting attribute
                    in this relation that has more
                    than one selecting value and
                    that does not appear in the view
(Here we do not need values for non-selecting
     attributes that do not appear in the view, as
     they remain unchanged.)
Ask: Can a view tuple insertion result in a change
     to a database tuple that does satisfy the
     selection condition? *
If not, view insertions requiring this are rejected
If so, the translation includes changing the database
     tuple so that it matches the projection of the
     view tuple to that relation. Such a change
     results in a side effect when the database
     tuple changed is one of the corresponding
     underlying tuples for some other view tuple.
     Then the change requested will affect those
     other view tuples that share this corresponding
     underlying tuple.
```

The implementor of a view update facility may offer the option of allowing the DBA to indicate that a view update is to be rejected if a particular non-appearing selecting attribute does not already have a selecting value in lieu of giving a selecting value to change it to; this is only meaningful when there are multiple selecting attributes, at least one of which does not appear in the view.

Note that most of the questions in this part of the dialog are starred (*). The only ones necessary for $\mathcal{T}_0$ are those that define values for attributes that do not appear in the view that are not otherwise constrained.

6 Dialog at View Definition: Replacement

Replacing in a SPJ view can be decomposed into a series of replacements and insertions in select, project views on the underlying relations. The following algorithm describes this process [Keller 85a].

REPLACE INTO SPJ VIEW: Perform pre-order traversal on query graph tree. 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 if allowed. 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 if allowed. 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. SP view insert tuple. 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. SP view replace if desired, else reject request. Move to next relation down. Go to State I.

Cases R-1, I-1, and I-3 require no action. Case I-2 can use the same algorithm as Case 2 from the previous

Old tuple / New tuple	Delete replaced tuple	Replace (in database) replaced view tuple
Insert into DB replacement view tuple	Scenario 1 One replacement	Scenario 3 Replace old view tuple Insert new view tuple
Replace in DB the replacement view tuple	Scenario 2 Delete old view tuple Replace new view tuple	Scenario 4 Replace both old and new view tuples

Scenarios for changing key of view tuple in replacement

section. Cases R-2 and I-4 can use the same algorithm as Case 3 from the previous section.

Case R-3 is more complicated. There are two alternative ways to remove the old database tuple based on the two alternatives for deleting a view tuple (delete and replace). There are two alternative ways to insert the new database tuple, depending on whether there is already a conflicting database tuple there. In the case where the old database tuple is to be deleted and the new one is inserted, a replacement is performed instead.

```
ALGORITHM DBA-R:
Ask: Are view tuple replacements permitted? *
If not, exit
Ask: Can the key of a view tuple be changed? *
If so, and there is a selecting attribute in the root
      relation not part of the key, ask: Which of the
      following four scenarios is permissible?
(If there is no selecting attribute in the root relation not
      part of the key, the only possibility is scenario 1.)
   (1) Changing the key of the corresponding root
         database tuple
   (2) Deleting the old root tuple and replacing a
         root tuple that has the new key so that it
         does appear in the view
   (3) Replacing the old root tuple so that it does not
         appear in the view and inserting a new root
         tuple
   (4) Replacing the old root tuple so that it does not
         appear in the view and replacing a root tuple
         that has the new key so that it does appear
         in the view
   (Scenarios 1 and 3 are mutually exclusive as are 2
         and 4.)
   For scenarios 2 and 4, for each selecting attribute
         in the root relation not appearing in the view
         that has more than one selecting value
         Ask: Which selecting value should be used if
               the tuple does not already have a selecting
               value for this attribute? (Note that this
               question need not be asked if the answer
               is already available from the insertion
               algorithm dialog.)
   For each non-key attribute appearing in the view,
         ask: Can this attribute be changed? *
   For scenarios 3 and 4, ask the last two questions
         of the deletion dialog if not already done or
         the answers are not assumed to be the same
For each relation in the view other than the root (using
      a pre-order traversal of query graph)
   Ask: Are modifications permitted to this relation? *
   If not, continue loop with next relation (When
         view replacements require changes to this
         relation, they will be rejected.)
   For each non-key attribute appearing in the view,
         ask: Can this attribute be changed? *
   Ask: Can a new tuple be inserted into this rela-
         tion? *
   If not, view replacements that require insertion of
         a tuple into this relation are rejected
   (The following loop need only be performed when
         not asked during the insertion dialog, or if
         the answers to the questions asked are not
         assumed to be the same.)
   For each attribute in this relation that does not
         appear in the view
         If it is not a selecting attribute
               Then ask: Which domain value should be
                     used for this attribute?
               Else if there is only one selecting value
                     for this attribute
                     Then use that one
                     Else ask: Which selecting value
                           should be used for this attribute
   (When inserting a tuple into the database, we
         take the projection of the new view tuple to
         this relation and extend it with these values
         chosen.)
   Ask: Can a view tuple replacement result in a
         change to a database tuple that does not
         satisfy the selection condition (as arises in the
         baseball team manager example)? *
```

```
If not, we reject view replacements that would
    require it
If so, we change the corresponding database tuple
    (key matches values in new view tuple) so
    that its values match those of the inserted
    view tuple.
    (The following questions need only be asked
        when not asked during the insertion
        dialog, or if the answers are not assumed
        to be the same.)
    If there are selecting attributes in this relation
        that do not appear in the view, we will
        have to change every excluding value in
        the corresponding database tuple to a
        selecting value.
        If we have obtained a list of values before
            Then use those here too
            Else ask: Choose a selecting value to
                use for each selecting attribute
                in this relation that has more
                than one selecting value and
                that does not appear in the view
Ask: Can a view tuple replacement result in a
    change to a database tuple that does satisfy
    the selection condition? *
If not, view insertions requiring this are rejected
If so, the translation includes changing the database
    tuple so that it matches the projection of the
    view tuple to that relation. Such a change
    results in a side effect when the database
    tuple changed is one of the corresponding
    underlying tuples for some other view tuple.
    Then the change requested will affect those
    other view tuples that share this corresponding
    underlying tuple.
```

The dialog for replacement requests can assume the answers to some questions asked during the deletion and insertion dialogs. Specific to replacement requests is the handling in the root relation when the view tuple replacement request includes a change to the key of the view tuple (and correspondingly, the root relation). In addition, it may be desirable to ask whether specific attributes may be changed, as we have shown in the dialog above.

7 Conclusion

We have described a method that can make updating relational databases through views reliable and convenient. The database administrator (DBA) (or any other sufficiently knowledgeable user) defines the view and answers a sequence of questions to choose a valid view update translator for a large class of select, project, and join views. The definition of the translator is stored along with the view definition. The class of translators we choose from are based on the algorithm templates that generate all possible translations that satisfy five criteria for view update translation [Keller 85a]. After the view and translator are defined, users may request insertions, deletions, and replacements through the view, and these will be translated by the chosen translator into database updates without any disambiguating dialog. Side effects may result from some insertions and replacements only for join views when translation is not otherwise possible and if permitted by the DBA; it may be desirable to have the user confirm such side effects, especially for insertions.

Not all possible translators are subject to being chosen by our questions. The set of candidate view update translators is quite large; we have characterized this set by enumerating the set of all view update translations in earlier work [Keller 85a]. Some of the translators chosen here will translate all updates that have translations satisfying our criteria; others will reject some updates because they were proscribed by the answers by the DBA to the questions asked by the view definition facility. The translators that accept all updates form a basis set for the set of all possible translators under two transformations. Some translators that reject some updates on a systematic basis are included to give the DBA more flexibility in defining a view update translator; this can be used as part of an effective security system. The translators obtained by the dialog are completely characterized by the answers to the questions in the dialog.

The process of defining a view and choosing a translator has been described here as being performed by the DBA. While this is the simplest case for the use of such a system, it is clear that this system could be used by any user with the wherewithal to define a view. The distinction to make is that such a dialog would be most effective for static views that are defined once and used repeatedly. For dynamic views, defined by natural language dialog or universal relation interfaces, the overhead of answering the questions would not be amortized over performing many view updates. Heuristics and user profiles could be used to determine the answers we need to choose a translator [Davidson 83].

Querying and updating through a view reduces the security and protection problem, but does not eliminate it. Clearly, a view circumscribes the collection of data a user is permitted to access. The question of how to give each manager access to the data for that department can be addressed either by a parameterizing the view to only show that department's data or by parameterized protection scheme that allows access only to tuples containing data for that department. Using both may seem redundant but need not be. A parameterized view will make fewer demands on the database and the security system. A security system could have a large loophole if it gave special consideration to queries and updates specified through views. Of course, an effective security system is needed when a view definition facility and an *ad hoc* query facility is made available to users.

With views and queries described non-procedurally, relational databases are an effective tool for productivity [Codd 82]. We have shown how to describe view update translators non-procedurally by answering a sequence of questions based on the view definition and the database structure. This has the potential to dramatically increase the productivity of views, and consequently, relational databases.

8 Acknowledgements

Gio Wiederhold and Jeff Ullman provided support, advice, and encouragement. Moon Ho Chung and Nazir Alimohammad programmed the algorithms presented here. The implementation effort was continued by Laurel Harvey, who also implemented a prototype view updater that executes the view update translator chosen by the algorithms presented in this paper. This paper is dedicated to the memory of Joey Sussman, who taught me about professionalism in computer programming.

8 Bibliography

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

[ANSI 82] "Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group," in *SIGMOD Record*, **12**:4, July 1982.

[Brosda 85] Volkert Brosda and Gottfried Vossen, "Updating a Relational Database through a Universal Schema Interface," *4th PODS*, March 1985.

[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.

[Clemons 78] E. K. Clemons, "An External Schema Facility to Support Data Base Updates," in *Databases: Improving Usability and Responsiveness*, Academic Press, 1978.

[Codd 82] E. F. Codd, "Relational Database: A Practical Foundation for Productivity," *Comm. ACM*, **25**:2, February 1982.

[Cosmadakis 84] Stavros S. Cosmadakis and Christos H. Papadimitriou, "Updates of Relational Views," in *Journal of the Assoc. Comput. Mach.*, **31**:4, October 1984.

[Davidson 83] J.E. Davidson, "Interpreting Natural Language Database Updates," Stanford University, Computer Science Dept., Ph.D. dissertation, December 1983.

[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.

[Finkelstein 82] Sheldon Finkelstein, "Common Expression Analysis in Database Applications," *Proc. Int. Conf. on Management of Data*, ACM SIGMOD, 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, 1979.

[Furtado 85] A. L. Furtado and M. A. Casanova, "Updating Relational Views," in *Query Processing in Database Systems*, W. Kim, D. S. Reiner, and D. S. Batory, eds., Springer-Verlag, 1985.

[Hegner 84] Stephen J. Hegner, "Canonical View Update Support through Boolean Algebras of Components," *3rd PODS*, ACM, April 1984.

[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.

[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, "Algorithms for Translating View Updates to Database Updates for Views Involving Selections, Projections, and Joins," *4th PODS*, ACM, March 1985.

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

[Keller 86] Arthur M. Keller, "The Role of Semantics in Translating View Updates," *IEEE Computer*, 19:1, January 1986, pp. 63–73.

[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.

[Medeiros 85] C.M.B. Medeiros, "A Validation Tool for Designing DSatabase Views that Permit Updates," Ph.D. dissertation, Data Structuring Group, Dept. of Computer Science, University of Waterloo, November 1985.

[Rowe 79] L. Rowe and K. A. Schoens, "Data Abstractions, Views, and Updates in RIGEL," Proc. ACM SIGMOD Int. Conf. on Management of Data, May 1979.

[Salveter 84] Sharon Salveter, "A Transportable Natural Language Database Update System," *3rd PODS*, ACM, April 1984.

[Sevcik 78] K. C. Sevcik and A. L. Furtado, "Complete and Compatible Sets of Update Operators," *Proc. Int. Conf. on Management of Data*, ACM, June 1978.

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

[Tuchermann 83] L. Tuchermann, A. L. Furtado, M. A. Casanova, "A Pragmatic Approach to Structured Database Design," *Proc. 9th VLDB Conference*, October 1983.

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