ELSEVIER

Data & Knowledge Engineering 19 (1996) 135–169

DATA & KNOWLEDGE ENGINEERING

View update in entity-relationship approach

Tok Wang Ling*, Mong Li Lee

Department of Information Systems & Computer Science, National University of Singapore, 10 Kent Ridge Crescent, Singapore 0511, Singapore

Received 20 May 1994; revised 18 August 1995; accepted 20 November 1995

Abstract

The traditional problem of updating relational databases through views is an important practical problem that has attracted much interest. In this paper, we examine the problem of view update in Entity-Relationship based database management systems [1] where the conceptual schema is represented by a normal form ER diagram [2] and views may be modelled by ER diagrams. We develop a theory within the framework of the ER approach that characterizes the conditions under which there exist mappings from view updates into updates on the conceptual schema. Concepts such as virtual updates and three types of insertability are introduced. We also present two algorithms, the View Updatability Algorithm and the View Update Translation Algorithm.

Keywords: Entity-Relationship based database management systems; View Updatability; View Update Translation

1. Introduction

Views are external schemas. They increase the flexibility of a database by allowing multiple users to see the data in different ways. They offer a measure of protection by letting users have access to only part of the data and preventing the users from accessing data outside their view. They provide logical independence by allowing some changes to be made to the conceptual schema without affecting the application programs. Views also simplify the user interface by allowing the user to ignore data that are of no interest to him.

For a view to be useful, users must be able to apply retrieval and update operations to it. These operations on the view must be translated into the corresponding operations on the conceptual schema instances. Ling and Lee [3] describe how we can automatically generate the external-to-conceptual mapping and the conceptual-to-internal mapping of an ER based DBMS. Using this mapping, retrievals from a view can always be mapped into equivalent retrievals from the conceptual schema.

A mapping is also required to translate view updates into the corresponding updates on the

* Corresponding author. E-mail: {lingtw,leeml}@iscs.nus.sg

SSDI: 0169-023X(95)00042-9

conceptual schema. However, such a mapping does not always exist, and even when it does exist, it may not be unique [4]. In this paper, we examine the problem of view update in Entity-Relationship based database management systems [1] where views may be modelled by ER diagrams. Section 2 gives a survey of related research works. Section 3 gives the terminologies used in this paper. Section 4 explains what is meant by view updatability in ER approach. We present a theory within the framework of the ER approach that characterizes the conditions under which there exist mappings from view updates into updates on the conceptual schema in Section 5. Sections 6 and 7 describe the *View Updatability Algorithm* and the *View Update Translation Algorithm*, respectively. The View Updatability Algorithm determines if a view is insertable, deletable or modifiable. The View Update Translation Algorithm translates a valid view update into the corresponding database update to be executed. Section 8 describes view implementations in current DBMS systems.

2. A survey of related research works

The problem of updating relational databases through views in an important practical problem that has attracted much interest [5–19]. The user specifies queries to be executed against the database view; these queries are translated to queries against the underlying database through query modification [20]. One of the problems in updating through views lies in determining whether a given view modification can be correctly translated by the system. To define an updatable view, a view designer must be aware of how an update request in the view will be mapped into updates of the underlying relations. Moreover, because of side effects, the view designer must also be made aware of the effects of underlying updates back into the view. In current practice, updates must be specified against the underlying database rather than against the view. This is because the problem of updating through views is inherently ambiguous [14]. How this ambiguity is handled is an important characteristic that differentiates various approaches to supporting view updates. Yet, none has been able to handle the view update problem satisfactorily.

There are two approaches to the problem of mapping view updates. One approach is to regard the conceptual schema and view as abstract data types [13]; the view definition not only describes how view data are derived from the conceptual schema instances, but also how operations on the view are mapped into (that is, implemented using) operations on the conceptual schema [18, 19]. This approach is dependent on the database designer to design views and their operational mappings and to verify that the design is correct. That is, that the conceptual schema operations indeed perform the desired view operations "correctly".

The second approach is to define general translation procedures [6, 8, 11, 14, 16, 20]. These procedures input a view definition, a view update and the current schema instances. They produce, if possible, a translation of the view update into updates on the conceptual schema satisfying some desired properties. [11] develops a theory within the framework of the relational model that characterizes precisely the conditions under which there exist mappings from view updates into updates on the conceptual schema satisfying various properties. He formalizes the notion of update translation and derive conditions under which translation procedures will produce correct translations of view updates. However, the problem of

choosing among several alternative update sequences that might be available for performing a desired relational view update still exists. Our approach to view update in the ER approach eliminates this problem.

Langerak [16] gives algorithms for the translation of a single view tuple insertion, deletion or modification. The single view tuple update is translated into a resulting series of database updates. However, Langerak's approach allows extra tuples entering the view as the result of translating a view tuple insertion or modification. [11] classifies these extra unwanted tuples as anomalies caused by permitting a certain class of view updates. Our approach eliminates these anomalies by characterizing the conditions under which such anomalies will occur. The view updates that will cause these anomalies will not be allowed.

Legg and McDonell [17] present a method for performing arbitrary updates on relational views whose definitions form acyclic hierarchies. The desired state of the view after the update is used to determine the appropriate states for the base relations after the update. The actual base relation updates are derived from these final states. Update translation graphs, which show the dependencies between the views and the base relations, are constructed for this purpose. During translation, ambiguities in the update operations are retained, in the hope that identification of interactions in the complete update translation graphs will allow maximal ambiguity resolution. The generated base relation updates will be applied only if certain correctness criteria are satisfied. The extractness criterion requires that the generated updates do not cause changes in the updated view other than those requested by the user. The uniqueness criterion requires that there be only one generated set of updates that could produce the desired change in the updated view. The shortcoming of this method is that an update translation graph needs to be constructed dynamically for every update, and the cost is likely to be quite high.

Chan [8] describes a method in which translation templates for each view type (relational algebra operator) have been defined [9]. For a particular view update, these templates may be combined to produce a translation tree. Ambiguity is expressed by OR nodes in the tree that represent alternative translations of the update operation, and some ambiguity resolution is achieved by evaluating integrity assertions on the arcs connecting the nodes in an attempt to prune the tree and remove all OR nodes.

Keller [14] analyses the possible translations of particular classes of update operations for relational views. The considered updates are insertions, deletions and replacements. Keller gives five criteria that all candidate update translations must satisfy, which include no database side effects, only one step changes, no unnecessary changes, replacements cannot be simplified and no delete-insert pairs. The satisfaction of these criteria implies restrictions on the view definition function and on the form of view-update expressions. Keller's method [15] resolve semantic ambiguity as it arises. This is achieved by declaring for each view at view definition time, the additional semantics for translating any updates against the view into prescribed unambiguous updates on the operands of the view. These declarations define an update policy for each view; this is more precise technique for defining ambiguity resolution policies than Chan's. The choice of a translator is obtained semi-automatically by a program which conducts a structured dialog with the database administrator. Using the specified translator, user-specified view updates can be translated into database updates without the need for any disambiguiting dialog. However, Keller allows the update policy of translating a deletion or

insertion against a selection view into a modification of the operand view or base relation. For example, consider the relation EMP which contains each employee's number, name, location, and whether the employee is a member of the company baseball team. Given the following view definition,

```
Select*
From EMP
Where Baseball = 'Yes'
```

Keller [15] propose that the request to delete an employee from the view should be translated into a modification of the Baseball attribute value to 'No'. However, complication arises when the domain of the selection attribute has more than two values or the selection condition is a conjunction of terms.

On the other hand, there is very little research on the problem of view update in the ER approach despite its popularity as a conceptual database model and graphical query language. An attempt to investigate view definition and view updates for an extended ER model is done by Czejdo et al. [10]. Czejdo et al. give an informal discussion of the view update problem in the context of graphical query languages for extended ER models. Views are defined by invoking diagram-manipulation operators to transform an original schema into a view schema. The diagram-manipulation operators have an object-traceability property so that the identity of any entity object in any view can be traced to a unique entity object in the underlying database, and the identity of any relationship object in any view can be traced either to a unique relationship object or to a unique set of component relationship objects in the underlying database. Although Czejdo et al. and we, both recognize the fundamental importance of establishing a one-to-one correspondence between view objects and underlying objects, we differ, however, in the way in which we define views and the way in which we recognize this one-to-one correspondence. Moreover, while Czejdo et al. give examples of how various view updates can be translated into updates on the conceptual schema, we give a more formal treatment of the view update problem in the ER approach.

3. Terminologies

Chen [21] proposes the ER approach for database schema design. It uses the concepts of **entity type** and **relationship set** and incorporates some of the important semantic information about the real world. An entity type or relationship set has **attributes** which represents its structural properties. An attribute can be **single-valued, multivalued** or **composite**. A minimal set of attributes K of an entity type E which defines a one-to-one mapping from E into the Cartesian product of the associated value sets of K is called a **key** of E. An entity type may have more than one key and we designate one of them as the **primary key** or the **identifier** of the entity type. Let K be a set of identifiers of some entity types participating in a relationship set R. K is called a key of the relationship set R if there is a one-to-one mapping from R into the Cartesian product of the associated value sets of K and no proper subset of K has such property. One of the keys of a relationship set is designated as the primary key or identifier of

the relationship set. If the existence of an entity in one entity type depends upon the existence of a specific entity in another entity type, then such a relationship set and entity type are called **existence dependent relationship set** and **weak entity type.** A special case of existence dependent relationship occurs if the entities in an entity type cannot be identified by the values of their own attributes, but has to be identified by their relationship with other entities. Such a relationship set is called **identifier dependent relationship set**. A relationship set which involves weak entity type(s) is called a **weak relationship set**. Weak entity types can sometimes be represented as composite, multivalued attributes. One criterion we use is to choose the weak entity type representation if the weak entity type has many attributes or participates independently in other relationship sets. An entity type which is not a weak entity type is called a **regular entity type.** In the ER approach, **recursive relationship sets** and special relationship sets such as **ISA, UNION, INTERSECT**, etc. are allowed. A relationship set which is not weak or special is called a **regular relationship set** (for more details, see [2]).

A schema is usually shared by many applications. A user may be interested in only a subset of the objects, that is, a view of the schema or the external schema. Using the ER approach in a systematic way, we can construct ER based external views. An entity type in an ER external view is called an **external** or **view entity type.** A view entity type is a projection of some entity type, called the **base entity type**, in the conceptual schema. Basically, there is one-to-one correspondence between the entities of a view entity type and the entities of its base entity type. Note that a view entity type may have more than one base entity types. Such a situation arises when these base entity types are connected by one-to-one relationship sets. A relationship set in an ER external view is called an **external** or **view relationship set.** Unlike the view entity type, the relationships of a view relationship set may not have a one-to-one correspondence with the relationships of any relationship set in its corresponding conceptual schema. A view relationship set can be derived by applying some join, project and/or selection operations on one or more relationship sets and special relationships such as ISA, UNION, INTERSECT, etc. We define a **derivation** as a list of conceptual schema relationship sets which are involved in joins to obtain a view relationship set. We observe that if a view relationship set R is functionally equivalent to some conceptual schema relationship set R_i w.r.t. a derivation $\langle R_1, R_2, \ldots, R_n \rangle$ where $i \in \{1, 2, \ldots, n\}$, then we have a one-to-one correspondence between the relationships of R and the relationships of R_i. Otherwise, we do not have a one-to-one correspondence between R and R_i. Note that to construct a view relationship set from a recursive relationship set, the rolenames of the participating entity type of the recursive relationship set must be used [1].

An attribute in a view is called an **external** or **view attribute**. A view entity type may include some or all the attributes of its base entity type. A view entity type may also include attributes from an entity type which is connected to its base entity type by one or more relationship sets in the conceptual schema. We use the concept of derivation to specify the list of conceptual schema relationship sets which are involved in joins to obtain a view attribute. If a view attribute A has a derivation $\langle R_1, R_2, \ldots, R_n \rangle$, where R_i is a regular or special relationship set in the conceptual schema, $1 <= i <= n$, then we call A a **derived attribute**. It can be easily shown that derived attributes cannot participate in the keys of view entity types. The base attribute of A can be in R_n or in some participating entity type of R_n. We can obtain a **derived relationship set** by joining all the relationship sets in the attribute derivation. If we have

special relationship sets such as ISA in the derivation, then the join is over the identifiers of the superclass and subclass entity type of the ISA relationship set.

A special case of derived attributes occurs if the derivation of a view attribute A contains only special relationship sets. For example, let E be the base entity type of an external entity type E′. Suppose E is connected to another entity type F by one or more special relationship sets such that E ISA F, or F = UNION (. . . , E, . . .), or E = INTERSECT (. . . , F, . . .), or F = DECOMPOSE (. . . , E, . . .). E′ may contain attributes of F and we call such attributes **inherited attributes. Multilevel attribute inheritance** is allowed. If a view attribute A has associated with it some functions or arithmetic expressions, then we call A a **computed attribute.** A view attribute can also be obtained from a combination of computation and derivation, or computation and inheritance. We consider such an attribute as computed (for more details, see [22]).

Lee [22] proposes an ER schema and view data definition language. Fig. 2 shows an ER external view which is based on the example medical database in Fig. 1. We illustrate the view definition language obtained during the construction of this external view in an ER based DBMS Workbench [23]. This is a user-friendly graphical tool which allows the design of database conceptual schema, definition of user views based on a schema, and formulation of queries and updates against a view.

By default, a base entity type (or attribute) has the same name as its view entity type (or attribute). Otherwise, we need to specify the base object in the view definition. In cases of ambiguity, the system will internally ensure uniqueness of attribute name by attaching the name of the owner to the attribute. The system also assigns unique identifiers to special relationship sets if there exists any ambiguity. The view definition for Fig. 2 is as follows. The keywords are in italics.

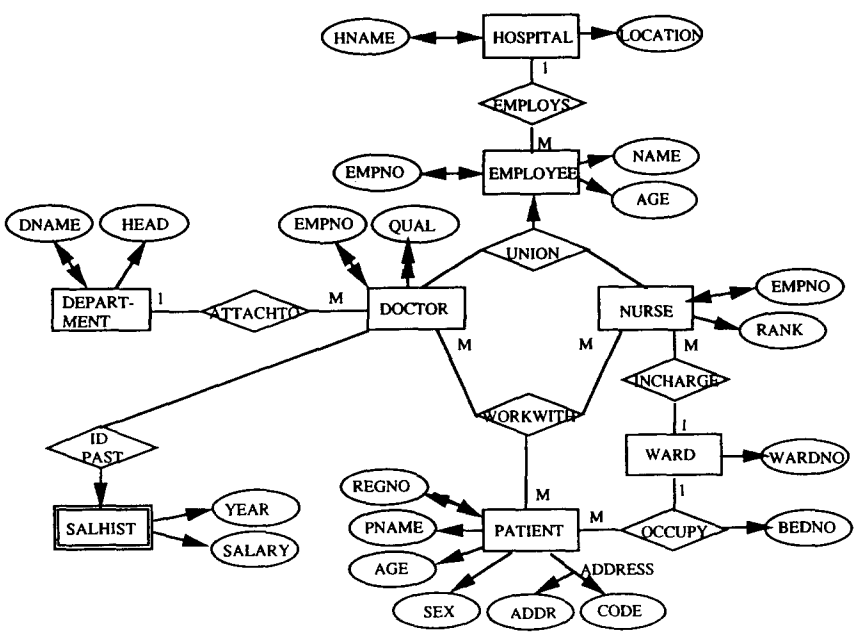


Fig. 1. An example ER medical database.

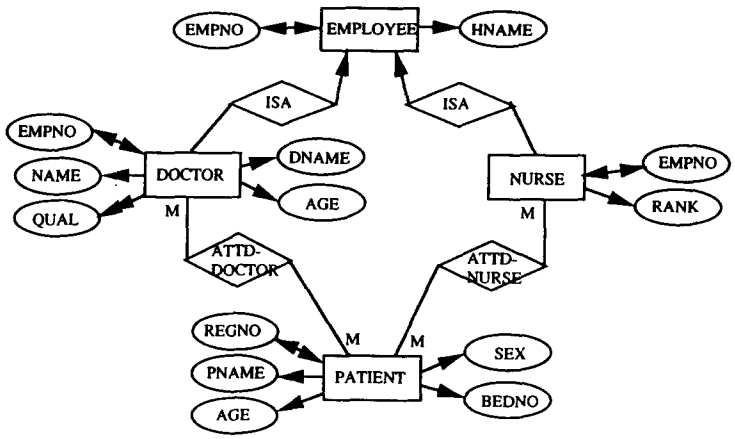


Fig. 2. An example ER external view of conceptual schema in Fig. 1.

VIEW DOCTPAT OF MEDICALDB
  VIEW ENTITY TYPE EMPLOYEE /*By default, base and view entity types have same name*/
    (*ATTRIBUTES* (EMPNO, /*Base attribute of EMPNO is in base entity type of EMPLOYEE*/
                HNAME *DERIVED* (⟨EMPLOYS⟩) *OWNER* (HOSPITAL))
    IDENTIFIER (EMPNO))
  VIEW ENTITY TYPE DOCTOR
    (*ATTRIBUTES* (EMPNO, QUAL,
                NAME *INHERITED* (⟨UNION⟩) *OWNER* (EMPLOYEE),
                AGE *INHERITED* (⟨UNION⟩) *OWNER* (EMPLOYEE),
                DNAME *DERIVED* (⟨ATTACHTO⟩) *OWNER* (DEPARTMENT))
    IDENTIFIER (EMPNO))
VIEW ENTITY TYPE PATIENT
    (*ATTRIBUTES* (REGNO, PNAME, AGE, SEX,
                BEDNO *DERIVED* (⟨OCCUPY⟩) *OWNER* (OCCUPY))
    IDENTIFIER (REGNO))
  VIEW ENTITY TYPE NURSE
    (*ATTRIBUTES* (EMPNO, RANK)
    IDENTIFIER (EMPNO))
  VIEW RELATIONSHIP SET ATTD-DOCTOR
    (*PART-VIEW-ENTITIES* (DOCTOR, PATIENT)
            /*PART-VIEW-ENTITIES denote participating view entity types*/
    IDENTIFIER (DOCTOR, PATIENT)
    DERIVATION (⟨WORKSWITH⟩))
  VIEW RELATIONSHIP SET ATTD-NURSE
    (*PART-VIEW-ENTITIES* (NURSE, PATIENT)
    IDENTIFIER (NURSE, PATIENT)
    DERIVATION (⟨INCHARGE, OCCUPY⟩))

ISA (*PART-VIEW-ENTITIES* (DOCTOR, EMPLOYEE)
DERIVATION (⟨UNION⟩))
ISA (*PART-VIEW-ENTITIES* (NURSE, EMPLOYEE)
DERIVATION (⟨UNION⟩))

Note that there are two possible derivations for the view relationship set ATTD-NURSE, that is, ⟨WORKWITH⟩ and ⟨INCHARGE, OCCUPY⟩. These derivations are automatically generated by the system and presented to the user for selection in the DBMS Workbench. For this example, the user selects the derivation ⟨INCHARGE, OCCUPY⟩.

4. View Updatability in ER approach

An ER user view can be represented in Prolog by using a predicate symbol for each entity type and relationship set [24]. Using Fig. 2 as an example, we have

EMPLOYEE (EMPNO, HNAME).
DOCTOR (EMPNO, NAME, AGE, QUAL, DNAME).
NURSE (EMPNO, RANK).
PATIENT (REGNO, PNAME, AGE, SEX, BEDNO).
ATTD-DOCTOR (DOCTOR, PATIENT).
ATTD-NURSE (NURSE, PATIENT).

Note that the entity types in a relationship set predicate are complex objects. For example, DOCTOR and PATIENT are complex objects in ATTD-DOCTOR. QUAL is a multivalued attribute and is thus a list in DOCTOR predicate. Any composite attribute is a complex object in its owner (entity type or relationship set) predicate. Any weak entity type is a list of complex objects in the parent entity type predicate.

Thus, views in ER approach are not necessarily flat relations. As a result, view update in the ER approach is different from that in relational model. It has the following important unique features.

(1) *Entity types*. Identifiers of entity types are not modifiable. This is because they are used as object identifiers in the relationship sets in which the entity types participate in. Modification of entity type identifiers will cause undesirable updating anomalies. The insertion of an entity requires the identifier value to be defined.

(2) *Relationship sets*. Identifiers of relationship sets can be modified without causing any side effects or updating anomalies. This is because a relationship specifies the way participating entities are related. The attributes of a relationship set and the identifiers of the participating entity types can be modified. Take for example in Fig. 2, we may have a user request to change the attending doctor of the patient Mr Ng (Regno 05211), from Dr Lee (Empno 114211) to Dr Chew (Empno 114220) as follows.

```
?-modify (attd-doctor (doctor (114211, _, _, _, _), patient (05211, _, _, _, _)),
          attd-doctor (doctor (114220, _, _, _, _), patient (05211, _, _, _, _))).
```

Note that internally for relationship sets, we only store the attributes of a relationship set and the identifiers of its participating entity types. Hence, to specify any update of a relationship set, the user needs to supply only the values of the attributes of the relationship set and the values of the identifiers of its participating entity types. No update is allowed on the non-identifier attributes of the participating entity types. Thus, in the above modification request, anonymous variables are used for the attributes of the participating entity types. Note that when we refer to the attributes of a relationship set, it does not include the attributes of the participating entity types of the relationship set. The insertion of a relationship requires the identifier values of **all** the participating entity types to be defined. This is because a relationship set is an association among the participating entity types. It violates the meaning of a relationship set in an ER database if we allow insertion to occur when only a partial (including some but not all participating entity type identifiers) identifier is given.

(3) *Multivalued attributes and weak entity types*. Weak entity types are set-valued attributes in the parent entity type predicate. Multivalued attributes are also lists in the owner predicate. We use set operations such as REMOVE and APPEND to update such attributes. For example, to reflect the fact that Dr Chew, employee number 114211, has just received his MFRC degree and will be transferred to the pediatrics department, we can have the following Prolog goal to update the view entity type DOCTOR in Fig. 2.

```
?-retrieve (doctor (114220, Name, Age, Qual, Dname)),
  append (Qual, ['MFRC'], NewQual),
  modify (doctor (114220, Name, Age, Qual, Dname),
          doctor (114220, Name, Age, NewQual, pediatrics)).
```

(4) *Special relationship sets*. Special relationship sets such as ISA, UNION, INTERSECT, etc. are actually constraints and hence cannot be updated. However, inherited attributes can be modified using the identifiers of the participating entity types in these special relationship sets.

We have the following principles that guide us in updating ER views. The first two principles are generally well-accepted and most works [10, 11] have been based on them.

(1) There must be a clear one-to-one correspondence between the objects (attributes, entity types and/or relationship sets) in the view and the underlying database schema so that we can uniquely translate the view updates into the corresponding updates on the conceptual schema. That is, there is **no ambiguity of origin** in the view objects. Otherwise, certain anomalies may occur when translating the view updates. There may be spurious tuples appearing or disappearing from the view after a view insertion or deletion.

(2) The result of a view update must not violate the definition of the view. This is because a user will not be able to retrieve the new updated data through the view since they do not meet the conditions specified by the view. For example, we may define a view which selects all the parts that are red or blue from a supplier-part database. Then, we will not allow an update which changes the part's color to, say, yellow. We can enforce such an update rule by including the selection criteria of views in the mapping rules.

(3) Side effects that are results of the system's actions to ensure that changes requested in a view are consistent with the rest of the database are permitted. The following definition introduces the concept of **virtual update** to refer to such side effects.

Definition 1. Let A be a subset of the attributes of an entity type or a relationship set in a view. Let B be an attribute in the entity type or relationship set such that $B \notin A$. If the value of the base attribute of B is a function of the values of the base attributes of A, then the modification of any of the attributes in A will cause the system to retrieve or re-compute the corresponding value of B whenever the value of B is required. We call such an action **virtual update.**

Note that virtual updates are automatically carried out by the system and not the user to maintain database consistency after a view update. Virtual updates are important in the following cases.

(1) Computed attributes in a view are not directly modifiable by the user but their values can be implicitly updated by the system. For example, suppose we can modify attributes A and B of a view entity type E but not the computed attribute $C = (A + B)/2$. But, the modification of A and/or B will cause the system to re-compute the value of C whenever the value of C is required.

(2) Let a view entity type E′ have a base entity type E. Suppose E′ contains attributes $A_1, A_2, \ldots, A_k$ whose base attributes are not in E but in another entity type F connected to E by relationship sets $R_1, R_2, \ldots, R_n$. If the base attribute of A_1 is the identifier of the entity type F, and A_1 has been determined to be modified in the view entity type E′, then the modification of A_1 will automatically cause the system to retrieve the corresponding values of the attributes $A_2, \ldots, A_k$ whenever these values are required. For example, suppose the view entity type DOCTOR in Fig. 2 also contains the attribute HEAD from the entity type DEPARTMENT. Then, the modification of DNAME in DOCTOR will cause the system to retrieve the corresponding value for HEAD when required. Note that HEAD is not modifiable by the user.

(3) Let a view relationship set R′ have a derivation $\langle R_1, R_2, \ldots, R_n \rangle$. Suppose R′ contains attributes $A_1, A_2, \ldots, A_k$ whose base attributes are in an entity type E. E is a participating entity type in some relationship set in the derivation, say R_i, for some i where $1 \leq i \leq n$. If the base attribute of A_1 is the identifier of E, and A_1 has been determined to be modifiable in the view relationship set R′, then the modification of A_1 will cause the system to retrieve the values of the attributes $A_2, \ldots, A_k$ whenever these values are required.

5. A theory for ER View Update

Next, we give a theory within the framework of the ER approach that characterizes the conditions under which there exist mappings from view updates into updates on the conceptual schema. Note that an entity type or relationship set is updatable if and only if the

entity type or relationship set is deletable, modifiable or insertable. We first examine the conditions under which a view entity type or relationship set is deletable or modifiable. A view entity type or relationship set is deletable (or modifiable) if we are able to delete (or modify) some corresponding entities or relationships in the database without violating any of the three view update principles stated in Section 4.

Definition 2. A **key-preserving projection** is a projection of an entity type or relationship set which includes a key of the entity type or relationship set.

Theorem 1. *Any view entity type is deletable. Let E be the base entity type of a view entity type E'. Any view attribute of E' whose base attribute is in E and is not part of the identifier of E is modifiable.*

Proof. We observe that a view entity type is always a key-preserving projection of its base entity type. Thus, we can always use the given key value of E' to retrieve the corresponding entity in E to delete or modify. Recall that the identifier value of any entity type cannot be modified.

Note that there are two ways to enforce the referential constraints when an entity is deleted. The first approach is to cascade the entity deletion into deletion of any relationships the entity is involved in. The second approach is that the user must explicitly issue a request to delete the relationships the entity is involved in before he issues a request to delete the entity.

Two sets of attributes X and Y in the relational model are said to be **functionally equivalent** if and only if $X \rightarrow Y$ and $Y \rightarrow X$. We can determine the functional equivalence of these two sets of attributes using Armstrong's axioms [25].

Definition 3. Two sets of entity types S_1 and S_2 are **functionally equivalent,** denoted $S_1 \leftrightarrow S_2$, w.r.t. a derivation $\langle R_1, R_2, \ldots, R_n \rangle$, if and only if

(1) S_1 participates in R_1 while S_2 participates in R_n and
(2) we can establish that the set of identifiers of the entity types in S_1 is functionally equivalent to the set of identifiers of the entity types in S_2 from the functional dependencies in the relationship sets $R_1, R_2, \ldots, R_n$.

Definition 4. Given relationship sets $R_1, R_2, \ldots, R_n$, let R be the join of $R_1, R_2, \ldots, R_n$ and S be the set of participating entity types of R whose identifiers form a key of R. Let S_i be the set of participating entity types of the relationship set R_i whose identifiers form a key of R_i, for some $i \in \{1, 2, \ldots, n\}$. We say that R and R_i are **functionally equivalent**, denoted $R \leftrightarrow R_i$, w.r.t. derivation $\langle R_1, R_2, \ldots, R_n \rangle$, if and only if S and S_i are functionally equivalent w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$.

Fig. 3a shows an ER diagram in which {A} is functionally equivalent to {B} w.r.t. $\langle R_1 \rangle$, but $\{A\} \rightarrow \{B\}$, $\{B\} \not\rightarrow \{A\}$ w.r.t. $\langle R_2 \rangle$. Since $\{B\} \leftrightarrow \{C\}$ in R_3, we can conclude that $\{A\} \rightarrow \{C\}$ from the functional dependencies in $\langle R_1, R_3 \rangle$ (or we can say that $\{A\} \leftrightarrow \{C\}$ w.r.t. $\langle R_1, R_3 \rangle$) by transitivity. On the other hand, Fig. 3b shows an ER diagram in which the

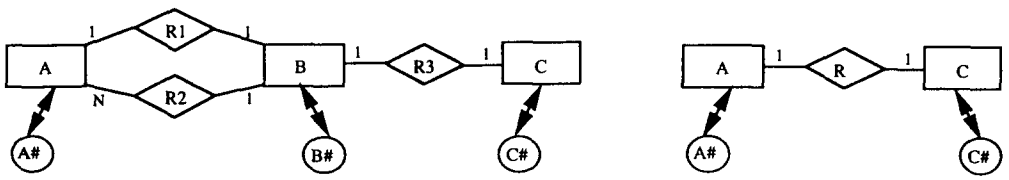


Fig. 3(a). Entity types A, B and C are all functionally equivalent; (b) Relationship set R, obtained by joining R1 and R3 in (a), is functionally equivalent to both R1 and R3.

relationship set R is obtained by joining relationship sets R_1 and R_3 over the common entity type B. We have $R \leftrightarrow R_1$ and $R \leftrightarrow R_3$ w.r.t. $\langle R_1, R_3 \rangle$ since the identifiers of these relationship sets are functionally equivalent to w.r.t. $\langle R_1, R_3 \rangle$.

Theorem 2. *Let R be a view relationship set with the relationship derivation* $\langle R_1, R_2, \ldots, R_n \rangle$. *R has the following updatability if and only if R is functionally equivalent to some relationship set* R_i *w.r.t.* $\langle R_1, R_2, \ldots, R_n \rangle$ *where* $i \in \{1, 2, \ldots, n\}$:

(1) R is deletable and

(2) R is modifiable for those attributes which are also attributes of R_i.

Proof. If the view relationship set R is functionally equivalent to some conceptual schema relationship set R_i w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$, where $i \in \{1, 2, \ldots, n\}$, then we have a one-to-one correspondence between the relationships of R and the relationships of R_i. Thus, when we delete a relationship of R, we delete the corresponding relationship of R_i which is retrieved using the key value of R. Moreover, when we modify the values of the attributes of a relationship of R, the corresponding attributes' values of the corresponding relationship in R_i retrieved using the key value of R are modified. Otherwise, if R is not functionally equivalent to any of the relationship set R_i w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$, where $i \in \{1, 2, \ldots, n\}$, then there will not be a one-to-one correspondence between the relationships of R and the relationships of R_i. R_i is not the base relationship set of R and the system will not be able to determine uniquely the relationship to be deleted or modified.

Corollary 1. *A view relationship set obtained from a* ***key-preserving*** *projection of a base relationship set is modifiable and deletable.*

Definition 5. A **derived relationship set of a view attribute** of a view entity type is obtained by joining all the relationship sets in the attribute derivation and projecting out all the participating entity types of the relationship sets in the attribute derivation except the base entity type of the view entity type and the owner entity type of the view attribute. We can also find a key of the derived relationship set from the set of functional dependencies in the attribute derivation.

Theorem 1 restricts the modifiable attributes of a view entity type to those view attributes whose base attributes are in the base entity type of the view entity type. However, we can

apply the argument used in proving Theorem 2 to extend the modifiable attributes of a view entity type to include certain derived attributes. We generalize this concept of modifying the derived attributes of a view entity type when certain conditions are satisfied in the following theorem.

Theorem 3. *Let E be the base entity type of a view entity type E'. Let A be a single-valued attribute of E' with the attribute derivation* $\langle R_1, R_2, \ldots, R_n \rangle$. *If the base attribute of A is the identifier of the entity type F, a participating entity type in* R_n, *then A is modifiable if and only if the derived relationship set of A is functionally equivalent to* R_n *w.r.t.* $\langle R_1, R_2, \ldots, R_n \rangle$.

Proof. The derived binary relationship set R of the view attribute A is constructed by joining all the relationship sets in the attribute derivation of A and projecting out all the participating entity types of $R_1, R_2, \ldots, R_n$ except E and F. Note that the construction of the derived relationship set is similar to the construction of view relationship sets. There is a one-to-one correspondence between the relationships of R and the relationships of R_n if and only if R is functionally equivalent to R_n w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$. If the base attribute of A is an identifier of F, then it is part of the relationship set R_n. Since A is a single-valued attribute in E', therefore there is a one-to-one correspondence between the entities in E' and the relationships in R. Hence A is modifiable if and only if R is functionally equivalent to R_n w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$.

Example 1. The single-valued derived attribute DNAME in the view entity type DOCTOR in Fig. 2 is modifiable according to Theorem 3. The derived relationship set of DNAME is functionally equivalent to ATTACHTO. Since the base attribute of DNAME is the identifier of the entity type DEPARTMENT, it is part of the relationship set ATTACHTO. Moreover, since DNAME is a single-valued attribute in the view entity type DOCTOR, there is a one-to-one correspondence between the view entities in DOCTOR and the relationships in ATTACHTO. Hence, when we modify the value of DNAME of a view entity in DOCTOR, the value of the base attribute of DNAME in the corresponding relationship in ATTACHTO retrieved using the key value of DOCTOR is modified.

Note that we do not allow the modification of any multivalued derived view attribute A as it will be ambiguous. Each value of A, which is a set, will correspond to a set of relationships in the conceptual schema and there is no unique translation of the modification request. On the other hand, each value of a single-valued derived view attribute will correspond to a unique relationship in the conceptual schema.

Corollary 2. *Let E be the base entity type of a view entity type E'. If E' contains a single-valued attribute A whose base attribute is not in E, but is the identifier of another entity type F which is connected to E by some regular binary relationship set R, then A is modifiable.*

Corollary 3. *Let E be the base entity type of a view entity type E'. Let A be a single-valued attribute of E' with the attribute derivation* $\langle R_1, R_2, \ldots, R_n \rangle$. *If the base attribute of A is the*

identifier of an entity type F, a participating entity type in R_n, then A is modifiable if and only if the derived relationship set of A is functionally equivalent to R_n w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$.

We next consider insertion in the ER approach. A view entity type or view relationship set is insertable if we are able to insert some corresponding entities or relationships into the database without violating any of our three view update principles stated in Section 4. Moreover, the entities or relationships inserted into the ER database are subjected to meet the domain constraints, the key constraints, as well as the referential constraints in the case of a relationship insertion. We now examine the conditions under which a view entity type or view relationship set is insertable.

Theorem 4. *A view entity type is insertable if and only if the identifier of its base entity type is included in the view.*

Proof. A request to insert a new entity in a view entity type will be translated into a request to insert a corresponding entity in its base entity type. Now to insert an entity into the database, we require its identifier value to be given. Thus, we can only insert a new entity into a view entity type if and only if the identifier of its base entity type is included in the view.

Example 2. To insert a new patient into the view entity type PATIENT in Fig. 2, we create a new patient entity in its base entity type. The values of the identifier and the attributes which also appear in the view are assigned as given by the user. Null values are assigned to the attributes of the base entity type which do not appear in the view. Hence, we obtain a new well-defined patient entity to be inserted into the database. Note that this new entity is still subjected to domain constraints and key constraint checks. We will discuss the insertion of derived view attributes such as BEDNO in the view entity type PATIENT later in this section.

Corollary 4. *A view entity type obtained from the selection of a base entity type is always updatable.*

Proof. From Theorems 1 and 4.

Theorem 5. *Let R be a view relationship set with relationship derivation $\langle R_1, R_2, \ldots, R_n \rangle$. R is insertable and new values can be given for those attributes of R whose base attributes are in some relationship set R_i where $i \in \{1, 2, \ldots, n\}$ if R is functionally equivalent to R_i w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$ and all the participating entity types of R_i are also the base entity types of the participating view entity types of R. We say that R_i is a* ***Type 1 base relationship set*** *of R.*

Proof. If R is functionally equivalent to some relationship set R_i w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$, then we have a one-to-one correspondence between the relationships of R and the relationships of R_i. R_i is a base relationship set of R. Thus, the insertion of a new relationship into R will be translated into an insertion of a corresponding relationship into R_i. Now, to insert a relationship into the database, we require the identifier values of its participating entities to be given. Note that the ER database will enforce the referential constraints for the new

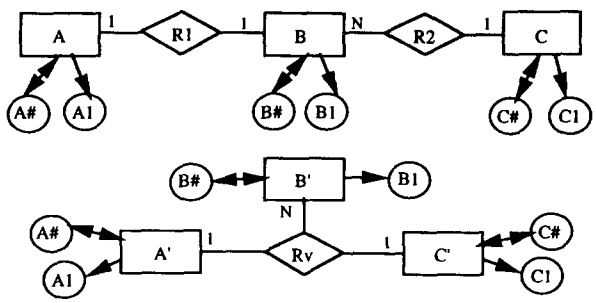


Fig. 4. A view relationship set Rv obtained from a join of the conceptual schema relationship sets R1 and R2.

relationship, that is, ensure that the participating entities in the new relationship exist in the database. Thus, we can insert a new relationship into R with new values given to those attributes of R whose base attributes are in R_i if all the participating entity types of R_i are also the base entity types of the participating view entity types of the view relationship set.

Corollary 5. *A view relationship set obtained from the selection of a conceptual schema relationship set is always updatable.*

Proof. From Theorems 2 and 5.

Example 3. Fig. 4 shows a view relationship set R_v obtained from a join of two conceptual schema relationship sets R_1 and R_2. A′, B′ and C′ are the view participating entity types of R_v whose base entity types are A, B and C, respectively. Both R_1 and R_2 are Type 1 base relationship sets of R_v according to Theorem 5. To insert a relationship (a, b, c) into R_v, we insert the relationships (a, b) and (b, c) into R_1 and R_2, respectively, if they do not already exist in database. Otherwise, if both the relationships exist in the database, we reject the insertion.

Example 4. Fig. 5 shows a view relationship set R_w obtained from a join of the two conceptual schema relationship sets R_1 and R_2 in Fig. 4. Here, the common entity type B has been projected out from the view. Although R_w is not insertable according to Theorem 5, it is possible to insert a relationship (a, c) into R_w without violating any of our view update principles. We first check if a is participated in some relationship in R_1, that is, if there exists an entity b of B such that the relationship (a, b) is in R_1. If the relationship (a, b) exists in R_1 and the relationship (b, c) does not exist in R_2, then we can insert (b, c) into R_2. Otherwise, we reject the insertion.

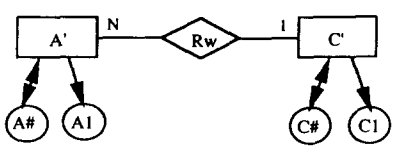


Fig. 5. A view relationship set Rw obtained from a join of the conceptual schema relationship sets R1 and R2 with the common entity type B projected out.

We have a few observations from Example 4.

(1) Two possible situations can occur when we insert a relationship (a, c) into R_w. We try to retrieve the identifier value of B from R_1 using the key value of A'.
Case 1. The relationship (a, b) does not exist in R_1.
That is, a has not participated in any of the relationships in R_1. For this case, there is no way we can insert the relationship (a, c) into R_w. Hence, we reject the insertion.
Case 2. The relationship (a, b) exists in R_1.
Using the retrieved identifier value of B, we can insert a relationship (b, c) into R_2 and still satisfy our three view update principles. Hence, the insertion of (a, c) into R_w is translated into the insertion of (b, c) into R_2.

(2) Although R_w is not insertable according to Theorem 5, we have seen that it may still be possible to insert a relationship into R_w. We observe that although the participating entity type B of R_2 does not appear as a base entity type of some participating entity type of R_w, the base entity type A of A' is functionally equivalent to B w.r.t. R_1. Hence, there is a one-to-one correspondence between the relationships in R_w and the relationships in R_2. R_2 is a base relationship set of R_w.

(3) In the ER approach, the existence of an entity in a relationship could be defined as either **mandatory** or **optional**[1]. If we know that the existence of the entity type A in the relationship set R_1 is mandatory, then we can always retrieve the identifier value of B in R_1 given a key value of A. Thus, we can always find the mapping to translate any insertion requests on R_w.

Definition 6. Suppose an entity type E_0 is involved in a relationship set R_0 with another entity type E_1, and E_1 is involved in a relationship set R_1 with an entity type E_2, and so on, and eventually we have an entity type E_{j-1} involved in a relationship set R_{j-1} with an entity type E_j. If the existence of E_k is mandatory in R_k (which may be n-ary) for $0 \leq k < j$, then we say that the existence of E_0 is **transitively mandatory** in the relationship set R which is obtained from a natural join of all the relationship sets R_k.

Based on the above discussion, we have the following theorem.

Theorem 6. *Let R be a view relationship set with the relationship derivation* $\langle R_1, R_2, \ldots, R_n \rangle$. *R is insertable and new values can be given to those attributes whose base attributes are in some relationship set* R_i *where* $i \in \{1, 2, \ldots, n\}$, *if R is functionally equivalent to* R_i *w.r.t.* $\langle R_1, R_2, \ldots, R_n \rangle$, *and for each participating entity type E of* R_i *either* (1) *E is a base entity type of some participating view entity types of R, or* (2) *E is functionally equivalent to some entity type F w.r.t. a derivation T such that F is a base entity type of some participating view entity type of R and T is* $\langle R_j, R_{j+1}, \ldots, R_k \rangle$, $1 \leq j \leq k \leq n$.

We say that R_i is a **Type 3 base relationship set** of R. Moreover, if the existence of the entity

[1] Optional existence, denoted by a 'o' on the connectivity line between an entity type and a relationship set, defines a minimum cardinality of zero; mandatory existence defines a minimum cardinality of one.

type F is transitively mandatory in the relationship set which is obtained from a join of all the relationship sets in the derivation T, then we say that R_i is a **Type 2 base relationship set** of R.

Proof. If R is functionally equivalent to some relationship set R_i w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$, then we have a one-to-one correspondence between the relationships of R and the relationships of R_i. R_i is a base relationship set of R. Thus, the insertion of a new relationship into R will be translated into an insertion of a corresponding relationship into R_i. Now, to insert a relationship into the database, we require the identifier values of its participating entities to be given. Thus, for each participating entity type E of R_i, if E is a base entity type of some participating view entity type of R, then the value of the identifier of E will be given. On the other hand, if E satisfies condition 2 in the Theorem, then we can attempt to retrieve the identifier value of E through the derivation T from the given identifier value of F. Therefore, R is insertable.

Example 5. Fig. 6 shows an insertable view relationship set R_x obtained from a join of three conceptual schema relationship sets, R1, R_2 and R_3. A', B' and D' are the view participating entity types of R_x whose base entity types are A, B and D, respectively. Clearly, R_1, R_2 and R_3 are all functionally equivalent to R_x w.r.t. to $\langle R_1, R_2, R_3 \rangle$. R_1 is a Type 1 base relationship set of R_x according to Theorem 5. Although R_2 is functionally equivalent to R_x w.r.t. $\langle R_1, R_2, R_3 \rangle$ and B is the base entity type of B', C is not functionally equivalent to any entity type F w.r.t. a derivation such that F is a base entity type of some participating entity type of R_x. Therefore, according to Theorem 5, R_2 is not a Type 1 base relationship set of R_x. According to Theorem 6, R_2 is also neither a Type 2 nor Type 3 base relationship set of R_x. On the other hand, we see that C is functionally equivalent to B w.r.t. $\langle R_2 \rangle$. Hence, R_3 is a Type 2 base relationship set of R_x if B is mandatory in R_2 according to Theorem 6. Otherwise, R_3 is a Type 3 base relationship set of R_x.

The above example also illustrates that an insertable view relationship set can have more than one type of base relationship sets.

Definition 7. A view relationship set is **Type 1 insertable** if it has some Type 1 base relationship sets but has neither Type 2 nor Type 3 base relationships sets. A view relationship set is **Type 2 insertable** if it has some Type 2 base relationship sets but has no Type 3 base

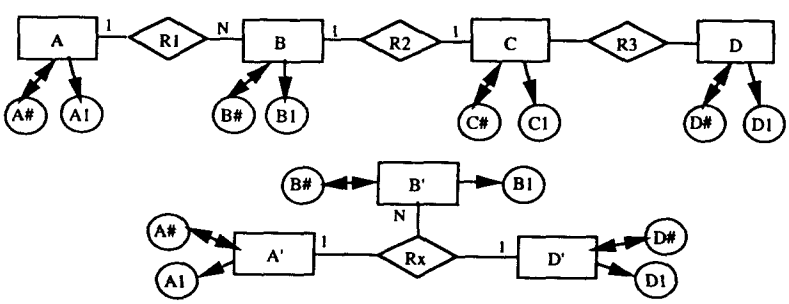


Fig. 6. A view relationship set Rx obtained from a join of the conceptual schema relationship sets R1, R2 and R3.

relationship sets. A view relationship set is **Type 3 insertable** if it has Type 3 base relationship sets.

We conclude in the following theorem that if a view relationship set is neither Type 1 nor Type 2 nor Type 3 insertable, then it is not insertable at all.

Theorem 7. *If a view relationship set is neither Type 1 nor Type 2 nor Type 3 insertable, then it is* not *insertable.*

Proof. Let R be a view relationship set with the relationship derivation $\langle R_1, R_2, \ldots, R_n \rangle$. Since R is neither Type 1 nor Type 2 nor Type 3 insertable, therefore for all the relationship sets R_j, $1 \leq i \leq n$, R_i is neither a Type 1 nor a Type 2 nor a Type 3 base relationship set of R.

By Theorems 5 and 6, for all the relationship sets R_i, $1 \leq i \leq n$, either case (1) R is not functionally equivalent to R_i w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$, or case (2) there exists some participating entity E of R_i such that E is not the base entity type of any participating view entry type of R, and E is not functionally equivalent to any entity type F w.r.t. a derivation T such that F is the base entity type of some participating view entity type of R and T is $\langle R_j, R_{j+1}, \ldots, R_k \rangle$, $1 \leq j \leq k \leq n$. Let us look at these two cases in detail.

Case (1). Now if R is not functionally equivalent to R_i w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$, then we do not have a one-to-one correspondence between the relationships of R and the relationships of R_i. The insertion of any new relationship into R cannot be translated into an insertion of some relationship into R_i. Therefore, R_i is not a base relationship set of R for insertion.

Case (2). If there exists some participating entity type E of R_i such that E is not the base entity type of any participating view entity types of R, and E is not functionally equivalent to any entity type F w.r.t. a derivation T such that F is the base entity type of some participating view entity type of R and T is $\langle R_j, R_{j+1}, \ldots, R_k \rangle$, $1 \leq j \leq n$, then during an insertion of R, there is no way we can obtain the (unique) identifier value of E such that there will not be any spurious tuples appearing in R after the view insertion. Therefore, R_i is not a base relationship set of R for insertion.

From the above discussion, R_i is not a base relationship set of R for insertion for all $i = 1, 2, \ldots, n$. Hence R has no base relationship sets for insertion, that is, R is not insertable.

Theorem 4 restricts the attributes of a view entity type which can be given values in an insertion of a view entity to those view attributes whose base attributes are in the base entity type of the view entity type. However, in certain cases, we can allow values to be given to derived attributes of a view entity type in an insertion of a new entity without violating any of our view update principles.

Example 6. We may want to insert a new doctor into the view entity type DOCTOR in Fig. 2, and at the same time give the name of the department the doctor is attached to. This insertion request can be translated into an insertion of a corresponding entity into the base entity type of DOCTOR and an insertion of a relationship into the conceptual schema relationship set ATTACHTO. The new relationship which is inserted into ATTACHTO is created using the

identifier values of its two participating entity types, DOCTOR and DEPARTMENT, that is, the user-given values for the attributes EMPNO and DNAME which are both in the view entity type DOCTOR. There is a one-to-one correspondence between the relationships in ATTACHTO and the entities in the view entity type DOCTOR. Hence, when we give a value to the derived attribute DNAME in an insertion of a new view entity into the view entity type DOCTOR, a new relationship is inserted into the relationship set ATTACHTO in the database in addition to the insertion of a corresponding entity into the base entity type of DOCTOR.

We say that a view attribute of a view entity type or view relationship set is **insertable** if values can be given to it in an insertion of a new view entity or view relationship into the view entity type or view relationship set respectively. For example, the view attributes of a view entity type whose base attributes are in the base entity type of the view entity type are insertable. We generalize the concept of *insertable derived attributes* in the following theorem.

Theorem 8. Let E be the base entity type of a view entity type E′ and let A be a *derived* attribute of E′ with the attribute derivation $\langle R_1, R_2, \ldots, R_n \rangle$ such that R_n is a binary relationship set. Suppose E_n is a common entity type of R_{n-1} and R_n, and F is the other participating entity type of R_n such that the base attribute of A is the identifier of F. A is insertable if:

(1) derived relationship set R of A is functionally equivalent to R_n w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$, and
(2) E is functionally equivalent to E_n w.r.t. $\langle R_1, R_2, \ldots, R_{n-1} \rangle$, and
(3) E is transitively mandatory in the relationship set which is obtained from a join of the relationship sets $R_1, R_2, \ldots, R_{n-1}$.

Proof. Recall that the derived relationship set R is obtained by joining all the relationship sets in the attribute derivation of A and projecting out all the participating entity types of $R_1, R_2, \ldots, R_n$ except E and F. There is a one-to-one correspondence between the relationships of R and the relationships of R_n if and only if R is functionally equivalent to R_n w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$. If E is functionally equivalent to E_n w.r.t. $\langle R_1, R_2, \ldots, R_{n-1} \rangle$, and E is transitively mandatory in the relationship set obtained from a join of the relationship sets $R_1, R_2, \ldots, R_{n-1}$, then R is Type 2 insertable with R_n as its base relationship set. If the base attribute of A is the identifier of F, then it is part of the relationship set R_n. Therefore, if A is single-valued attribute in E′, then when A is given a value during an insertion of a view entity into E′, we can insert a new relationship into the binary relationship set R_n using the retrieved identifier value of E_n and the given value of A. If A is a multivalued attribute in E′, then a set of values S will be given to A during an insertion of a view entity into E′. In this case, we will insert |S| new relationships into R_n. Each of these new relationships is created using the retrieved identifier value of E_n and a value in S. These insertions will not cause any violation of our view update principles. Hence, A is insertable.

Corollary 6. *Let E be the base entity type of a view entity type E′. If E′ contains an attribute A*

whose base attribute is not in E, but is the identifier of another entity type F which is connected to E by some regular binary relationship set R, then A is insertable.

For example, suppose we have a view entity type DEPT (base entity type is DEPT in Fig. 1) with a derived multivalued attribute EMPNO whose base attribute is the identifier of the entity type DOCTOR which is connected to DEPT by the relationship set ATTACHTO (see Fig. 1). EMPNO is insertable according to Corollary 6. If we are given a set S of doctors to be inserted into a particular department in the view entity type DEPT, then we will need to insert |S| relationships into ATTACHTO. Each of these relationships is created using the department name and the employee number of a doctor in S.

Corollary 7. *An inherited attribute is insertable.*

6. The View Updatability Algorithm

Based on the theory developed in the previous section, we present a View Updatability algorithm to systematically determine the updatability of view entity types and relationship sets in a view. In addition, this algorithm also determines the different types of insertability for view relationship sets. We will first give a structure chart of the calling sequence of the modules in the algorithm before going into the details of the algorithm. Note that there are no separate modules to determine if a view entity type or a view relationship set is deletable as these are quite trivial (see Fig. 7).

Algorithm 1 (Evaluation of View Updatability)
Input – A conceptual schema definition and a view definition.
Step 1: For each view entity type, evaluate its updatability using Algorithm 1.1.
Step 2: For each view relationship set, evaluate its updatability using Algorithm 1.2.

Algorithm 1.1 (Updatability of View Entity type)

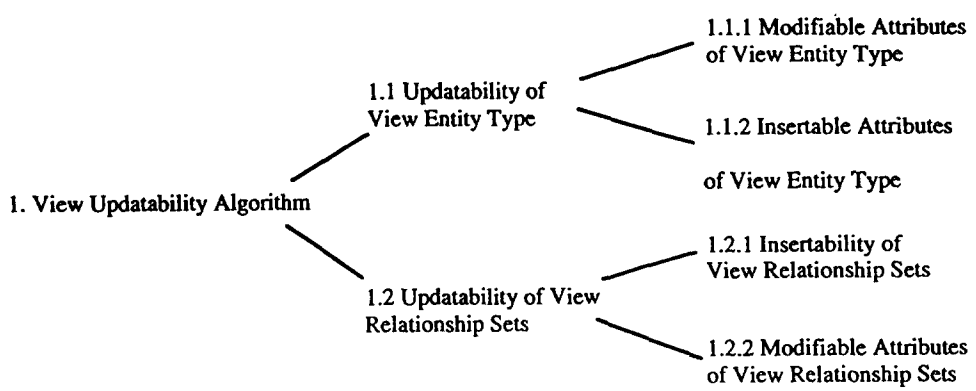


Fig. 7.

Let E be the base entity type of the view entity type E′ in the view definition.
Step 1: E′ is deletable. /*E and E′ are always functionally equivalent*/
Step 2: If E′ includes the identifier of E Then
  E′ is insertable
  Else E′ is not insertable.
Step 3: Group the same type of attributes (*such as derived, computed, etc.*) of E′ together. Further partition derived and inherited attributes such that attributes with the *same owner and derivation* are grouped together.
Step 4: For each group of attributes A_g do
  Call Algorithm 1.1.1 to determine the modifiable attributes in A_g.
  Call Algorithm 1.1.2 to determine the insertable attributes in A_g.
Step 5: Any weak entity type of E′ is modifiable and insertable.
  /*Weak entity types are considered as set-valued attributes*/

Algorithm 1.1.1 (Determine modifiable attributes of a View Entity type)
Let E be the base entity type of the view entity type E′.
Let A_g be a group of attributes obtained from Step 3 of Algorithm 1.1.
Case 1: Base attributes of A_g belong to the base entity type E of E′.
  For each attribute A in A_g do
    If base attribute of A is part of the identifier of E Then
      A is not modifiable
    Else A is modifiable.
Case 2: A_g is a set of computed attributes.
  Each attribute in A_g is not modifiable, but is virtually updatable.
Case 3: A_g is a set of inherited attributes.
  Each attribute in A_g is modifiable.
Case 4: A_g is a set of multivalued derived attributes.
  Each attribute in A_g is not modifiable.
Case 5: A_g is a set of single-valued derived attributes (*with same owner and derivation*).
  /*Based on Theorem 3*/
  Let $\langle R_1, R_2, \ldots, R_n \rangle$ be the attributes' derivation.
  Obtain the derived relationship set R of the attributes in A_g.
  If R and R_n are functionally equivalent w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$ Then
  5.1: Owner of the attributes in A_g is R_n.
    Each attribute in A_g is modifiable.
  5.2: Owner of attributes in A_g is F, a participating entity type of R_n.
    For each attribute A in Ag do
      If base attribute of A is the identifier of F Then
        A is modifiable.
      Else A is not modifiable, but is virtually updatable
  Else Each attribute in A_g is not modifiable.

Algorithm 1.1.2 (Determine insertable attributes of a View Entity type)
Let E be the base entity type of the view entity type E′ in the view definition.

Let A_g be a group of attributes obtained from Step 3 of Algorithm 1.1.
Case 1: Base attributes of A_g belong to the base entity type E of E'.
Each attribute in A_g is insertable.
Case 2: A_g is a set of computed attributes.
Each attribute in A_g is not insertable.
Case 3: A_g is a set of inherited attributes.
Each attribute in A_g is insertable.
Case 4: A_g is a set of derived (both single-valued and multivalued) attributes (*with same owner and derivation*).
/*Based on Theorem 8*/
Let $\langle R_1, R_2, \ldots, R_n \rangle$ be the attributes' derivation.
If the owner of the attributes in A_g is R_n Then
Each attribute in A_g is not insertable
Else Let the owner of the attributes in A_g be F, a participating entity type of R_n.
If R_n is binary Then
Obtain the derived relationship set R of the attributes in A_g.
If R and R_n are functionally equivalent w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$ Then
Set Flag = True. /*to cater for the case n = 1*/
If n > 1 Then
Let C be the common entity type of R_n and R_{n-1}.
If E and C are functionally equivalent w.r.t. $\langle R_1, R_2, \ldots, R_{n-1} \rangle$ and E is transitively mandatory in $\langle R_1, R_2, \ldots, R_{n-1} \rangle$ Then
Set Flag = True
Else Set Flag = False.
If Flag = True Then
For each attribute A in A_g do
If base attribute of A is identifier of F Then
A is insertable
Else A is not insertable
Else Each attribute in A_g is not insertable.
Else /*R and R_n not functionally equivalent*/
Each attribute in A_g is not insertable.
Else /*R_n is not binary*/
Each attribute in A_g is not insertable.

Example 7. The entity type DOCTOR in the view in Fig. 2 has an identifier EMPNO which is equal to the identifier of its base entity type. Hence, it is insertable and deletable according to Steps 1 and 2 of Algorithm 1.1. The same type of attributes of DOCTOR are grouped together. EMPNO and QUAL form the first group whose base attributes are in the base entity type of DOCTOR. DNAME, with attribute derivation ⟨ATTACHTO⟩, forms the second group. The inherited attributes, NAME and AGE, with attribute derivation ⟨UNION⟩, form the third group. According to Algorithm 1.1.1, all the attributes of DOCTOR except EMPNO are modifiable. According to Algorithm 1.1.2, all the attributes of DOCTOR are insertable. For instance, the derived relationship set of DNAME (obtained from the attribute derivation

⟨ATTACHTO⟩) is functionally equivalent to ATTACHTO in Case 5 of Algorithm 1.1.1. And since the base attribute of DNAME is the identifier of entity type DEPARTMENT in the conceptual schema, we determine that DNAME is modifiable. Similarly, in Case 4 of Algorithm 1.1.2, determine that DNAME is insertable.

Algorithm 1.2. (Updatability of View Relationship set)
Let R be the view relationship set with the relationship derivation $\langle R_1, R_2, \ldots, R_n \rangle$.
Step 1: Determine the set $S \subseteq \{R_1, R_2, \ldots, R_n\}$ where each relationship set in S is functionally equivalent to
R w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$.
If S is non-empty Then
R is deletable.
Call Algorithm 1.2.1 to determine the type of insertability of R and insertable attributes of R.
All the identifiers of the participating entity types of R are modifiable.
Other attributes of the participating entity types of R are not modifiable, but virtually updatable.
Else R is not deletable and not insertable.
All the identifiers and the attributes of the participating entity types of R, and all the attributes of R are not modifiable.
Return.
Step 2: Group the same type of attributes (*such as derived, computed, etc.*) of R together.
Further partition the derived and inherited attributes such that attributes with the same owner and derivation are grouped together.
Further partition the derived and inherited attributes such that attributes with the same owner and derivation are grouped together.
Step 3: For each group of attributes A_g do
Call Algorithm 1.2.2 to determine the modifiable attributes in A_g.

Algorithm 1.2.1 (Determine insertability of View Relationship set)
Let S be the set of relationship sets obtained in Step 1 of Algorithm 1.2 where each relationship set in S is functionally equivalent to the view relationship set R w.r.t. the relationship derivation $\langle R_1, R_2, \ldots, R_n \rangle$.
/*Determine the Type 1 base relationship sets of R.*/
Determine the set $U \subseteq S$ such that for each relationship set R′ in U, all the participating entity types of R′ are also the base entity types of the participating view entity types of R.
Let $T = S - U$.
If T is non-empty Then /*Determine the Type 2 & 3 base relationship sets of R.*/
Set V = Ø. /*V is a set of Type 2 base relationship sets of R*/
Set W = Ø. /*W is a set of Type 3 base relationship sets of R*/
For each relationship R′ in T do
Set Type2 = True. /*to indicate if R′ is a Type 2 base relationship set of R*/
Set Type3 = True. /*to indicate if R′ is a Type 3 base relationship set of R*/
For each of the participating entity types E of R′ and while Type3 = True do

/*Test conditions for Theorem 6*/
If E is not a base entity type of some participating view entity types of R Then
If E is not functionally equivalent to some entity type F w.r.t. a derivation T such that F is the base entity type of some participating view entity type of R and T is $\langle R_j, R_{j+1}, \ldots, R_k \rangle$,
$1 \Leftarrow j \Leftarrow k > = n$ Then
Set Type3 = False
Set Type2 = False
Else /*E is functionally equivalent to some entity type F w.r.t. a derivation T such that F is the base entity type of some participating view entity type of R and T is $\langle R_j, R_{j+1}, \ldots, R_k \rangle$, $1 \leqslant j \leqslant k \leqslant n$*/
If F is not transitively mandatory in the relationship set obtained from a join of all the relationship sets in T Then
Set Type2 = False.
If Type2 = True Then $V = V \cup \{R'\}$
Else If Type3 = True Then $W = W \cup \{R'\}$.
If W is non-empty Then R is Type 3 insertable
Else If V is non-empty Then R is Type 2 insertable
Else, if U is non-empty Then R is Type 1 insertable
Else R is not insertable.
Return.
/*R is insertable*/
Let $X = U \cup V \cup W$. /*X contain all the base relationship sets of R*/
The attributes of R whose base attributes are the attributes of the relationship sets in X are insertable.
Other attributes of R are not insertable.

Algorithm 1.2.2. (Determine modifiable attributes of a View Relationship set)
Let R be the view relationship set with the relationship derivation $\langle R_1, R_2, \ldots, R_n \rangle$.
Let A_g be a group of attributes obtained from Step 3 of Algorithm 1.2.
Case 1: A_g is a set of computed attributes.
Each attribute in A_g is not modifiable, but is virtually updatable.
Case 2: A_g is a set of inherited attributes.
Each attribute in A_g is modifiable.
Case 3: A_g is a set of derived attributes (*with the same owner and derivation*).
Let $\langle R_{d_1}, R_{d_2}, \ldots, R_{d_k} \rangle$ be the attributes' derivation.
If R and R_{d_j} are functionally equivalent w.r.t. $\langle R_{d_1}, R_{d_2}, \ldots, R_{d_k} \rangle$ for some $j \in \{1, 2, \ldots, k\}$ Then
Case 3.1: Owner of the attributes in A_g is R_{d_j}.
Each of the attributes in A_g is modifiable
Case 3.2: Owner of the attributes in A_g is F, a participating entity type of R_{d_j}.
For each attribute A in A_g do
If A is the identifier of F Then
A is modifiable

Else A is not modifiable but is virtually updatable
Else Each of the attributes in A_g is not modifiable.

Example 8. Both the view relationship sets ATTD-DOCTOR and ATTD-Nurse in Fig. 2 are not updatable as they are not functionally equivalent to any of the relationship sets in their relation derivations. The view relationship set R_v in Fig. 4 is deletable according to Step 1 of Algorithm 1.2. It is Type 1 insertable according to Algorithm 1.2.1. The view relationship set R_w in Fig. 5 can be Type 2 or Type 3 insertable depending on whether the entity type A is mandatory in R_1. The determination of modifiable attributes of view relationship sets is similar to the determination of modifiable attributes of view entity types.

7. The View Update Translation Algorithm

Next, we present a View Update Translation Algorithm to translate a view update request into the corresponding database update based on the results obtained from the View Updatability Algorithm. Information regarding the updatability of a view generated from the View Updatability Algorithm is stored in the data dictionary. The View Update Translation Algorithm will use this information during any view update request translation. The main idea of the View Update Translation Algorithm is to obtain the base entity type or base relationship set. For instance, to update a view entity type, the algorithm will find the base entity type. To update inherited or derived attributes, the algorithm will find the owner entity type or relationship set in the conceptual schema. We will first give a structure chart of the calling sequence of the modules in the algorithm before going into the details of the algorithm (see Fig. 8). Note again that there are no separate modules to translate the deletion of a view entity type or relationship set as these are quite trivial.

Algorithm 2 (Translation of View Updates)
Case 1: Update a view entity type.
Translate the update using Algorithm 2.1.

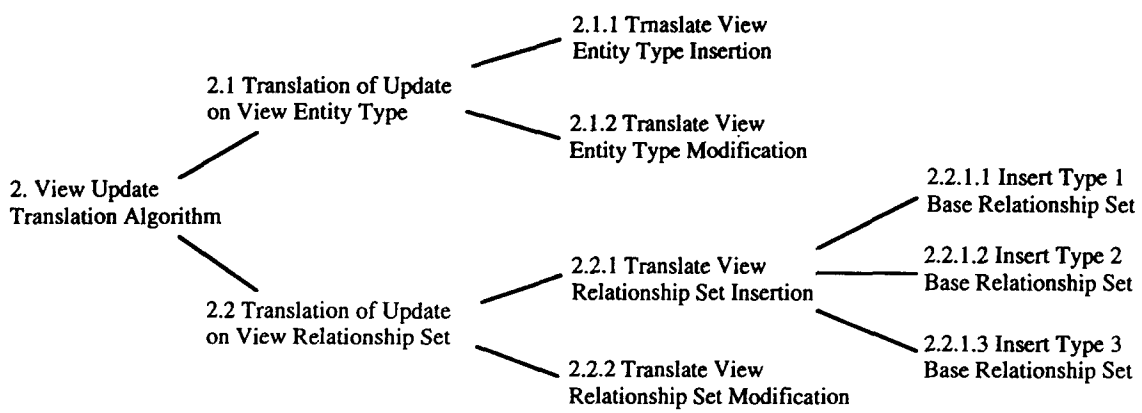


Fig. 8.

Case 2: Update a view relationship set.
Translate the update using Algorithm 2.2.

Algorithm 2.1 (Translation of Update on View Entity type)
Given a view containing an updatable view entity type E′ of a base entity type E and a view update against this entity type, we translate the view update as follows:
Case 1: Delete an entity in E′.
Delete the corresponding entity in E using the key value of E′.
Case 2: Insert an entity in E′.
Call Algorithm 2.1.1 to translate the insertion request.
Case 3: Modify an entity in E′.
Call Algorithm 2.1.2 to translate the modification request.

Algorithm 2.1.1 (Translate a View Entity type insertion)
Let E′ be the view entity type of a base entity type E in the view definition.
If E′ is insertable Then
For each group of attributes A_g obtained in Step 3 of Algorithm 1.1 (*where the attributes in A_g have the same owner and derivation*) do
Partition A_g into insertable and not insertable attributes.
Case 1: Insertable attributes.
Let A_i be the group of insertable attributes.
Case 1.1: Base attributes of A_i belong to E.
Create a new entity in E as follows:
Assign the new identifier value to new entity in E.
For attributes of E also found in E′, values are assigned as in E′.
For attributes of E not found in E′, null values are assigned.
Insert the new entity into E.
Case 1.2: A_i is a set of inherited attributes.
Let $\langle R_1, R_2, \ldots, R_n \rangle$ be the attributes' derivation.
/*Note that all the relationship sets in the derivation are special relationship sets such as ISA, UNION, etc*/
Let C be the superclass entity type of the special relationship set R_n obtained from the conceptual schema.
If the identifier value of the new entity in E′ does not match the identifier value of any entity in C Then
Create a new entity in C as follows:
Assign the new identifier value to the new entity.
For attributes of C also found in E′, values are assigned as in E′.
For attributes of C not found in E′, null values are assigned.
Insert the new entity into C.
Case 1.3: A_i is a set of derived attributes.
Let $\langle R_1, R_2, \ldots, R_n \rangle$ be the attributes' derivation.
Let E_i be the common entity type of R_i and R_{i+1}, for $1 \leq i < n$, and let E_n be the owner entity type of the base attribute of the attribute in A_i.

/*Note that A_i will only contain one insertable attribute which is the attribute whose base attribute is the identifier of E_n.*/
Let $E = E_0$.
For $1 \leqslant i < n$ do
Use the identifier value of E_{i-1} to retrieve the identifier value of E_i from R_i.
/*Recall that E and E_{n-1} are functionally equivalent w.r.t. $\langle R_1, R_2, \ldots, R_{n-1} \rangle$. Hence we can uniquely determine the identifier value of E_i from E_{i-1}.*/
Create a new relationship in R_n using the identifier values of E_{n-1} and E_n.
/*Note that R_n is binary and E_n is the value of the attribute in A_i*/
Insert the new relationship into R_n.

Case 2: Not insertable attributes.
If values are given to such attributes Then
Inform user of error.

Else
Reject the insertion request.

We observe that it is trivial to delete an entity from a deletable view entity type. However, to insert a new entity into an insertable entity type, we need to take into consideration the presence of inherited and/or derived attributes. If we have derived attributes in the view entity type, then in addition to the insertion of a corresponding entity into the base entity type of the view entity type, we will need to insert corresponding relationships into some relationship sets.

Example 9. To insert a new doctor into the view in Fig. 2, we have the Prolog goal

?-insert (doctor (116790, 'H. Goh', 35, ['MBBS', 'MMed'], surgery)).

Algorithm 2.1.1 will translate this view insertion request into the following three facts to be inserted into the database.

```
doctor (116790, ['MBBS', 'MMed']).     /*Case 1.1–The base attributes of EMPNO and
                                          QUAL are in the base entity type */
employee (116790, 'H. Goh', 35).       /*Case1.2– NAME and AGE are inherited
                                          attributes with derivation ⟨UNION⟩*/
attachto (116790, surgery).            /*Case 1.3–DNAME is a derived attribute with
                                          derivation ⟨ATTACHTO⟩*/
```

Algorithm 2.1.2 (Translate a View Entity type modification)
Let E′ be the view entity type of a base type E.
For each group of attributes A_g obtained in Step 3 of Algorithm 1.1 (*where the attributes in A_g have the same owner and derivation*) do
Partition A_g into modifiable and not modifiable attributes.

Case 1: Modifiable attributes.
Let A_m be the group of modifiable attributes.
Case 1.1: Base attributes of A_m belong to E.
Modify the values of the base attributes of A_m in the corresponding entity in E obtained using the key value of E′.
Case 1.2: A_m is a set of inherited attributes.
Modify the values of the base attributes of A_m in the corresponding owner entity obtained using the key value of E′.
Case 1.3: A_m is a set of single-valued derived attributes.
Let $\langle R_1, R_2, \ldots, R_n \rangle$ be the attributes' derivation.
Modify the values of the base attributes of A_m in the corresponding relationship in R_n obtained using the key value of E′.
Case 2: Not modifiable attributes.
If new values are given to such attributes. Then
Inform user of error.

Example 10. To modify a particular doctor in the view entity type DOCTOR in Fig. 2, we use the given key value of the doctor to retrieve and modify the corresponding doctor entity in the database if the attribute QUAL is given a new value. If either one or both the attributes NAME and AGE are given new values, we modify the corresponding employee entity in Case 1.2 of Algorithm 2.1.2. If the attribute DNAME is given a new value, we modify the corresponding attachto relationship in Case 1.3.

Similar forms of translations can be carried out for view update requests on relationship sets using the following algorithm.

Algorithm 2.2 (Translation of Update on View Relationship set)
Given a view with an updatable view relationship set R with the relationship derivation $\langle R_1, R_2, \ldots, R_n \rangle$ and a view update against this relationship set, we translate the view update as follows:
Case 1: Delete a relationship in R.
If R is deletable Then
Let S be the set of relationship sets obtained in Step 1 of Algorithm 1.2 where each relationship set in S is functionally equivalent to R w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$.
For each relationship set R′ in S do
Delete the corresponding relationship in R′ using the identifier value of R
Else Reject the deletion request.
Case 2: Insert a relationship in R.
Call Algorithm 2.2.1 to translate the insertion request.
Case 3: Modify a relationship in R.
Call Algorithm 2.2.2 to translate the modification request.

Algorithm 2.2.1 (Translate a View Relationship set insertion)
Let R be the view relationship set with the relationship derivation $\langle R_1, R_2, \ldots, R_n \rangle$.

If R is insertable Then
  Verify that only the insertable attributes of R are given values, otherwise inform the user of error.
  Case 1: R is Type 1 insertable.
    Call Algorithm 2.2.1.1 to insert corresponding Type 1 base relationships of R.
  Case 2: R is Type 2 insertable but not Type 3 insertable.
    Call Algorithm 2.2.1.1 to insert corresponding Type 1 base relationships of R.
    Call Algorithm 2.2.1.2 to insert corresponding Type 2 base relationships of R.
  Case 3: R is Type 3 insertable.
    Call Algorithm 2.2.1.1 to insert corresponding Type 1 base relationships of R.
    Call Algorithm 2.2.1.2 to insert corresponding Type 2 base relationships of R.
    Call Algorithm 2.2.1.3 to insert corresponding Type 3 base relationships of R.
Else /*R is not insertable.*/
  Reject the insertion request.

Algorithm 2.2.1.1 (Insert Type 1 base relationships)
Let R be the insertable view relationship set with derivation $\langle R_1, R_2, \ldots, R_n \rangle$.
Let U be the set of Type 1 base relationship sets obtained in Algorithm 1.2.1 where each relationship set R′ in U is functionally equivalent to R w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$ and all the participating entity types of R′ are the base entity types of the participating view entity types of R.
For each of the relationship sets R′ in U do
  Create a new relationship in R′ as follows:
    For the identifiers of the participating entity types of R′, values are assigned as in R.
    For attributes of R′ also found in R, values are assigned as in R.
    For attributes of R′ not found in R, null values are assigned.
  Insert the new relationship into R′.

Algorithm 2.2.1.2 (Insert Type 2 base relationships)
Let R be the insertable view relationship set with derivation $\langle R_1, R_2, \ldots, R_n \rangle$.
Let V be the set of relationship sets obtained in Algorithm 1.2.1 where each relationship set R′ in V is functionally equivalent to R w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$ and each of the participating entity types E of R′ either
  (1) E is the base entity type of some participating view entity types of R
or (2) E is functionally equivalent to some entity type F w.r.t. a derivation T such that F is the base entity type of some participating view entity type of R and T is $\langle R_j, R_{j+1}, \ldots, R_k \rangle$, $1 \leq j \leq k \leq n$. Moreover, F is transitively mandatory in the relationship set obtained from a join of all the relationship sets in T.
For each of the relationship set R′ in V do
  Create a relationship set in R′ as follows:
    For attributes of R′ also found in R, values are assigned as in R;
    For attributes of R′ not found in R, null values are assigned.
    For each of the participating view entity type E of R′ do
      If E satisfies Condition 1 above Then

The identifier value of E is assigned as given in R
Else /*E satisfies condition 2 above*/
Retrieve the identifier value of E using the identifier value of F from the relationship sets in T in the database.
Insert the above new relationship created into R'.

Algorithm 2.2.1.3 (Insert Type 3 base relationships)
Let R be the insertable view relationship set with derivation $\langle R_1, R_2, \ldots, R_n \rangle$.
Let W be the set of relationship sets obtained in Algorithm 1.2.1 where each relationship set R' in W is functionally equivalent to R w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$ and each of the participating entity types E of R' either
(1) E is the base entity type of some participating view entity types of R
or
(2) E is functionally equivalent to some entity type F w.r.t. a derivation T such that F is the base entity type of some participating view entity type of R and T is $\langle R_j, R_{j+1}, \ldots, R_k \rangle$, $1 \leq j \leq k \leq n$.
For each of the relationship set R' in the set W do
Create a relationship set in R' as follows.
For attributes of R' also found in R, values are assigned as in R;
For attributes of R' not found in R, null values are assigned.
Set Flag = True. /*indicate if we can retrieve the identifier values of participating entity types of R'*/
For each of the participating view entity type E of R' and while Flag = True do
If E satisfies Condition 1 above Then
Identifier value of E is assigned as given in R
Else /*E satisfies condition 2 above*/
Retrieve the identifier value of E using the identifier value of F from the relationship sets in T in the database.
If we cannot retrieve the identifier value of E Then
Set Flag = False.
If Flag is True Then
Insert the above new relationship created into R'.
Else Reject the insertion request.

Example 11. To translate an insertion of a new relationship (a, b, c) into the view relationship set R_v which is Type 1 insertable in Fig. 4, we create the corresponding new relationships (a, b) and (b, c) in R_1 and R_2, respectively, according to Case 1 of Algorithm 2.2.1. To translate an insertion of a new relationship (a, c) into the view relationship set R_w in Fig. 5, we first retrieve the identifier value of the entity type B from the R_1 using the given value a. If R_w is Type 2 insertable, that is, A is mandatory in R_1, then we can always find the B-value b from R_1 given the A-value a and create the corresponding new relationship (b, c) in R_2. Otherwise, if R_w is Type 3 insertable, that is, A is optional in R_1, then whether we can retrieve the B-value from R_1 given the A-value depends on the contents of the database.

Algorithm 2.2.2 (Translate a View Relationship set of modification)
Let R be the view relationship set with the relationship derivation $\langle R_1, R_2, \ldots, R_n \rangle$.
Let S be the set of relationship sets obtained in Step 1 of Algorithm 1.2 where each relationship set in S is functionally equivalent to R w.r.t. $\langle R_1, R_2, \ldots, R_n \rangle$.
Retrieve the corresponding relationships in S using the key value of R.
Modify the values of the identifiers of the participating entity types of these retrieved relationships.
/*Other attributes of the participating entity types are not modifiable*/
For each group of attributes A_g obtained in Step 2 of Algorithm 1.2 (*where the attributes in A_g have the same owner and derivation*) do
Partition A_g into modifiable attributes and not modifiable attributes.
Case 1: Modifiable attributes.
Let A_m be the group of attributes to be modified.
Case 1.1: A_m is a set of inherited attributes.
Modify the values of the base attributes of A_m in the corresponding owner entity using the key value of R.
Case 1.2: A_m is a set of derived attributes with derivation $\langle R_{d_1}, R_{d_2}, \ldots, R_{d_k} \rangle$.
/*Base attributes of A_m are the attributes of a relationship set R_{d_j} in the derivation of A_m, or the identifiers of some participating entity types of R_{d_j}*/
Modify the values of the base attributes of A_m in the corresponding relationship of R_{d_j} using the key value of R.
Case 2: Not modifiable attributes.
If values are given to these attributes. Then
Inform user of error.

8. View implementations in current DBMS systems

The ability to update views is a well-known requirement, and many systems do already support it in some shape or form. However, that support is usually both severely limited and extremely ad hoc in most, if not all, DBMS systems currently available. Typically, selection and projection views are handled by the base DBMS itself, and join views are handled – if they are handled at all – through some kind of frontend system, such as an application generator. In other words, current systems have a built-in understanding of what it means to update a selection or a projection, but must be told explicitly what it means to update a join. They must be told explicitly what it means to update each specific join; it is not possible to tell them once and for all what 'updating a join' means in general terms.

Pre-relational DBMS products were quite weak in their support of views. The CODASYL-proposed DBTG standard of the 1970s support nothing more than those views that just one of the relational operators **project** can generate. IDMS (Integrated Database Management System) is a product of Cullinet Software Inc., and probably the best known example of a network-structured database or DBTG system. The subschema DDL is a language for defining

an external view of the DBTG database. However, the view defined by a subschema is not actually all that different from the underlying schema. The only really significant difference is that certain sets or records or fields can be excluded from the subschema. That is, a subschema is *a simple subset of the schema.*

IBMs IMS (Information Management System) and SYSTEM 2000 DBMS are database systems based on the hierarchical database model. Views in the hierarchical model are equivalent to the declaration of PCB (Program Control Block). A PCB is a subhierarchy of the underlying DBD hierarchy, derived from that underlying hierarchy in accordance with the following three rules:

(1) Any field type can be omitted.
(2) Any segment type can be omitted.
(3) If a given segment type is omitted, then all its children must be omitted too.

Again, we see that an external view in the hierarchical database is a simple subset of the underlying database. IMS also introduced the concepts of physical/logical parent pointer, physical/logical pairs, and secondary indexing which can be used to define significantly different views from the underlying physical structure. The IMS rules for updates in these complex cases are complicated and ad-hoc. For details, the reader is referred to the IMS manuals [26].

The relational model, on the other hand, supports the full power of first-order predicate logic in defining views. The relational DBMS products available today do not yet possess the strength of the relational model in regard to defining views, not to mention updating views. The concept of an external level is available to relational DBMS users through the provision of derived relations. [27] defines a *key-preserving-subset* view as a view which is derived from a single base table by simply eliminating certain rows and certain columns of that table while preserving that table's primary key. Note that although key-preserving-subset views are always theoretically updatable, not all the theoretically updatable views are key-preserving-subset views. INGRES [27], unfortunately has no knowledge or understanding of primary keys. There is thus no chance of INGRESs view-updating mechanism operating in terms of key-preserving-subset views. Instead, it operates in terms of *row-and-column-subset* views. A row-and-column-subset view is a view that is derived from a single base table by simply eliminating certain rows and certain columns of that table. A row-and-column-subset view may or may not be a key-preserving-subset view. More precisely, all key-preserving-subset views are row-and-column-subset views but the converse is not true. *In INGRES, only row-and-column-subset views can be updated.* INGRES is not alone in this regard. Very few products currently support update operations on views that are not row-and-column subsets, and *no* product currently supports update operations on all views that are theoretically updatable. DB-2 is another example [28].

REQUIEM [29] is a relational database system implemented on the Unix operation system and currently runs on the Sun workstations. It allows the definition of two types of views, those which are updatable, and those which are not. In REQUIEM, a view V is updatable, that is, one can perform insertions, deletions or modifications in its context, if and only if **all** the following conditions apply to the view context.

(1) V is made up from only one base relation, R.
(2) There exists a one-to-one correspondence between the tuples of V and those of R, such that any modification on V results into an equivalent operation on R.

VAX Rdb/VMS [30] is yet another relational database system implemented by Digital Equipment Corporation on the VAX/VMS systems. The user needs to take note of the following when using INSERT, DELETE, or UPDATE statements that refer to views in VAX Rdb/VMS.

(1) Do not refer to read-only views in INSERT, DELETE, or UPDATE statements. SQL considers as read-only views those with select expressions that:
   * Use the DISTINCT argument to eliminate duplicate rows from the result table.
   * Name more than one table or view in the FROM clause.
   * Specify a subquery in the predicate of the WHERE clause.
   * Include a function in the select list.
   * Contain a GROUP BY, HAVING or ORDER BY clause.

(2) In the INSERT and UPDATE statements, the user cannot refer to columns in views that are the result of an arithmetic expression or a function.

(3) The user needs to use the WITH CHECK OPTION clause to make sure that rows inserted or updated in a view conform to its definition. Omitting the WITH CHECK OPTION clause allows the user to insert or update rows through a view that do not conform to the view's definition. Once stored, however, the user cannot retrieve these rows through the view, because they do not meet the conditions specified by the view definition.

The above conditions do not support updates on views involving joins. Furthermore, to allow updates which violates the view's definition, and later disallow the user to retrieve the new updated data through the view because they do not meet the conditions specified by the view definition is not being consistent.

9. Conclusions

We can see that the implementations of view updates in current database management systems are severely limited and ad-hoc in complex cases. All of them do not support views involving joins nor do they have the concept of identifiers or primary keys. Note that the one essential information for determining the updatability of relations is functional dependencies. However, none of the relational DBMS can handle functional dependencies in their DDL. On the other hand, we have presented a systematic approach to solve the problem of view update in ER based DBMS where views are modelled by ER diagrams. We have seen that views on ER approach are not necessary flat relations, but can be nested. The entity types in a relationship set predicate are complex objects. Any multivalued and weak entity types are sets in the owner predicate. As a result, view update in the ER approach is different from that in the relational model. However, the results of evaluating view updatability in the ER approach can be applied to the relational model as the latter is a special case of the ER approach.

We have developed a theory within the framework of the ER approach that characterizes the conditions under which there exist mappings from view updates for view entity types and view relationship sets into updates on the conceptual schema. We allowed the concept of virtual updates which are carried out by the system to ensure that changes in a view requested are consistent with the rest of the database. This is important in cases where the value of a view attribute cannot be changed by the user but whose value is a function of the values of

other modifiable view attributes. With the concept of derivations, we are able to handle view updates involving derived attributes, relationship set joins and multilevel inheritances through the special relationship sets ISA, UNION, etc. We have also defined three types of insertability for view relationship sets. We can always find the mapping to translate any insertion requests on Type 1 insertable view relationship sets. If a view relationship set is Type 2 insertable, then any view relationship insertion request is subjected to domain and key constraint checks. On the other hand, if a view relationship set is Type 3 insertable, then any view relationship insertion request is not only subjected to domain and key constraint checks, but is also dependent on the contents of the database. We have also seen that if a view relationship set is Type 1 insertable, then it is also Type 2 insertable. If a view relationship set is Type 2 insertable, then it is also Type 3 insertable. Moreover, we proved that if a view relationship set is not Type 3 insertable, then it is not insertable.

Based on the theory, we have developed the View Updatability Algorithm and the View Update Translation Algorithm. These algorithms also take into consideration the three types of insertability for view relationship sets. Ling [2] has an algorithm which gives a unique translation of a normal form ER diagram to a set of relations. Hence, any update in the ER approach can be translated uniquely to an equivalent update in the relational database. Note that our approach to view update is intended to fit into the framework of a general and systematic approach to the whole question of view updating.

References

[1] T.W. Ling, A three level schema architecture ER based database management systems, in: S.T. March, ed., *Entity-Relationship Approach* (North Holland, Amsterdam, 1987) 205–220.

[2] T.W. Ling, A normal form for Entity-Relationship diagrams, *Proc. 4th Int. Conf. on Entity-Relationship Approach* (1985).

[3] T.W. Ling and M.L. Lee, A prolog implementation of an ER based DBMS, *Proc. 10th Int. Conf. on ER Approach* (1991) 587–605.

[4] E.F. Codd, *Recent Investigations in a Relational Database System*, *Information Processing 74* (North-Holland, Amsterdam, 1974) 1017–1021.

[5] F. Bancilhon and N. Spyratos, Update semantics and relational views, *ACM Trans. Database Systems* 6(4) (1981).

[6] T. Barsalou et al., Updating relational databases through object-based views, *Proc. 1991 ACM SIGMOD Int. Conf. on Management of Data* (May 1991).

[7] C.R. Carlson and A.K. Arora, The updatability of relational views based on functional dependencies, *3rd Int. Computer Software and Applications Conf.*, IEEE Computer Society (1979).

[8] M.C. Chan, Translation templates for updates issued on relation views, Tech. Report 35, Dept. of Comp. Science, Monash University, Melbourne, Australia, April 1983.

[9] M.C. Chan and K.J. McDonell, An update schema for relational views, *Proc. 10th Aust. Computer Conf.* (September 1983) 150–163.

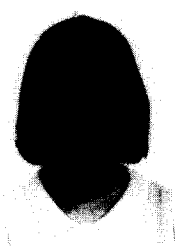
[10] B. Czejdo, D.W. Embley and M. Rusinkiewicz, View updates for an extended Entity-Relationship Model, *J. Information Sciences 62* (1992) 41–62.

[11] U. Dayal and P.A. Bernstein, On the correct translation of update operations on relational views, *ACM Trans. Database Systems* 7(3) (1982).

[12] A.L. Furtado, C.K. Sevcik and C.S. Santos, Permitting updates through views of databases, *Information Systems* 4(4) (Pergamon Press, UK, 1979),

[13] J. Guttag, Abstract data types and the development of data structures, *Commun. ACM* 20(6) (1977) 396–404.
[14] A.M. Keller, Algorithms for translating view updates to database updates for views involving selections, projections and joins, *4th PODS*, ACM (March 1985).
[15] A.M. Keller, Choosing a view update translator by Dialog at view definition time, *Proc. 12th Int. Conf. on Very Large Databases* (1986).
[16] R. Langerak, View updates in relational databases with an independent scheme, *ACM Trans. Database Systems 15*(1) (March 1990) 40–66.
[17] S.B. Legg and K.J. McDonell, Translating update requests on user views, Technical Report 77, Department of Computer Science, Monash University, Melbourne, Australia, Nov. 1986.
[18] L. Rowe and K.A. Schoens, Data abstractions, views and updates in RIGEL, in: *Proc. ACM-SIGMOD Int. Conf. on Management of Data* (1979) 71–81.
[19] K.C. Sevcik and A.L. Furtado, Complete and compatible sets of update operations, in: *Int. Conf. on Management of Data*, ICMOD (1978).
[20] M. Stonebraker, Implementation of integrity constraints and views by query modification, *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, San Jose (1975) 65–78.
[21] P.P. Chen, The Entity-Relationship model: Toward a unified view of data, *ACM Trans. Database Systems 1*(1) (1976) 166–192.
[22] M.L. Lee, An Entity-Relationship based database management system, a thesis submitted for the degree of Master of Science, National University of Singapore, 1992.
[23] T.W. Ling and M.L. Lee, A graphical Entity-Relationship based database management system workbench, *Proc. 4th Int. Workshop on Computer-Aided Software Engineering* (1990) 480–495.
[24] J. Grant and T.W. Ling, Database representation and manipulation using Entity-Relationship database logic, *Proc. of Methodologies for Intelligent Systems IV* (Elsevier Science Pub. Co, 1989) 102–109.
[25] D. Maier, *Theory of Relational Databases* (Computer Science Press, Rockville, MD, 1983).
[26] D. Kapp and J.F. Leben, *IMS Programming Techniques: A Guide to Using DL/I* (Van Nostrand Reinhold, New York, 1978).
[27] C.J. Date, *A Guide to INGRES* (Addison-Wesley, 1987).
[28] C.J. Date, *An Introduction to Database Systems*, 4th edition (Addison-Wesley, 1986).
[29] M. Papazoglou and W. Valder, *Relational Database Management: A Systems Programming Approach* (Prentice-Hall, 1989).
[30] Digital Equipment Corporation, VAX Rdb/VMS Reference Manual, 1989.

Tok Wang Ling is an Associate Professor of the Department of Information Systems and Computer Science at the National University of Singapore. He received his Ph.D. and M.Math., both in Computer Science, from Waterloo University (Canada) and B.Sc. in Mathematics from Nanyang University (Singapore).
His research interests include Data Modeling, Entity-Relationship Approach, Object-Oriented Data Model, Normalization Theory, Logic and Database, Integrity Constraint Checking.
He is a member of ACM, IEEE, and Singapore Computer Society.



Mong Li Lee received her B.Sc. (Hons 1) and M.Sc. degrees in Computer Science from the National University of Singapore in 1989 and 1992, respectively. She is a Senior Tutor in the Department of Information Systems and Computer Science, National University of Singapore. She is also currently pursuing her doctorate degree from the same department.
Her research interests include Entity-Relationship Approach, Logic and Databases, View Updates, Graphical User Interface, Database Schema Translation and Integration, and Heterogenous Databases.
She is a member of ACM.