To be appeared in the the Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication (ACM IMCOM '17), January 05-07, 2017, Beppu, Japan

An Intention-based Approach to the Updatability of Views in Relational Databases

Yoshifumi Masunaga
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku
Tokyo 112-8610, Japan
+81-3-3943-3151
yoshi.masunaga@gmail.com

ABSTRACT

Views are essential for relational databases in that they could provide a high degree of logical data independence. However, although views are a useful tool for queries, they present significant problems if insertions, deletions, or rewrites are expressed using views. This is due to the fact that views are not base relations stored in a relational database but virtual in the sense of being merely definitions of queries issued to a relational database. While the view update problem has attracted the attention of many scientists and business people, a complete resolution to this problem is still an open issue unfortunately. In this paper, in contrast to the traditional approaches such as the syntax-based approach and the semantics-based approach, an intention-based approach is presented to resolve this problem. By introducing the pro forma guessing of update intention approach, Cartesian product views and join views become updatable while they are not updatable in the traditional sense. This is due to the fact that in certain cases the user's view update intention can be guessed uniquely by checking the extension of each view update transformation candidate, which is calculated using temporarily materialized views. In addition to the above results, the updatability of other basic views such as union views, difference set views, intersection views, projection views, and selection views is re-examined under the intention-based approach. Based on the result, an algorithm is presented to determine whether a given view, defined arbitrarily by using relational algebra operations recursively, is updatable or not. Compared to the traditional approach, it is shown that the intention-based approach realizes richer solution to view updatability problem.

CCS Concepts

Information systems → Data management systems → Database design and models → Relational database model.

Keywords

Relational databases; views; view update problem; updatability of views; join views; Cartesian product views; materialized views; relational algebra; intention-based approach.

1. INTRODUCTION

Relational database views were first introduced by Codd, the inventor of the relational data model, himself in 1974 [1]. He used the characteristics of relational algebra, so that views are defined recursively using query results. As is well known, views could provide shorthand notations for user convenience; relational DBMSs could achieve a high degree of logical data independence by supporting views at the external schema level; and views could provide mechanisms to achieve database security within an authorization framework.

However, since views are not base relations stored in a database, but merely definitions of queries issued to a database, the updatability of views poses a formidable problem, although there is no problem as long as views are subject to query which is due to query modification [2]. We call this problem the view update problem. While the view update problem has attracted the attention of many scientists and business people, and much research and development has been reported since views were first introduced, a complete resolution to this problem is unfortunately, yet to be reached.

Let us summarize the history of the research into, and development of, the view update problem. First, the problem was investigated under the syntax-based approach. That is, a view was defined as a mapping function from a database state to a view state. Suppose that s_τ represents a database state at time τ, V is a view definition, $V(s_\tau)$ represents the view state at that time, and $\boldsymbol{u}$ represents the update operation issued to $V(s_\tau)$; then $\boldsymbol{u}$ is translatable if and only if there exists a translation T of $\boldsymbol{u}$ to an update request to s_τ, such that it has no side effects, and is unique [3]. That is, $\boldsymbol{u}$ is translatable if and only if the commutative diagram holds in Figure 1. In [3] Dayal and Bernstein mentioned that the uniqueness criterion could be controversial; however, since there is no other way to resolve translation ambiguity, they adopted this criterion. Additionally, they imposed no extraneous update condition, which means that the translation should be minimal in the sense that it should not update base relations, unless $\boldsymbol{u}$ otherwise requests. In our approach, which we will show later, this condition is not considered, because it is automatically satisfied by the construction of translation alternatives.

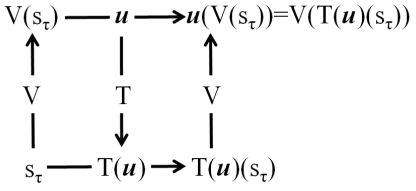

Figure 1. Commutative diagram of representing translatability of view update $\boldsymbol{u}$ at time τ.

However, as research progressed, a semantics-based approach was introduced in order to break through the limitations of the syntax-based approach [6]. Instead of relaxing the uniqueness criterion of the translation T in the syntax-based approach, it was intended to resolve the semantic ambiguities of translation alternatives by manipulating the meanings of views. That is, it was made clear that the view update problem is highly semantical, in the sense that interactions with view updaters are essentially necessary to resolve the semantic ambiguities of view update translations. In fact, the development of a view update tool named TAILOR was reported, which involved user interaction to resolve semantic ambiguity of translation [9]. This approach could be called interaction-based.

In this paper, we introduce yet another approach to resolve the translation ambiguity which is called the intention-based approach. This does not require human interaction as the interaction-based approach involved, but require materializing intermediate views to resolve the translation ambiguity. The pro forma guessing of update intention approach is proposed for this purpose. It will be shown that Cartesian product views and join views become updatable under the intention-based approach, while they are not under the traditional approach, i.e. the syntax-based approach and the semantics-based approach.

View definitions and their meanings will be examined in section 2. View updatability under the traditional approach will be summarized in section 3. Section 4 will outline the updatability of relational database views based on pro forma guessing of update intention. An algorithm to determine whether a generally defined view is updatable or not under the pro forma guessing of update intention approach will be given in section 5. Section 6 will conclude this paper.

2. VIEWS

2.1 View Definitions

According to the first view paper [1], we adopt relational algebra to define views. Seven relational algebra operators: union, difference set, intersection, Cartesian product, projection, selection, and join, are adopted to define views. Division operator is excluded, since it is expressed as a generally defined view using Cartesian product, projection and difference set operators. Strictly speaking, difference set and join operators are also unnecessary, since a set of five operators: union, intersection, Cartesian product, projection, and selection, forms an independent set. But we will include them because they are very commonly and widely used to define views.

[Definition 1] (Views)

1. Base relation R is a view.
2. Suppose V1 and V2 are views which are union-compatible. Then $V1 \cup V2$, $V1 - V2$, and $V1 \cap V2$ are union, difference set, and intersection views, respectively.
3. Suppose V1 and V2 are views. Then $V1 \times V2$ is a Cartesian product view.
4. Suppose V is a view. Then V[X] is a projection view, where X is a set of attributes of V.
5. Suppose V is a view. Then V[Ai θ Aj] is a selection view, where Ai and Aj are θ-comparable.
6. Suppose V1 and V2 are views. Then V1[Ai θ Bj]V2 is a join view, where Ai and Bj are θ-comparable.
7. Only those defined by 1 to 6 are views. ■

2.2 Meanings of Views

In the semantics-based approach, meanings of views play an essential role in the identification of translation alternatives of a view update [6]. First of all, the meaning of a base relation R is defined by $P_R(t)=\{t \mid t \in R\}$, which is the logical formula of tuple relational calculus. For example, the meaning of union view $R \cup S$ is defined by $P_{R \cup S}(t)=\{t \mid t \in R \vee t \in R\}$. The meanings of the seven basic views introduced above are summarized in Table 1. We will show how view update translation rules are derived from the meanings of views in the next section.

Table 1. Meanings of the basic views.

Types of Basic Views	Notations of Views	Meanings of Views
Union	$R \cup S$	$\{t \mid t \in R \vee t \in S\}$
Difference set	$R - S$	$\{t \mid t \in R \wedge \neg (t \in S)\}$
Intersection	$R \cap S$	$\{t \mid t \in R \wedge t \in S\}$
Cartesian product	$R \times S$	$\{(r, s) \mid r \in R \wedge s \in S\}$
Projection	R[X]	$\{u \mid u \in dom(X) \wedge (\exists t \in R)(t[X] = u)\}$
Selection	R[Ai θ Aj]	$\{t \mid t \in R \wedge t[Ai]\ \theta\ t[Aj]\}$
Join	R[Ai θ Bj]S	$\{(t, u) \mid t \in R \wedge u \in S \wedge t[Ai]\ \theta\ u[Bj]\}$

3. VIEW UPDATE TRANSLATION

3.1 Translation Mechanism

As it was introduced by the syntax-based approach, the updatability of views is defined as follows, where updatability is a generic term for deletability, insertability, and rewritability:

[Definition 2] (View Updatability)

Let V be a view definition. Then V is updatable if and only if the commutative diagram of Figure 1 holds for any update request and for any database state at any time. ■

It is important to notice here that view updatability is an issue examined on the database schema level, not on the instance level. In other words, for example, if join views are deletable, then the commutative diagram of Figure 1 should hold for any join view definition, for any delete request against it, and for any database state at any time. Conversely, if there exists a join view which is not deletable under a certain circumstance, then join views are not deletable.

In order to reveal the criteria to provide a correct translation T of Figure 1 under the syntax-based approach, much research and development has been reported taking into account the database integrity conditions such as functional dependencies [3, 4, 5, 7, 8, 10, 11, 12, 18]. However, as we mentioned earlier, the semantics-based approach [6] was introduced in order to break through the limitations of the syntax-based approach. That is, by using the meanings of views, it was revealed that the entire view update translation mechanism was made clear. More precisely, given a view update request ***u*** issued to view definition V, we can identify a unique translation T or a set of translation alternatives depending on the kind of update request ***u*** and view definition V. For example, suppose that deletion request ***d***=delete t_1 from V, where $V=R \cup S$, is issued. Then $\neg P_V(t_1)$ must hold because t_1 lost the meaning of V. Since $P_{R \cup S}(t)=\{t \mid t \in R \vee t \in R\}$, $\neg(t_1 \in R) \wedge \neg(t_1 \in R)$ must hold. This implies that ***d*** must be translated into a

set of two deletion requests which are ***d1***=delete t_1 from R and ***d2***=delete t_1 from S. Notice that, in this case, the translation is unique and no translation ambiguity happens.

3.2 Translation Ambiguity

However, translation ambiguity happens depending on update requests and view definitions.

[Example 1] (Intersection views are not deletable)

Suppose that deletion request ***d***=delete t_1 from V, where $V=R \cap S$, is issued. Then translation alternatives occur. That is, since $P_{R \cap S}(t)=\{t \mid t \in R \wedge t \in R\}$, $\neg(t_1 \in R) \vee \neg(t_1 \in R)$ must hold. This means that ***d*** is realized by either ***d1***=delete t_1 from R or ***d2***=delete t_1 from S or ***d3***=execute both ***d1*** and ***d2***. Since we cannot find any information to resolve this ambiguity from deletion request ***d***, we conclude that intersection views are not deletable. ■

Another interesting example is delete requests to join views.

[Example 2] (Natural join views are not deletable under the traditional approach)

We first show how a delete request against a natural join view is translated into update requests to its base relations under the semantics-based approach: Suppose there are two base relations, R(A, B) and S(B, C), and the natural join view R*S is defined. The meaning of $P_{R*S}(w)$ is defined as follows, where w=(t, v) is a tuple of R*S:

$$\{(t, v) \mid t \in R \wedge v \in \mathrm{dom}(C) \wedge (\exists s \in S)(\, t[B] = s[B] \wedge v = s[C]\,)\}$$

This implies that tuple (a, b, c) is a member of R*S if and only if (a, b) and (b, c) are members of R and S, respectively. According to the semantics-based approach for view updatability, this means that the delete request, ***d***=delete (a, b, c) from R*S, is realized if and only if either ***d*** is translated into delete request, ***d1***=delete (a, b) from R, or delete request, ***d2***=delete (b, c) from S, or delete request, ***d3***=execute both ***d1*** and ***d2***. That is, we have three translation alternatives, T1(***d***), T2(***d***), and T3(***d***), to realize ***d***:

T1(***d***) = delete (a, b) from R

T2(***d***) = delete (b, c) from S

T3(***d***) = execute both T1(***d***) and T2(***d***)

However, this translation ambiguity causes a formidable problem: Suppose that deletion request, ***d1*** = delete (a, b', c) from R*S, whose instances at time τ are shown in Figure 2, is issued. As it is easily checked, deletion of tuple (a, b', c) from R*S, as depicted in Figure 2, is realized without side effects by adopting either T1(***d1***) = delete (a, b') from R, or T2(***d1***) = delete (b', c) from S, or T3(***d1***) = execute both T1(***d1***) and T2(***d1***). However, the essential point of the semantics-based approach is to notice that the meanings of alternatives which bear weight in the real world are completely different to each other. That is, alternative T1(***d1***) should be adopted if and only if an event occurs in the real world that requires deletion of (a, b') from R. The same is true for the other two alternatives. But, in general, it is impossible to identify the reason why the delete request ***d1*** was issued to R*S without human interaction (This is the reason why TAILOR [9] was developed.). Therefore, it was concluded that natural join views are not deletable in the traditional approach. ■

The same happens to Cartesian product views. However, as it will be shown in the next section, join views and Cartesian product views become updatable in certain cases under the intention-based approach.

R

A	B
a	b
a'	b
a	b'

S

B	C
b	c
b	c'
b'	c

R*S

A	B	C
a	b	c
a	b	c'
a'	b	c
a'	b	c'
a	b'	c

Figure 2. An instance of natural join view R*S.

4. VIEW UPDATABILITY BASED ON INTENTION-BASED APPROACH

4.1 Pro Forma Guessing of Update Intention

As it was shown in section 3.2, natural join views are not deletable in the traditional sense. This is due to the translation ambiguity, which seemed unable to be resolved without human interaction. In order to break down the wall of tradition, we carefully re-examined what happened in these cases, and found that, in certain cases, the user's view update intention can be guessed uniquely by checking the "extension" of each view update transformation candidate, which is calculated using temporarily materialized views. We call this novel method of view updating "view updatability based on pro forma guessing of update intention". A typical example will be shown below.

[Example 3] (Natural join views are deletable under the pro forma guessing of update intention approach)

Let us use the same example as in Example 2, i.e. let R(A, B) and S(B, C) be base relations, and R*S be the natural join view definition, whose instances at time τ are shown in Figure 2. Suppose delete request, ***d4*** = delete {(a, b, c), (a, b, c')} from R*S, is issued to the view. According to the traditional semantics-based approach, the following three translation alternatives are implied:

T1(***d4***) = delete (a, b) from R

T2(***d4***) = delete {(b, c), (b, c')} from S

T3(***d4***) = execute both T1(***d4***) and T2(***d4***)

Because of this translation ambiguity, delete requests to natural join views are not accepted in the traditional approach (Example 2). However, if we execute T1(***d4***) temporarily, then we find that the temporarily materialized view of R*S, which we call the "extension" of translation candidate T1(***d4***), is R*S − {(a, b, c), (a, b, c')}, which is exactly the desired update result of deletion ***d4***. If we execute T2(***d4***) temporarily, then we find that the temporarily materialized view is R*S − {(a, b, c), (a, b, c'), (a', b, c), (a', b, c')}, which has a side effect. Also, if we execute T3(***d4***) temporarily, then the same result happens as was held for the T2(***d4***) case. That is, although we had three translation alternatives for the realization of delete request ***d4***, we found that the commutative diagram of Figure 1 held only for T1(***d4***); i.e. the commutativity does not hold for both T2(***d4***) and T3(***d4***) in this case. We interpret this phenomenon in the following way: although there is no way to know the exact intention of the delete request to the natural join view without asking the view updater by human interaction, we guess to conclude that a certain event would have happened in the real world at the time when the delete request was issued which corresponds to T1(***d4***), because two other translation alternatives, T2(***d4***) and T3(***d4***), cannot satisfy the commutativity of Figure 1. In this case we say that the natural join view V*S is deletable with respect to delete request ***d4*** under

the pro forma guessing of update intention approach. Note that the updatability of a given view definition under this approach depends on both the database state and update request issued at that time. ■

It is not always true that every natural join view is deletable under the pro forma guessing of update intention approach. A typical example has already been given in Example 2. However, since certain cases exist where natural join views are deletable under the pro forma guessing of update intention approach, we say, in general, that natural join views are deletable under the pro forma guessing of update intention approach, i.e. the intention-based approach.

4.2 Updatability of Cartesian product Views and Join Views Based on the Intention-based Approach

As stated before, updatability is a generic term for deletability, insertability, and rewritability. In Example 3, we showed that join views are deletable under the intention-based approach. Similarly, we can show that Cartesian product views are deletable under the intention-based approach. So, let us examine the insertability and rewritability of those views.

[Example 4] (Cartesian product views are insertable under the pro forma guessing of update intention approach)

Without loss of generality, suppose that an instance of Cartesian product view R(A, B)×S(C, D) is given, as shown in Figure 3. Suppose that tuple insert request ***i1*** is issued:

i1 = insert (a, b, c, d) into R×S

Then, because of the meaning of the Cartesian product as shown in Table 1, (a, b)∈R ∧(c, d) ∈S should hold to realize this request. This implies a unique view update translation:

T(***i1***) = insert (a, b) into R and insert (c, d) into S

Obviously, this translation causes side effects, and therefore, in the traditional sense, it is concluded that Cartesian product views are not insertable.

Now, let us examine another case: suppose that tuple insert request ***i2*** is issued instead:

i2 = insert {(a3, b3, c1, d1), (a3, b3, c2, d2)} into R×S

Again, because of the meaning of the Cartesian product, ***i2*** should be translated into:

T(***i2***) =insert (a3, b3) into R and insert {(c1, d1), (c2, d2)} into S

If we temporarily update R and S according to T(***i2***), and materialize the view, then we find that the temporarily materialized view, i.e. the extension of T(***i2***), is exactly the intended update result. We interpret this in the following way: we guess that an event would happen in the real world, which corresponds to the insertion of (a3, b3) into R. (Since insert {(c1, d1), (c2, d2)} into S does not cause any change to S, we guess that nothing occurs in the real world with respect to S.) The pro forma guessing of update intention approach accepts this interpretation, and therefore we can say that Cartesian product views are insertable under the pro forma guessing of update intention approach. Note that calculation of the temporarily materialized view works not to eliminate translation ambiguity, as was shown in Example 3, but to ensure the avoidance of side effects by the view update translation. ■

A similar argument holds for join views, i.e. join views are insertable under the pro forma guessing of update intention approach.

R

A	B
a1	b1
a2	b2

S

C	D
c1	d1
c2	d2

R×S

A	B	C	D
a1	b1	c1	d1
a1	b1	c2	d2
a2	b2	c1	d1
a2	b2	c2	d2

Figure 3 . An instance of Cartesian product view R×S.

[Example 5] (Cartesian product views are rewritable under the pro forma guessing of update intention approach if the rewrite request is compatible)

Let us take the same example of an instance of Cartesian product view R(A, B)×S(C, D), as shown in Figure 3. Suppose that a tuple rewrite request ***r1*** is issued to R×S:

r1= rewrite {(a1, b1, c1, d1), (a1, b1, c2, d2)} of R×S to {(a3, b3, c1, d1), (a3, b3, c2, d2)}

We consider that rewrite is a sequence of delete and insert operations: First, the delete request, ***d1*** = delete {(a1, b1, c1, d1), (a1, b1, c2, d2)} from R×S, is realized by translating ***d1*** to T1(***d1***) = delete {(a1, b1)} from R, under the pro forma guessing of update intention approach; as was true for the natural join view shown in Example 3. Next, in order to fulfill request ***r1***, tuple insert request, ***i2***= insert {(a3, b3, c1, d1), (a3, b3, c2, d2)} into R×S, should be realized. As stated before, according to the meaning of the Cartesian product, this request is translated into base relation updates by translation T(***i2***), which was defined in Example 4. Note that insertion of {(c1, d1), (c2, d2)} into S does not affect S at all, because these two tuples exist in S. But, notice that both the target of delete request ***d1*** and the target of insert request ***i2*** are the same, i.e. R, in this case. In such a case, we say that translations T1(***d1***) and T(***i2***) are compatible with respect to rewrite request ***r1*** or simply that rewrite request ***r1*** is compatible.

If the compatibility does not hold, then the rewritability does not hold. For example, suppose that tuple rewrite request ***r2*** is issued to R×S:

r2 = rewrite {(a1, b1, c1, d1), (a1, b1, c2, d2)} of R×S to {(a1, b1, c3, d3), (a1, b1, c4, d4)}

Then, under the same delete request ***d1*** and translation T1(***d1***), R×S is deletable with respect to ***d1*** under the pro forma guessing of update intention approach. However, to realize ***r2***, it is necessary to execute a tuple insert request, ***i3*** = insert {(a1, b1, c3, d3), (a1, b1, c4, d4)} into R×S, after deletion ***d1*** is executed. This is translated by translation T(***i3***) = insert (a1, b1) into R and insert {(c3, d3), (c4, d4)} into S. However, in this case, if we calculate a temporarily materialized view of R×S, then we will see that a side effect occurs, i.e. the extension of T(***i3***) is R×S ∪ {(a1, b1, c3, d3), (a1, b1, c4, d4), (a2, b2, c3, d3), (a2, b2, c4, d4)}. This phenomenon occurred because rewrite request ***r2*** is not compatible. Therefore, we can say that Cartesian product views are rewritable under pro forma guessing of update intention approach if rewrite requests are compatible. ■

Analogous investigation reveals that, join views are rewritable under the pro forma guessing of update intention approach if rewrite requests are compatible.

4.3 Updatability of Seven Basic Views

In this section, we will summarize the updatability of seven basic views. As stated in the previous section, the updatability of Cartesian product views and join views was improved under the pro forma guessing of update intention approach, i.e. the intention-based approach.

However, pro forma guessing of update intention does not work to resolve the translation ambiguity for other cases, i.e. union, difference set, intersection, projection, and selection views. This is because view materialization does not cause any difference among translation alternatives. Table 2 shows the updatability of seven basic views under the intention-based approach and the traditional approach. In the next section, we will investigate the updatability of generally defined views using this table.

Table 2. List of the updatability of seven basic views under the intention-based approach and the traditional approach.

Types of Basic Views	View Update Operations	Updatability	
		Intention-based Approach	Traditional Approach
Union	Delete	○	○
	Insert	×	×
	Rewrite	×	×
Difference set	Delete	×	×
	Insert	○	○
	Rewrite	×	×
Intersection	Delete	×	×
	Insert	○	○
	Rewrite	×	×
Cartesian product	Delete	○	×
	Insert	○	×
	Rewrite	○ *1	×
Projection	Delete	○	○
	Insert	○ *2	○ *2
	Rewrite	×	×
Selection	Delete	×	×
	Insert	○	○
	Rewrite	×	×
Join	Delete	○	×
	Insert	○	×
	Rewrite	○ *1	×

○: Updatable. : ×: Not updatable,

*1: Request is compatible, *2: Primary key included.

5. UPDATABILITY OF GENERALLY DEFINED VIEWS

5.1 View Definition Trees and Their Data Structure

As per Definition 1, in general, a view is defined recursively using base relations and predefined views. Note first that parentheses are used when a view is defined. For example, a union view of R and S is denoted by (R ∪ S) instead of R ∪ S, except for the case where a base relation itself is defined as a view. Obviously, by virtue of parentheses, we can parse a view definition uniquely, so that we can easily obtain a view definition tree or a view parsing tree. Since the seven basic view-defining operators are either unary or binary operators (projection and selection operators are unary and the others are binary), a view definition tree forms a binary tree whose root is a generally defined view; its intermediate nodes represent intermediate views, and its leaves represent base relations which are used to define view. Because of the existence of difference set, Cartesian product, and join operators, view definition trees are ordered trees. Therefore, a breadth-first search is possible for view definition trees.

Now, in order for a decision algorithm of view updatability to be programmable, let us define a data structure of a view definition tree. Data structure for a node is shown in Figure 4. A node is denoted by NODE. It consists of four fields: VNAME, VDEF, LLINK, and RLINK. The node name field VNAME specifies identifiers such as the targeted view name, intermediate view name, or base relation name. The view-defining operation field VDEF concretely specifies view definition operations used to define the node in terms of its children, i.e. intermediate views or bases relations specified by the left link field LLINK and the right link field RLINK of that node, where the RLINK value is set to NULL if the view definition operation is unary. Note that nodes of a view definition tree are indexed according to the breadth-first search order, where the root is indexed "0" (zero).

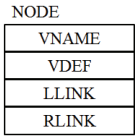

Figure 4 . Data structure of a node of view definition tree.

Now, let us examine a sample view named SE@Tokyo, which is defined as follows:

[Example 6] (View SE@Tokyo and its view definition tree)

Suppose there are two base relations, EMP(ENO, ENAME, DNO, JOB) and DEPT(DNO, DNAME, LOC) ; then a view named SE@Tokyo is defined as follows:

SE@Tokyo = (((EMP[JOB = 'SE']) [EMP.DNO = DEPT.DNO] (DEPT[LOC = 'Tokyo'])) [ENO, ENAME, EMP.DNO])

Obviously, this view lists the employee number (ENO), employee name (ENAME), and department number (DNO) of all employees whose job is SE and whose department location is Tokyo. Although set operators such as union, difference set, and intersection operators are not used, SE@Tokyo seems quite a common view because it uses selection, equi-join, and projection

operators. Because we required to use parentheses in defining views, SE@Tokyo is unambiguously parsed so that its view definition tree is obtained straightforwardly as shown in Figure 5. Here, shorthand notations appearing in the VNAME fields represent intermediate views: IntView1=((EMP[JOB='SE'])[EMP.DNO=DEPT.DNO] DEPT[LOC='Tokyo'])), IntView2=(EMP[JOB='SE']), and IntView3 = (DEPT[LOC='Tokyo']). ■

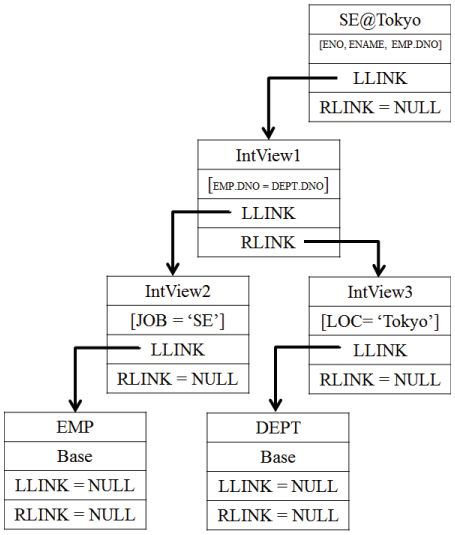

Figure 5 . Data structure of a binary tree representing view SE@Tokyo.

5.2 Decision Algorithm of View Updatability Based on Pro Forma Guessing of Update Intention

Let us show an algorithm to decide whether a generally defined view is updatable or not with respect to an update request issued to it under the pro forma guessing of update intention approach.

[*Algorithm D*] (Decision Algorithm of View Updatability Based on Pro Forma Guessing of Update Intention)

Given a view definition V and an update request ***u*** against V, this algorithm decides whether V is updatable or not with respect to ***u*** under the pro forma guessing of update intention approach. The third column of Table 2 is looked up to know the view updatability for the seven basic views. The pointer variable P is used to refer to a node of the view definition tree. The root of the tree is indexed "0", and the other nodes are indexed according to the breadth-first search order. The nodes of the tree are assumed to contain at least the following fields in addition to VNAME(P) = view name stored in NODE(P); VDEF(P) = view definition used to define NODE(P) in terms of its children, i.e. the roots of the subtrees pointed to by LLINK(P) and RLINK(P); LLINK(P) = pointer to left subtree of NODE(P); and RLINK(P) = pointer to right subtree of NODE(P):

UREQ (P) = update request to NODE(P);

UREL(P) = update relevance of NODE(P);

ICON(P) = integrity constraint of NODE(P).

D1. [Initialize] Construct a view definition tree of V. Set P ← 0 (zero), set UREQ(0) = ***u***, and set UREL(P) = 1 (1 means "relevant" and 0 means "irrelevant") for every P from 0 to N_{max}. (The pointer variable P will move down the tree according to the breadth-first search order. Nodes of the tree will be indexed 0, 1, 2, …, N_{max} ($N_{max} \geqq 0$), according to this order.) Set ICON(P) = NULL for every P, and VDEF(P) = 'Base' for every P indexed to leaves of the tree.

D2. [Detection of termination] If $P = N_{max}$, go to D7. Otherwise, if UREL(P) = 1, go to D3. If UREL(P) = 0, P ← P+1 and goes back to D2.

D3. [Table lookup] Look up the third column of Table 2, and check the updatability of the VDEF(P) – UREQ(P) pair. If the entry is "○", set ICON(P) and modify the update request UREQ(P) using ICON(P), and go to D4 or D5, respectively. Go to D6 otherwise.

D4. [Translate view update in the traditional sense] In this case, UREQ(P) will be translated into update requests against one or both of its children, i.e. the roots of the subtrees pointed to by LLINK(P) and RLINK(P). If the translation result of UREQ(P) is one sided, for example on the left side, set UREQ(LLINK(P)) accordingly, and set UREL(P) = 0 for every P indexed to any node of the subtree pointed to by RLINK(P). Set analogously for the right case. If two-sided, set UREQ(LLINK(P)) and UREQ(RLINK(P)) accordingly. Note that if VDEF(P) is a unary operation, translation is always on the left side. Set P ← P+1 and go back to D2.

D5. [Translate view update based on pro forma guessing of update intention] In this case, updatability of UREQ(P) depends on the database state at the time of update issued. Temporal updates are necessary. If it is updatable, take steps similar to D4. Go to D6 otherwise.

D6. [Not updatable] V is not updatable with respect to u.

D7. [Updatable] V is updatable with respect to u. The updates to base relations are calculated accordingly, by composing relevant update translations. ■

5.3 Explanation of the Movement of the Decision Algorithm of View Updatability

Let us explain how *Algorithm D*, the decision algorithm of view updatability based on pro forma guessing of update intention, works, by taking an example. To do this, we will use the view introduced in Example 6, SE@Tokyo. We assume that a delete request, ***d*** = delete {(003, J. Smith, K41)} from SE@Tokyo, is issued, where instances of relations EMP and DEPT are as depicted in Figure 6. (Values SE and NE represent system engineer and network engineer, respectively.)

Now, let us explain how *Algorithm D* works step by step:

[Example 7] (A typical movement of *Algorithm D*)

1. (Step D1) Given a view definition of SE@Tokyo, construct its view definition tree as depicted in Figure 5. According to the breadth-first search order numbering, nodes are indexed. Set UREL(P) = 1 and ICON(P) = NULL for every P. Set P = 0 and UREQ(0) = ***d***; go to step D2.
2. (Step D2) Since $P = 0 \neq N_{max}$ (= 5) and UREL(0) = 1, go to step D3.
3. (Step D3) By VDEF(0), NODE(0) = SE@Tokyo is a projection view of IntView1 on [ENO, NAME, EMP.DNO]. Set ICON(0) = (∀ s ∈ SE@Tokyo)(∃ t ∈ IntView1)(s = t[ENO, ENAME, EMP.DNO]). Since the VDEF(0) –

UREQ(0) pair actually comprises a projection view and a delete request, by table lookup, go to step D4.

4. (Step D4) UREQ(0) is translated into delete request UREQ(LLINK(0)) = UREQ(1) = ***d1*** = delete {(003, J. Smith, K41, *, *, *, *)} from IntView1; where asterisk (*) represents any value or NULL. Set P ← P+1 (therefore P = 1), and go back to step D2.
5. (Step D2) Since P = 1 ≠ N_{max} and UREL(1) = 1, go to step D3.
6. (Step D3) By VDEF(1), NODE(1) = IntView1 is a join view of IntView2 (= NODE(2)) and IntView3 (= NODE(3)) on [EMP.DNO = DEPT.DNO]. Set ICON(1) = (∀ s ∈ IntView1)(∃t2∈IntView2)(∃t3∈IntView3) (s = (t1, t2)∧ t1[DNO] = t2[DNO]). Using ICON(1), compensate UREQ(1) so that it is modified to: UREQ(1) = ***d1'*** = delete {(003, J. Smith, K41, *, K41, *, *)} from IntView1. Since the VDEF(1) – UREQ(1) pair actually composes an equi-join view and a delete request, by table lookup, we realize that the updatability of this pair will be decided based on pro forma guessing of update intention. Go to step D5.
7. (Step D5) In order to check the translatability of UREQ(1) under the pro forma guessing of update intention approach, IntView2 and IntView3 are temporarily materialized, and three translation alternatives T1, T2, and T3 of UREQ(1) = ***d1'***, are temporarily executed:

   T1(***d1'***) = delete {(003, J. Smith, K41, *)} from IntView2;

   T2(***d1'***) = delete {(K41, *, *)} from IntView3;

   T3(***d1'***) = execute both T1(***d1'***) and T2(***d1'***).

   Figure 7 shows the temporarily materialized intermediate views IntView2 and IntView3.

   In order to check whether one of the three translation alternatives, T1(***d1'***), T2(***d1'***), and T3(***d1'***), is accepted or not as a unique translation of UREQ(1) under the pro forma guessing of update intention approach, we also need to temporarily materialize IntView1, to which the delete request UREQ(1) is issued. Figure 8 shows the temporarily materialized intermediate view IntView1.

   By temporal execution of UREQ(1) against IntView1, we know that tuple (003, J. Smith, K41, SE, K41, AI, Tokyo) will be deleted. Now, we will examine what happens when alternatives T1(***d1'***), T2(***d1'***), and T3(***d1'***) are temporarily executed. If T1(***d1'***) is temporarily executed, we see that the extension of T1(***d1'***) is exactly equal to the result of the temporal execution of UREQ(1) against IntView1. In contrast, if T2(***d1'***) is executed, then we see that, in addition to tuple (003, J. Smith, K41, SE, K41, AI, Tokyo), tuple (006, R. Jones, K41, SE, K41, AI, Tokyo) is also deleted, which is a side effect. The same argument holds for T3(***d1'***). Since we have succeeded in identifying a unique translation T1(***d1'***) which satisfies the commutativity of Fig. 1, we conclude that the VDEF(1) is deletable with respect to UREQ(1) under the pro forma guessing of update intention approach. Set UREQ(2) = T1(***d1'***) = delete {(003, J. Smith, K41, *)} from IntView2; Since the update translation is one sided, set UREL(P) = 0 for every P indexed to any node of the subtree pointed by RLINK(1). (Therefore, UREL(3) = UREL(5) = 0.) Set P ← P+1 (therefore, P = 2), and go back to step D2.
8. (Step D2) Since P = 2 ≠ N_{max} and UREL(2) = 1, go to step D3.
9. (Step D3) By VDEF(2), NODE(2) = IntView2 is a selection view of EMP, where JOB = 'SE'. Set ICON(2) = (∀ s ∈ IntView2)(∃t∈EMP)(s = t ∧ t[Job] = 'SE'). Using ICON(2), compensate UREQ(2) so that it is modified to UREQ(2) = delete (003, J. Smith, K41, SE) from IntView2. Since the VDEF(2) – UREQ(2) pair actually composes a selection view and a delete request, by table lookup, go to step D4.
10. (Step D4) UREQ(2) is translated into delete request UREQ(LLINK(2)) = UREQ(4) = delete (003, J. Smith, K41, SE) from EMP. Set P ← P+1 (therefore P = 3), and go back to step D2.
11. (Step D2) Since P = 3 ≠ N_{max}, but UREL(3) = 0, set P ← P+1 (therefore P = 4), and go back to step D2.
12. (Step D2) Since P = 4 ≠ N_{max} and UREL(4) = 1, go to step D3.
13. (Step D3) Look up the third column of Table 2, and check the updatability of the VDEF(4) – UREQ(4) pair, which is actually a base relation and a delete request. Since base relations are updatable, UREQ(4) constitutes a member of the translation of UREQ(0) against base relations. Set P ← P+1 (therefore P = 5), and go back to step D2.
14. (Step D2) Since P = 5 = N_{max}, go to D7.
15. (Step D7) SE@Tokyo is updatable with respect to ***u***. UREQ(4) is the update to base relations which realizes the intended update request.

Actually, by executing UREQ(4), tuple (003, J. Smith, K41) is successfully deleted from SE@Tokyo.

EMP

ENO	ENAME	DNO	JOB
001	M. Johnson	K41	NE
003	J. Smith	K41	SE
006	R. Jones	K41	SE
007	J. Bond	K55	SE
010	P. Brown	K55	NE
013	D. Togo	K81	SE

DEPT

DNO	DNAME	LOC
K41	AI	Tokyo
K55	DB	Tokyo
K81	PL	Yokohama

Figure 6 . Instances of base relations EMP and DEPT.

IntView2

ENO	ENAME	DNO	JOB
003	J. Smith	K41	SE
006	R. Jones	K41	SE
007	J. Bond	K55	SE
013	D. Togo	K81	SE

IntView3

DNO	DNAME	LOC
K41	AI	Tokyo
K55	DB	Tokyo

Figure 7 . Instances of temporarily materialized intermediate views IntView2 and IntView3.

IntView2

ENO	ENAME	EMP.DNO	JOB	DEPT.DNO	DNAME	LOC
003	J. Smith	K41	SE	K41	AI	Tokyo
006	R. Jones	K41	SE	K41	AI	Tokyo
007	J. Bond	K55	SE	K55	DB	Tokyo

Figure 8 . Instance of temporarily materialized intermediate view IntView1.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated the view update problem, an old and new problem in the relational database theory. In order to break through the updatability of views, an intention-based approach called the pro forma guessing of update intention approach was introduced. This approach differs from the traditional approach such as the syntax-based approach and the semantics-based approach in the sense that, in certain cases, the user's view update intention can be guessed uniquely by checking the extension of each view update transformation candidate, which is calculated using temporarily materialized views. Under the intention-based approach, join views and Cartesian product views became updatable in certain cases although they are not updatable in the traditional approach. Since those views are the most commonly used views, the results presented in this paper seem significant.

In order to determine whether a given view is updatable or not under the intention-based approach, we re-examined the updatability of the seven basic views; union views, difference set views, intersection views, Cartesian product views, projection views, selection views, and join views. Based on the result, an algorithm is presented to determine whether a given view, defined arbitrarily by using the seven basic view-defining operators recursively, is updatable or not. An example is also given to deepen a concrete understanding of the algorithm. Consequently, a unified solution is applied to the view update problem which, until now, had not been thoroughly resolved.

Future work includes the application of the intention-based approach to the field of SQL. In order to do that, we will need to investigate at least two issues: first, since SQL is not set-based, but bag-based or multiset-based, the pro forma guessing of update intention approach should be modified, so that it can cover the bag-based logic. Second, we should elaborate on how to implement the pro forma guessing of update intention approach on a relational DBMS. The key idea is to use the INSTEAD OF trigger, which was standardized in SQL:2008 [16]. Although it is expected that the updatability of SQL views will be massively increased by using the INSTEAD OF trigger, the best way of using it is not clear. This is because there is not international standard of how best to use it, as well as a lack of theory regarding this matter. We believe that the pro forma guessing of update intention approach presented in this paper will contribute to the development of a mechanism for the realization of an extremely high level of SQL view updatability.

7. ACKNOWLEDGMENTS

The author thanks Ishii, T. and Nagata, Y. (SRA OSS, Inc., Japan) for their discussion. This work is partly supported by the Grant-in-Aid for Scientific Research (KAKENHI) (C) (16K00152) of the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan.

8. REFERENCES

[1] Codd, E. F. 1974. Recent investigations in a relational database system, *Information Processing 74* (1974), North-Holland, 1017-1021.

[2] Stonebraker, M. 1975. Implementation of integrity constraints and views by query modification, In *Proceedings of the 1975 ACM SIGMOD international conference on Management of data* (1975), ACM, New York, NY, 65-78.

[3] Dayal, D. and Bernstein, P. 1978. On the updatability of relational views, In *Proc. 4th VLDB* (1978), 368-377.

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

[5] Cosmadakis, S. and C. Papadimitriou, C. 1983. Updates of relational views, In *Proc. ACM PODS* (1983), 317-331.

[6] Masunaga, Y. 1984. A relational database view update translation mechanism, In *Proc. 10th VLDB* (1984), 309-320.

[7] Keller, A. 1985. Algorithms for translating view updates to database updates for views involving selections, projections, and joins, In *ACM PODS* (1985), 154-163.

[8] Keller, A. 1986. Choosing a View Update Translator by Dialog at View Definition Time, In *Proc. 12th VLDB* (1986), 467-474.

[9] Sheth,A., Larson, J. and Watkins, E. 1988. TAILOR, A Tool for Updating Views, *LNCS*, Vol. 303, Springer, 190-213.

[10] Gottlob, G., Paolini, P., and Zicari, R. 1988. Properties and update semantics of consistent views, *ACM TODS* 13, 4 (1988), 486-524.

[11] Langerak, R. 1990. View updates in relational databases with an independent scheme, *ACM TODS* 15, 1 (1990), 40-66.

[12] Chen, I.-M., Hull, H. and McLeod, D. 1995. An execution model for limited ambiguity rules and its application to derived data update, *ACM TODS* 20, 4, 365-413.

[13] ANSI/ISO/IEC International Standard (IS), Database Language SQL—Part 2: Foundation (SQL/Foundation), «Part 2», September 1999.

[14] Melton, J. and Simon, A. 2002. *SQL:1999 Understanding Relational Language Components*, Morgan Kaufmann.

[15] Nathan Foster, J., Greenwald, M. B., Moore, J. T., Pierce, B. C., and Schmitt, A. 2005. Combinators for bi-directional tree transformations: A linguistic approach to the view update problem, In *Proc. ACM POPL'05* (January, 2005).

[16] ISO/IEC 9075-2:2008 Information technology -- Database languages -- SQL -- Part 2: Foundation (SQL/Foundation), http://www.iso.org/iso/catalogue_detail.htm?csnumber=38640.

[17] PostgreSQL 9.3.9 Documentation, Chapter 34. The Information Schema, 34.63. Views, http://www.postgresql.org/docs/9.3/static/infoschema-views.html.

[18] Date, C. J. 2013. *View Updating and Relational Theory: Solving the View Update Problem*, O'Reilly.