A PROTOTYPE VIEW UPDATE TRANSLATION FACILITY

Arthur M. Keller and Laurel Harvey

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-46 December 1987

A Prototype View Update Translation Facility

Arthur M. Keller and Laurel Harvey
University of Texas at Austin

ABSTRACT. We describe a prototype implementation of a view update translation facility. This facility consists of two programs: one for defining the view and specifying the view update semantics and the other for translating view updates into database updates based on these semantics. The first program is run once at view definition time and obtains the necessary semantics by asking the view definer a sequence of questions in an interactive dialog. The second program takes any valid requests to insert, delete, or replace a single view tuple and performs the necessary operations, if permitted, on the database to accomplish the view update request without any disambiguating dialog. Prototype implementation was simplified by not actually using a relational database system but rather all interaction normally with the database instead is with the person running the program. This has the advantages of clearly showing all database operations and it eliminates the need to set up a database with desired test or demonstration cases. The database operations for performing a view update are those that are necessary for validating the request and changing the database in accordance with the specified semantics so that the view changes as requested. Unnecessary database operations have not been observed in this prototype system. The class of views that can be updated using this prototype system is a large class of selection, projection, and join views.

1 Introduction

In shared relational databases, views provide a way to give the database user only the information relevant to the user. Views may be defined for each class of user, and users may express queries and updates against them. The problem of answering queries expressed against views is well understood: The user's query is composed with the view definition to obtain a query that can be executed on the underlying database. The handling of updates expressed against views is more complex because view update requests must be translated into requests on the underlying database, because the translation process involves ambiguity, and because view updates potentially have far-reaching effects on underlying database relations and other views.

Since the view is only an uninstantiated window onto the database in the common model of relational databases [ANSI 82], any updates specified against the database view must be translated into updates against the underlying database. This updated database state induces a new view state, and it is desirable that the new view state correspond as closely as possible to performing the user-specified update directly on the original view. This is described by the following diagram.

$$\begin{tikzcd}
V(DB) \arrow[r, "U"] \arrow[d, Rightarrow, "T"] & U(V(DB)) \overset{?}{=} V(DB') \\
\text{DB} \arrow[u, "V"] \arrow[r, "T(U)"] & T(U)(DB) = DB' \arrow[u, "V"']
\end{tikzcd}$$

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. 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. The problem of translating updates expressed against views into updates expressed against the underlying database has been considered by many researchers [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]. The ways in which individual view update requests may be satisfied by database updates must be understood. Since the view is many-to-one, the new view state may correspond to many database states. We would like to choose the database state that is "as close as possible" under some measure to the original database state, minimizing the effect of the view update on the database. Five criteria used to obtain only the simplest (or minimal) view update translations proscribe (1) database side effects,

This work was supported in part by the Computer Sciences Research and Development Fund and the University Research Institute of The University of Texas at Austin.

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

(2) multiple changes to the same database tuple, (3) unnecessary database changes, (4) replacements that can be simplified, and (5) delete-insert pairs on the same relation [Keller 85a].

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 updates to database updates is inherently ambiguous. In earlier work [Keller 86a], we illustrated this ambiguity by showing two views that are structurally similiar but whose semantics require different view update translators. We have proposed [Keller 85a, 85b, 86a, 86b] that semantics be obtained at view definition time and choice of a translator made through use of a dialog with the database administrator (or other knowledgeable person). The dialog would be based on the view definition, structural schema information about the database, and answers to previous questions in the dialog.

We have previously described the class of view update translators [Keller 85a], the role of semantics in choosing a translator [Keller 86a], and algorithms that obtain at view definition time the semantics necessary to choose a view update translator [Keller 86b]. In this paper, we describe a prototype implementation of these algorithms and an interpreter for the translator they choose.

2 Prototype Programs

For a large class of views, we have described an algorithm for this dialog [Keller 86b]. Two prototype programs have been written to implement this specific class of view updates.

Figure 1 describes how these programs are intended to fit into a production database environment. The first program, the view definition facility, obtains the relation and view definitions and then engages in a dialog with the view definer to obtain the semantics to choose a specific view update translator. This translator is described in a translator specification file (TSF). The second program, the view update facility, implements view updates requested by the user by translating them into database operations according to the translator stated in the TSF. These two programs do not depend on a particular database system; rather, they are standalone prototype programs that have not been interfaced to a particular database system. Instead, all database operations are simulated by printing the operation requested on the terminal and having the terminal operator enter the results of the operation. This has the benefits of explicitly showing all interaction with the database and

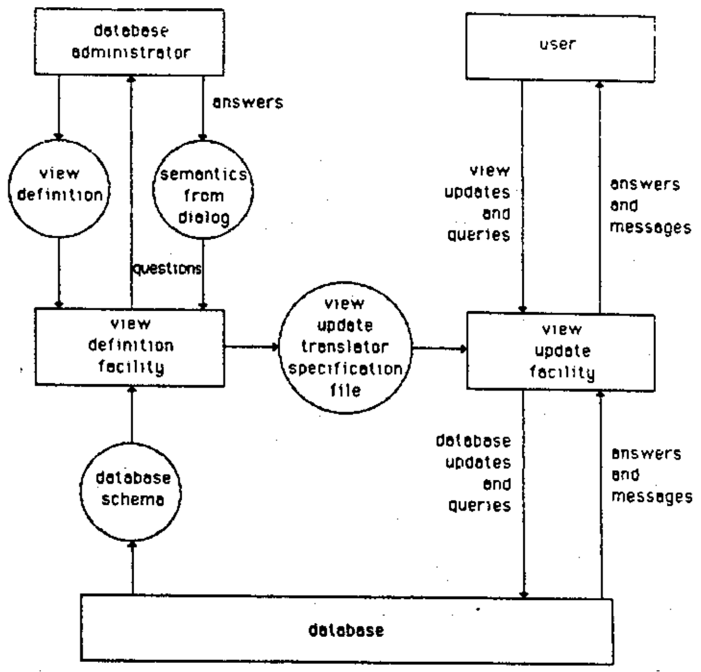

Figure 1

eliminating the need to preload the database with test or demonstration data.

3 Class of Views

The views handled by the prototype programs are a large class of select, project, and join views on Boyce-Codd Normal Form relations. The views must meet the following restrictions [Keller 85a, 85b, 86a]:

JOIN RESTRICTIONS The key of the root relation of the view must be a key or foreign key of every relation. The joins must all be extension joins that cascade from this root relation in a tree pattern.

PROJECTION RESTRICTIONS Relation keys and join attributes may not be removed from the view by projection. However, an attribute equated to another attribute by a join in the view may substitute for the latter attribute in the view.

SELECTION RESTRICTIONS The terms of the selection condition may only test whether an attribute is or is not a member of a constant set of domain values. The WHERE clause in the view definition may only be a conjunction of selection terms and equality comparisons specifying joins.

Formal descriptions, examples, and explanations of the class of views handled may be found in our prior work [Keller 85a, 85b, 86a, 86b].

4 Updates

A (database or view) update is either a deletion, an insertion, or a replacement of a single tuple. 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. View updates must satisfy relevant constraints on the database or view, such as key dependencies, and their effects must be visible to the view.

A view defines a virtual relation that appears as a stored relation to the user. Queries and updates on the view require that the database system compensate for the fact that the view contains no data itself but only reflects data actually stored in database relations.

5 Relation and View Definitions

Views may be structurally similar but semantically different, so that they require different view update translators. Two views which display this semantic difference will be described here and used in the following sections.

Before we can describe the views, we must describe the underlying database relations appearing in the views. We will define these using the modified SQL syntax used in the prototype View Definition Facility. We use the convention for this syntax that all capitals are used for reserved words, initial capitals for relation and view names, and lower case for attribute names.

```
DEFINE RELATION Employee (lastname,
    firstname :> phone, dept, loc,
    project, onbbteam);
DEFINE RELATION Dept (deptname :> manager,
    division);
```

As shown in this example, the relation name is followed by the attribute names enclosed in parentheses. The list of attributes in the key are followed by :> and the list of non-key attributes. Both lists contain attribute names separated by commas.

The personnel manager for New York, Susan, uses the following view to select employees working in New York.

```
DEFINE VIEW Nypersmgr AS
SELECT Employee.lastname, Employee.firstname,
    Employee.phone, Employee.dept,
    Dept.manager
FROM (Employee (Dept))
WHERE Dept.deptname = Employee.dept
    AND Employee.loc IN SET [NY];
```

There are several simplifications of SQL in this syntax. The SELECT clause contains only attributes qualified by relation names. The FROM clause is written in LISP tree notation; any cascaded extension join tree may be described this way. The WHERE clause consists of a conjunction of equality comparisons for joins and terms of the form *attribute* IN SET [$a, b, \ldots$] or *attribute* NOT IN SET [$a, b, \ldots$].

The key of the view is not explicitly specified; the key of the root relation (the root of the extension join tree) becomes the key of the view, as it is a key or foreign key for every attribute in the view.

The baseball team manager, Frank, uses the following view.

```
DEFINE VIEW Bbteammgr AS
SELECT Employee.lastname, Employee.firstname,
    Employee.phone, Employee.dept,
    Dept.manager
FROM (Employee (Dept))
WHERE Dept.deptname = Employee.dept
    AND Employee.onbbteam IN SET [YES];
```

These two views are structurally and syntactically similar, but have differing semantics. In the next section, we will consider the dialog part of the View Definition Facility and show how these semantics are obtained to choose differing view update translators.

6 View Definition Facility

When the View Definition Facility obtains a view definition, it verifies that the view is capable of being updated and then asks the view definer (such as the database administrator) information used in choosing the view update translator. The class of possible view update translators for a given view can be arranged into a decision tree. This decision tree can be traversed so that the view definer is asked a question at each node, until the desired translator is reached at a leaf of the decision tree. This decision tree is not actually created; rather it represents the tree of computational paths through the dialog algorithm. The algorithm for this dialog is presented in previous work [Keller 86b].

The dialog from the prototype View Definition Facility for the baseball team manager's view is given below. This dialog has been edited because of space limitations. System questions are written in this typeface, while user responses are written in `this typeface`.

Is deletion of view tuples allowed?
YES
Should deletion of a view tuple result in DELETION of the corresponding tuple in relation Employee or its REPLACEMENT?
REPLACE
You will replace the value of the attribute onbbteam from relation Employee. Choose a value *not* in this set: [YES]
NO
View tuple deletions are permitted and are done by replacing Employee.onbbteam with the value NO.
Is insertion of view tuples allowed?
YES
Are modifications to the Employee relation permitted?
YES
Can a new tuple be inserted into the Employee relation?
NO
Can a view tuple insertion result in a change to a tuple in this relation which presently is not satisfying a selecting condition needed for appearance in the view?
YES
If a view tuple is inserted, should Employee.onbbteam be changed to the value YES?
YES
Are modifications to the Dept relation permitted?
NO
Insertion of a view tuple now is defined.
Is replacement of view tuples allowed?
YES
Can the root relation (Employee) tuple of a view tuple be modified?
YES
Can the key of a view tuple (the root tuple key) be changed?
YES
Should deletion of a view tuple result in DELETION of the corresponding root relation (Employee) tuple or its REPLACEMENT?
REPLACE
You will replace the value of Employee.onbbteam
Choose a value *not* in this set: [YES]
NO
Can a new tuple be inserted into the Employee relation?
NO
Can a view tuple replacement result in a change to a tuple in this relation which presently is not satisfying a selecting condition needed for appearance in the view?
YES
Can a view tuple replacement result in a change to a tuple in this relation which presently *does* satisfy the selecting condition(s) needed for appearance in the view?
YES
Can the non-key attribute Employee.phone be changed?
YES
Can the non-key attribute Employee.dept be changed?
YES
Are modifications to the Dept relation permitted?
NO
Replacement of a view tuple now is defined.

After the dialog is completed, the prototype View Definition Facility prints a summary of the dialog and translator defined. The edited summary of the New York manager's view follows. (Not the baseball team view!)

```
View tuple deletions are permitted.
    Deletion of a view tuple is done by deleting the
    corresponding tuple from root relation Employee.
View tuple insertions are permitted.
    For relation Employee:
        Modifications are permitted.
        A new tuple can be inserted into this relation.
        View tuple insertion can result in change
        to a relation tuple not satisfying a selecting
        condition.
        Non-view, non-selecting attributes and values:
            project = available
            onbbteam = no
        Non-view, selecting attributes and values:
            loc = NY
    For relation Dept
        Modifications are permitted.
View tuple replacements are permitted.
    For relation Employee:
        Modifications are permitted.
        The key of a view tuple can be changed.
            The old root tuple is deleted.
        A new tuple can be inserted into this relation.
        View tuple replacement can result in change
        to a relation tuple not satisfying a selecting
        condition.
        View tuple replacement can result in change to
        a relation tuple satisfying selection conditions.
        Non-view, selecting attributes and values:
            loc = NY
```

Non-key, in-view attributes: (YES = value can be changed)
  phone = YES
  dept = YES
For relation Dept:
  Modifications are *not* permitted.
  A new tuple can *not* be inserted into this relation.
  View tuple insertion can result in change to a relation tuple not satisfying a selecting condition.
  View tuple insertion can result in change to a relation tuple *satisfying* selection conditions.
  Non-key, in-view attributes: (YES = value can be changed)
    manager = YES

The output of the View Definition Facility is the Translator Specification File (TSF). This file is described in the next section.

7 View Update Translator Specification File

The Translator Specification File (TSF) tells how view updates are to be translated into database updates. The semantics obtained during the dialog in the View Definition Facility are encoded in a precise description of the translator chosen. In our prototype, the relation definitions appear at the beginning of the TSF, followed by the information for each view: the view definition and its view translator.

The TSF is designed to be easily parsed by the View Update Facility but still be intelligible to facilitate debugging the prototype View Definition Facility and checking that the correct translator is chosen.

The translator for the baseball team manager's view is described as follows in the TSF.

```
DIALOG PART = DELETION
ALLOWED = YES
RELATION = Employee
METHOD = REPLACE
RSET onbbteam = NO
DIALOG PART = INSERTION
ALLOWED = YES
RELATION = Employee
METHOD = CHANGE_TUPLE
CHANGE_NON_SATISFYING_SELECT_COND_ALLOWED = YES
SSET onbbteam   = YES
RELATION = Dept
METHOD = NO_CHANGE
DIALOG PART = REPLACEMENT
ALLOWED = YES
RELATION = Employee
KEY_CHANGE_ALLOWED = YES
METHOD = REPLACE_OLD
RSET onbbteam = NO
NEW_TUPLE_ALLOWED = NO
CHANGE_NON_SATISFYING_SELECT_COND_ALLOWED = YES
CHANGE_SATISFYING_SELECT_COND_ALLOWED = YES
SSET onbbteam   = YES
VIEW_NON_KEY_ATTR_CHANGE_ALLOWED?
CHANGE phone        = YES
CHANGE dept         = NO
RELATION = Dept
METHOD = NO_CHANGE
/END OF DIALOG/
```

The translator for the New York personnel manager's view is described as follows in the TSF.

```
DIALOG PART = DELETION
ALLOWED = YES
RELATION = Employee
METHOD = DELETE
DIALOG PART = INSERTION
ALLOWED = YES
RELATION = Employee
METHOD = NEW_TUPLE/CHANGE_TUPLE
CHANGE_NON_SATISFYING_SELECT_COND_ALLOWED = YES
SET project     = available
SET onbbteam    = no
SSET loc          = NY
RELATION = Dept
METHOD = NO_CHANGE
DIALOG PART = REPLACEMENT
ALLOWED = YES
RELATION = Employee
KEY_CHANGE_ALLOWED = YES
METHOD = DELETE_OLD
NEW_TUPLE_ALLOWED = YES
CHANGE_NON_SATISFYING_SELECT_COND_ALLOWED = YES
CHANGE_SATISFYING_SELECT_COND_ALLOWED = YES
SET project     = available
SET onbbteam    = no
SSET loc          = NY
VIEW_NON_KEY_ATTR_CHANGE_ALLOWED?
CHANGE phone        = YES
CHANGE dept         = YES
RELATION = Dept
METHOD = CHANGE_TUPLE
CHANGE_NON_SATISFYING_SELECT_COND_ALLOWED = YES
CHANGE_SATISFYING_SELECT_COND_ALLOWED = YES
VIEW_NON_KEY_ATTR_CHANGE_ALLOWED?
CHANGE manager      = YES
```

`/END OF DIALOG/`

Examples of view updates translated into database updates by the View Update Facility using these TSF excerpts are given in the next section.

8 View Update Facility

The View Update Facility interprets the TSF information to translate view update requests into database requests. These view update requests may be deletions, insertions, and replacements of individual view tuples. To simplify implementation of the prototype, new and old attributes values associated with a view update request are asked as needed while executing the update request; this allows the user to specify (by using '`.`' instead of a value) that the current value should be used for an attribute in an existing tuple.

The prototype View Update Facility has not been interfaced with a database system. All interaction intended with the database instead appears on the user's terminal. This approach facilitates understanding the operation of the program and is a useful debugging aid. It also eliminates the need to preload the database with demonstration or test data. In the examples that follow, such interaction intended for the database is preceded by `(D)`.

8-1 View Update Validation

View updates must be validated by the view update facility. Updating through views imposes constraints, in particular those involving view tuple visibility, that are not required when updating directly through the database.

For deletion, the inclusion dependency on the extension join means that if the root database tuple satisfies the selection clause, the view tuple will appear in the view unless some extension join tuple does not satisfy its part of the selection clause. Verifying that the root database tuple satisfies the selection clause is easily done by reading the tuple before replacing or deleting it. We do not check whether all extension join tuples satisfy their selection clauses, if any; it can be expensive to check whether all extension join tuples satisfy their selection conditions and there is no other reason to read them. (We are adding such checking to the prototype implementation to show how it can be done and to help in deciding whether to do such checking; perhaps the decision to do this checking should be left up to the view definer.) We can be certain, however, that after a successful view delete operation, there is no longer a view tuple with the key of the view tuple deleted.

For insertion, we must ensure that the new view tuple is really new. This means that there was no view tuple whose key matched that of the view tuple being inserted. This implies that there was no database tuple in the root relation with the key of the new view tuple, there was such a database tuple but it did not satisfy the selection condition on the root relation, or some extension join tuple does not satisfy its selection clause. The two cases involving the root relation are easily checked while determining what database updates are to be done to the root. We can easily check the third case, as the relevant tuples are already read while we ensure that the values in the database tuple match the corresponding values in the view. Thus, we are certain that after a successful view insert operation the new view tuple will actually appear in the view.

For replacement, similar considerations apply about the appearance of the relevant view tuples. In this case, we verify that the old view tuple does in fact appear in the view by checking that the database tuples whose attribute values match the corresponding ones for the view do satisfy view selection conditions. As usual, we are certain that after a successful view replacement operation the old view tuple will no longer appear while the new one will.

The lack of complete checking of whether extension join tuples (non-root tuples) satisfy their selection conditions is a simplification made in our prototype implementation that is not inherent in our approach. The effect on deletion is to allow some invalid deletion operations when an extension join tuple does not satisfy its selection condition. We have essentially traded some correctness for some improved performance and efficiency in our prototypes. System designers who adopt our approach to updating views need to evaluate such tradeoffs for their own systems.

8-2 Deletion

We will first illustrate a deletion from the baseball team manager's view. We read the corresponding database root tuple, ensure that it satisfies the selection condition, and then delete it.

User, what view do you want to update?
`Bbteammgr`
User, do you want to delete, insert, or replace?
`DELETE`
User, please enter the view tuple value for the following view attribute(s):
lastname = `Smith`
firstname = `T.`

(D) Database, does a tuple with this key exist in view Bbteammgr (root relation Employee)?
(D) YES
(D) What are the values for these attribute(s)?
(D) Employee.onbbteam = YES
This view tuple will be deleted by replacing the old value of Employee.onbbteam with the new value NO in the Employee relation tuple with the key:
Employee.lastname = Smith
Employee.firstname = T.

The New York manager now deletes an employee from her view.

User, what view do you want to update?
Nypersmgr
User, do you want to delete, insert, or replace?
DELETE
User, please enter the view tuple value for the following view attribute(s):
lastname = Bergman
firstname = P.
(D) Database, does a tuple with this key exist in view Nypersmgr (root relation Employee)?
(D) YES
(D) What are the values for these attribute(s)?
(D) Employee.loc = NY
This view tuple will be deleted by deleting from root relation Employee the tuple with the key:
Employee.lastname = Bergman
Employee.firstname = P.

8-3 Insertion

We insert into the baseball team manager's view by performing a replacement on the database.

User, what view do you want to update?
Bbteammgr
User, do you want to DELETE, INSERT, or REPLACE?
INSERT
(Employee relation)
User, please enter the view tuple value for the following attribute(s):
lastname = Johnson
firstname = J.
(D) For key value(s):
(D) Employee.lastname = Johnson
(D) Employee.firstname = J.
(D) Database, does a tuple with this key exist in relation Employee?
(D) YES
(D) What are the values for these attribute(s)?
(D) Employee.phone = 333-3333
(D) Employee.dept = service
(D) Employee.loc = NY
(D) Employee.project = s33
(D) Employee.onbbteam = NO
User, please enter the view tuple value for the following view attribute(s):
attribute = OLD value → NEW value
phone = 333-3333 → .
dept = service → .
(Dept relation)
(D) For key value(s):
(D) Dept.deptname = service
(D) Database, does a tuple with this key exist in relation Dept?
(D) YES
(D) What are the values for these attribute(s)?
(D) Dept.manager = S.Survik
(D) Dept.division = auto
User, please enter the view tuple value for the following view attribute(s):
attribute = OLD value → NEW value
manager = S.Survik → .

The view tuple is successfully inserted, and the following summary table describes the translation.

```
ATTRNAME   KEY? OLDVALUE   NEWVALUE   VIEWORNOT
***Employee   relation tuple was replaced
lastname   Key  Johnson    Johnson    IN VIEW
firstname  Key  J.         J.         IN VIEW
phone      Non  333-3333   333-3333   IN VIEW
dept       Non  service    service    IN VIEW
loc        Non  NY         NY
project    Non  s33        s33
onbbteam   Non  NO         YES
***Dept       relation tuple was unchanged
deptname   Key  service    service    IN VIEW
manager    Non  S.Survik   S.Survik   IN VIEW
division   Non  auto       auto
```

We insert into the New York manager's view by inserting into the database.

(Employee relation)
User, please enter the view tuple value for the following attribute(s):
lastname = Guerra
firstname = R.
(D) For key value(s):
(D) Employee.lastname = Guerra
(D) Employee.firstname = R.

(D) Database, does a tuple with this key exist in relation Employee?
(D) NO
User, please enter the view tuple value for the following view attribute(s):
attribute = NEW value
phone → **444-4444**
dept → **finance**
(Dept relation)
(D) For key value(s):
(D) Dept.deptname = finance
(D) Database, does a tuple with this key exist in relation Dept?
(D) YES
(D) What are the values for the following attribute(s)?
(D) Dept.manager = P.Ellis
(D) Dept.division = auto
User, please enter the view tuple value for the following view attribute(s):
attribute = OLD value → NEW value
manager = P.Ellis → .

8-4 Replacement

The first example shows how replacement is used by the baseball team manager to replace on employee on the team by another. Notice how the manager elects to use the existing phone number and department name for the new team member, rather than using the values from the old team member.

(Employee relation)
User, please enter the OLD and NEW value for the following attribute(s):
attribute = OLD value → NEW value
lastname = **Talley** → **Chang**
firstname = **B.** → **N.**
(Old view tuple)
(D) For key value(s):
(D) Employee.lastname = Talley
(D) Employee.firstname = B.
(D) Database, does a tuple with this *old* key exist in relation Employee?
(D) YES
(D) What are the values for these attribute(s)?
(D) Employee.phone = 555-5555
(D) Employee.dept = sales
(D) Employee.loc = NY
(D) Employee.project = sal5
(D) Employee.onbbteam = YES
(D) For key value(s):
(D) Dept.deptname = sales
(D) Database, does a tuple with this key exist in relation Dept?
(D) YES
(D) What are the values for the following attribute(s)?
(D) Dept.manager = G.Jacobson
(D) Dept.division = auto
(New view tuple)
(D) For key value(s):
(D) Employee.lastname = Chang
(D) Employee.firstname = N.
(D) Database, does a tuple with this key exist in relation Employee?
(D) YES
(D) What are the values for the following attribute(s)?
(D) Employee.phone = 777-7777
(D) Employee.dept = service
(D) Employee.loc = NY
(D) Employee.project = ser77
(D) Employee.onbbteam = NO
User, please enter the view tuple value for the following view attribute(s):
attribute = OLD value → NEW value
phone = 777-7777 → .
dept = service → .
(Dept relation)
(D) For key value(s):
(D) Dept.deptname = service
(D) Database, does a tuple with this key exist in relation Dept?
(D) YES
(D) What are the values for the following attribute(s)?
(D) Dept.manager = S.Survik
(D) Dept.division = auto
User, please enter the view tuple value for the following view attribute(s):
attribute = OLD value → NEW value
manager = S.Survik → .

The translation of the view tuple replacement on the baseball team view was done according to the following summary table.

```
ATTRNAME   KEY? OLDVALUE   NEWVALUE   VIEWORNOT
***Employee   relation tuple was replaced
lastname   Key  Talley     Talley     IN VIEW
firstname  Key  B.         B.         IN VIEW
phone      Non  555-5555   555-5555   IN VIEW
dept       Non  sales      sales      IN VIEW
loc        Non  NY         NY
project    Non  sal5       sal5
onbbteam   Non  YES        NO
***Employee   relation tuple was replaced
```

```
lastname   Key  Chang      Chang      IN VIEW
firstname  Key  N.         N.         IN VIEW
phone      Non  777-7777   777-7777   IN VIEW
dept       Non  service    service    IN VIEW
loc        Non  NY         NY
project    Non  ser77      ser77
onbbteam   Non  NO         YES
***Dept       relation tuple was unchanged
deptname   Key  service    service    IN VIEW
manager    Non  S.Survik   S.Survik   IN VIEW
division   Non  auto       auto
```

The next update, by the New York manager, transfers an employee from the sales department to the finance department. The replacement can be performed because there already is a finance department tuple. Note also that the manager of the finance department has been changed in the department relation by explicit user request. The dialog allows us to change existing tuples in the department relation, but not insert new ones, through this particular view. The change to the manager of the finance department necessarily affects any other employees in the finance department, and hence their corresponding view tuples. This side effect means that the view tuple replacement request was not implemented exactly, as additional changes were made to other view tuples that are implied by the user's request. We considered this issue in more detail in our earlier work [Keller 85a, 85b, 86a, 86b].

(Employee relation)
User, please enter the OLD and NEW value for the following attribute(s):
attribute = OLD value → NEW value
lastname = `Meyer` → .
firstname = `W.` → .
(Old view tuple)
(D) For key value(s):
(D) Employee.lastname = Meyer
(D) Employee.firstname = W.
(D) Database, does a tuple with this *old* key exist in relation Employee?
(D) `YES`
(D) What are the values for these attribute(s)?
(D) Employee.phone = `666-6666`
(D) Employee.dept = `sales`
(D) Employee.loc = `NY`
(D) Employee.project = `sal66`
(D) Employee.onbbteam = `NO`
(D) For key value(s):
(D) Dept.deptname = sales
(D) Database, does a tuple with this key exist in relation Dept?
(D) `YES`
(D) What are the values for the following attribute(s)?
(D) Dept.manager = `G.Jacobson`
(D) Dept.division = `auto`
User, please enter the view tuple value for the following view attribute(s):
attribute = OLD value → NEW value
phone = 666-6666 → .
dept = sales → `finance`
(Dept relation)
(D) For key value(s):
(D) Dept.deptname = finance
(D) Database, does a tuple with this key exist in relation Dept?
(D) `YES`
(D) What are the values for the following attribute(s)?
(D) Dept.manager = `P.Ellis`
(D) Dept.division = `auto`
User, please enter the view tuple value for the following view attribute(s):
attribute = OLD value → NEW value
manager = P.Ellis → `J.Parks`

The view tuple replacement on the New York manager view was translated according to the following summary table.

```
ATTRNAME   KEY? OLDVALUE   NEWVALUE   VIEWORNOT
***Employee   relation tuple was replaced
lastname   Key  Meyer      Meyer      IN VIEW
firstname  Key  W.         W.         IN VIEW
phone      Non  666-6666   666-6666   IN VIEW
dept       Non  sales      finance    IN VIEW
loc        Non  NY         NY
project    Non  sal66      sal66
onbbteam   Non  NO         NO
***Dept       relation tuple was replaced
deptname   Key  finance    finance    IN VIEW
manager    Non  P.Ellis    J.Parks    IN VIEW
division   Non  auto       auto
```

9 Simplifications and Limitations

Our prototypes used a simplified language based on SQL that facilitated parsing and building data structures for schemas, etc. Rather than interacting with a real database, our prototype View Update Facility printed all database requests on the terminal and read in from the terminal the database values.

We handle a large but somewhat limited class of select, project, and join views. This class can be extended somewhat but at increased complexity. The simplest enlargement of the class of views would include directed acyclic graphs in addition to trees. Disjunctions and non-join comparisons of attributes would complicate the decisions of what attributes to change and to what values to change them. Allowing join attributes to be removed from the view would incur a level of ambiguity that would require the use of heuristics or a clarification dialog at update request time rather than static semantics we use. (For example, see [Davidson 83].)

10 Implementation Statistics

Three undergraduate students wrote the two programs expending 8 man-months of effort during one academic year. The prototype View Definition Facility contains less than 3000 lines of code which includes parsing (about 150 lines), reading relation definitions (about 300 lines), reading view definition (about 400 lines), verifying that view can be updated (about 400 lines), obtaining view update semantics using the dialog (about 850 lines), displaying relation, view, and translator summaries (about 200 lines), and writing the TSF file (about 400 lines). The prototype View Update Facility contains about 3200 lines of code, of which about 1100 lines are used to load in the TSF file containing the relation and view definitions and the view update translator, and about 1500 lines of code are used to translate view updates into database operations (simulated on the terminal).

11 Conclusion

We have implemented a prototype view update translation system. This system consists of two parts: a View Definition Facility run once when a view is defined (probably by the Database Administrator), and a View Update Facility run whenever the user specifies an update in terms of a view.

Our system handles a large class of select, project, and join views. It can translate every single view tuple insertion, deletion, and replacement that satisfies database constraints. However, our approach allows some expressible and correct updates to be rejected for security or other related reasons.

We have shown that it is feasible to update relational databases through views if the ambiguities are resolved at view definition time by obtaining the semantics for choosing a translator. We have developed a theory of view update translation that allows us to enumerate the set of possible translators for a view. By thinking about these translators as organized into a decision tree, we are able to choose the desired translator through a dialog. It is the semantics of the problem and view that affect how the decision tree is traversed and what translator is chosen. We have created an encoding for describing the translator chosen and shown how to use this encoding to translate view update requests into database update requests.

There is a significant difference between a demonstration prototype and integration of the techniques in production commercial systems. However, we believe that our prototype View Update Translation Facility demonstrates the feasibility of updating relational databases through views, so that commercial relational database systems will soon support updating through views.

12 Acknowledgements

This work is based on a dissertation [Keller 85b] researched under the supervision of Gio Wiederhold, Jeff Ullman, and Christos Papadimitriou. Gio Wiederhold suggested developing a prototype implementation and describing it in this paper. The implementation of the prototype View Definition Facility was begun by Moon Ho Chung and Nazir Alimohammad. This part was completed by Laurel Harvey, who also implemented the prototype View Update Facility and assisted in writing this paper. The programs were designed by Arthur Keller, who supervised their implementation after developing the necessary theories and algorithms.

13 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 86a] Arthur M. Keller, "The Role of Semantics in Translating View Updates," *IEEE Computer*, **19**:1, January 1986, pp. 63–73.

[Keller 86b] Arthur M. Keller, "Choosing a View Update Translator by Dialog at View Definition Time," *12th Int. Conf. on Very Large Data Bases*, Kyoto, Japan, August 1986, pp. 467–474.

[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.