A Theory for Entity-Relationship View Updates

Tok-Wang Ling   Mong-Li Lee

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

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 [17] where the conceptual schema is represented by a normal form ER diagram [16] 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 conceptual schema updates. Concepts such as virtual updates and three types of insertability are introduced.

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.

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. [19] describes 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 conceptual schema. However, such a mapping does not always exist, and even when it does exist, it may not be unique [6]. The problem of updating relational databases through views is an important practical problem that has attracted much interest [1, 2, 3, 7, 8, 10, 11, 12, 13, 15, 23]. The user specifies queries to be executed against the database view; these queries are translated to queries against the underlying database through query modification [24]. 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. In current practice, updates must be specified against the underlying database rather than against the view. This is because the problem of updating relational databases through views is inherently ambiguous [11]. 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 [10]; 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 [22, 23]. 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 [2, 4, 7, 11, 13, 15]. 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 conceptual schema updates satisfying some desired properties. [7] develops a theory within the framework of the relational model that characterizes precisely the conditions under which there exist mappings from view updates into conceptual schema updates satisfying various properties. He formalize 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 updates 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.

[11] analyses the possible translations of particular classes of update operations for relational views and obtains the semantics of the application to choose among the alternative translations from a dialog with the database administrator at view-object definition time. However, Keller's update policy of translating deletion or insertion against a selection view into a modification of the operand view or base relation has some problems. 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'
```

[12] proposes 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 has been a lack of literature in the area of view update for the ER approach. The problem of view update in the ER approach is quite different from that in relational databases as views in ER approach are not necessarily flat relations. Furthermore, the ER approach uses the concepts of entity types and relationship sets and incorporates some of the important semantics about the real world which helps us in resolving ambiguity when translating view updates. For instance, the special relationship sets such as ISA, UNION etc in the ER approach reflects inheritance in the real world. In this paper, we examine the problem of view update in Entity-Relationship based database management systems [17] (which is quite different from relational databases) where views may be modelled by ER diagrams. Section 2 gives the terminologies used in this paper. Section 3 explains what is meant by view updatability in ER approach. We develop a theory within the framework of the ER approach that characterizes the conditions under which there exist mappings from view updates into conceptual schema updates in section 4.

2. Terminologies

[5] proposes the ER approach for database schema design. It uses the concepts of *entity type* and *relationship set.* An entity type or relationship set has *attributes* which represent its structural properties. An attribute can be *single-valued, multivalued or composite.* A minimal set of attributes of an entity type E which uniquely identifies E is called a *key* of E. An entity type may have more than one key and we designate one of them as the *identifier* of the entity type. A minimal set of identifiers of some entity types participating in a relationship set R which uniquely identifies R is called a key of R. A relationship set may have more than one key and we designate one of them as the identifier of the relationship set. Note that there are entity types in which entities cannot be identified by the values of its own attributes, but has to be identified by its relationship with other entities. Such an entity type is called a *weak entity type* and the relationship set which is used to identify the entity is said to be an *identifier dependent relationship set.* If the existence of an entity in one entity type depends upon the existence of a specific entity in another entity type, such a relationship set and entity type are called *existence dependent relationship set* and weak entity type. An entity type which is not a weak entity type is called a *regular entity type.* A relationship set which involves weak entity type(s) is called a *weak relationship set.* A relationship set which does not involve weak entity types is called a *regular relationship set.* In the ER approach, *recursive* relationship sets and *weak* relationship sets such as *existence dependent* (EX) and *identifier dependent* (ID) relationship sets are allowed. We can also have special relationship sets such as *ISA, UNION, INTERSECT* etc. For more details, see [16].

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*. There is a one-to-one correspondence between the entities of a view entity type and the entities of some entity type which is called the *base entity type* of the view entity type, in the conceptual schema. 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 [17].

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 define a *derivation* as a list of conceptual schema relationship sets which are involved in natural joins to obtain a view attribute. If a view attribute A has a *derivation* $<R_1, R_2, ..., R_n>$, where R_i is a relationship set in the conceptual schema, $1 <= i <= n$, then we call A a *derived attribute*. 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* from by joining all the relationship sets in the attribute derivation. A derivation also specifies how a view relationship set is obtained from the relationship sets in the conceptual schema. A special case of derived attributes occurs if the derivation of a view attribute A contains only special relationship sets. We call such attributes *inherited attributes*. *Multi-level 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 [14].

[14] proposes an ER schema and view data definition language. Figure 2 shows an ER external view which is based on the example medical database in Figure 1. We illustrate the view definition obtained during the construction of this external view in an ER based DBMS Workbench [18]. 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. The view definition for Figure 2 is as follows. The keywords are in italics.

```
VIEW DOCTPAT OF MEDICALDB
     VIEW ENTITY TYPE EMPLOYEE  /*By default, base and view entity types have same name*/
          ( ATTRIBUTES ( EMPNO,   /*Base attribute 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 indicates 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> ) )
```

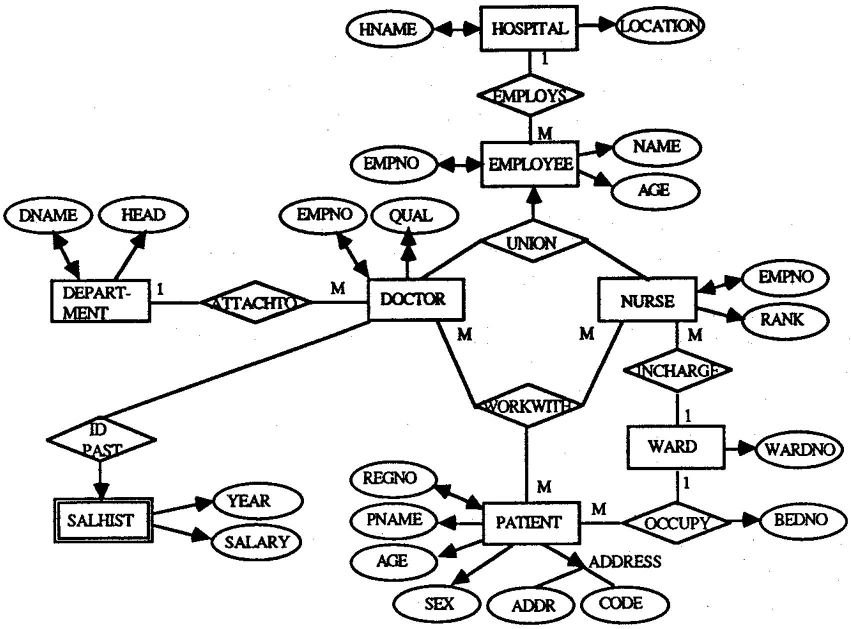

Figure 1: An Example ER Medical Database

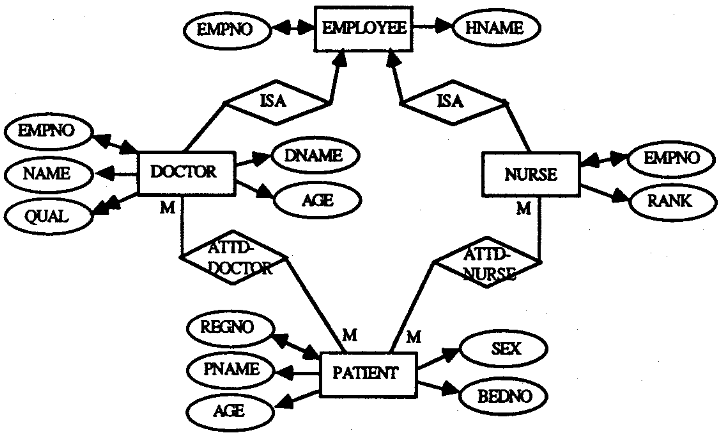

Figure 2: An Example ER External View of Conceptual Schema in Figure 1.

3. 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 [9]. Using Figure 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 multi-valued 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 identifer 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. No update is allowed on the non-identifier attributes of the participating entity types. The insertion of a relationship requires the identifer values of all the participating entity types to be defined. It violates the meaning of a relationship set in an ER database if we allow insertion to occur when only the identifier of a relationship defined.
3. *Multivalued Attributes and Weak Entity Types*
   Weak entity types are set-valued attributes in the parent entity type predicate. Multivalued attributes are also sets 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 transfered to the pediatrics department, we can have the following Prolog goal to update the view entity type DOCTOR in Figure 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.

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. That is, there must *no ambiguity of origin* in the view objects.
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. 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 in a view requested are consistent with the rest of the database are permitted. The following definition introduce 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.
2. Let E be the base entity type of a view entity type E'. Suppose E' contains attributes A_1, A_2, ...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, ..., 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 modifiable in the view entity type E', then the modification of A_1 will cause the system to retrieve the corresponding values of the attributes A_2, ...A_k whenever these values are required.
3. Let $<R_1, R_2, ..., R_n>$ be the derivation of a view relationship set R'. Suppose R' contains attributes A_1, A_2, ...A_k whose base attributes are in an entity type E, where E is a participating entity type in some relationship set in the derivation, say R_i, for some i where $1 <= i <= n$. If the base attributes 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, ...A_k whenever these values are required.

4. 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 conceptual schema updates. 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 scction 3.

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 : Trivial. ■

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 [21].

Definition 3 : Two sets of entity types E_i and E_j are *functionally equivalent* w.r.t. a derivation $<R_1, R_2, ..., R_n>$, denoted $E_i \leftrightarrow E_j$, if and only if we can establish that the set of identifiers of the entity types in E_i is functionally equivalent to the set of identifiers of the entity types in E_j from the functional dependencies in the relationship sets $R_1, R_2, ..., R_n$.

Figure 3 shows an ER diagram in which A is functionally equivalent to B w.r.t. $<R_1>$, but $A \rightarrow B$, $B \not\rightarrow A$ w.r.t. $<R_2>$. Since $B \leftrightarrow C$ in R_3, we can conclude that $A \leftrightarrow C$ from the functional dependencies in $<R_1, R_3>$ (or we can say that $A \leftrightarrow C$ w.r.t. $<R_1, R_3>$) by transitivity.

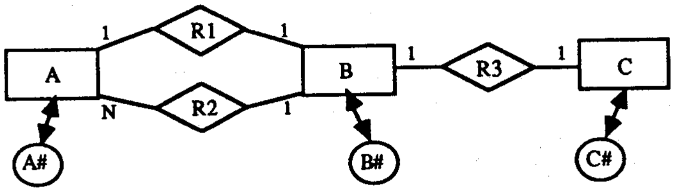

Figure 3: An ER diagram to illustrate the functional equivalance of entity types.

Definition 4 : Let E_i be the set of participating entity types of a relationship set R_i whose identifiers form a key of R_i. Similarly, let E_j be the set of participating entity types of another relationship set R_j whose identifiers form a key of R_j. We say that

R_i and R_j are *functionally equivalent*, denoted $R_i \leftrightarrow R_j$, w.r.t. a derivation $<R_1, R_2, ..., R_n>$ if and only if E_i and E_j are functionally equivalent w.r.t. $<R_1, R_2, ..., R_n>$.

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

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. $<R_1, R_2, ..., R_n>$, where $i \in \{1, 2, ..., n\}$, 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_j w.r.t. $<R_1, R_2, ..., R_n>$, where $j \in \{1, 2, ..., n\}$, then there will not be a one-to-one correspondence between the relationships of R and the relationships of R_j. R_j 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. ■

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 derived attributes.

For example, the single-valued derived attribute DNAME in the view entity type DOCTOR in Figure 2 can be modified as follows. We observe that the base attribute of the key of the view entity type DOCTOR and the key of the conceptual schema relationship set ATTACHTO are functionally equivalent w.r.t. <ATTACHTO>, thus resulting in a one-to-one correspondence between the view entities in DOCTOR and the relationships in ATTACHTO. Hence, when we modify the value of the derived attribute 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.

We generalize this concept of modifying single-valued 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 $<R_1, R_2, ..., R_n>$. If the base attribute of A is the identifier of some 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. $<R_1, R_2, ..., R_n>$.
Proof : The derived 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 $R_1, R_2, ..., R_n$ 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. $<R_1, R_2, ..., R_n>$. If the base attribute of A is an identifier of F, then it is part of the relationship set R_n. If A is a single-valued attribute in E', then 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. $<R_1, R_2, ..., R_n>$. ■

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.

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 $<R_1, R_2, ..., R_n>$. If the base attribute of A is an attribute of 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. $<R_1, R_2, ..., R_n>$. ■

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 3. 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.

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 : Trivial. ■

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

Theorem 5 : Let R be a view relationship set with relationship derivation $<R_1, R_2, ..., R_n>$. R is insertable for those attributes which are also the attributes of some relationship set R_i where $i \in \{1, 2, ..., n\}$ if R is functionally equivalent to R_i w.r.t.

$<R_1, R_2, ..., R_n>$ and all the participating entity types of R_i are also the base entity types of the participating view entity types of R.
Proof : If R is functionally equivalent to some relationship set R_i w.r.t. $<R_1, R_2, ..., R_n>$, 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, we can only insert a new relationship into R 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 base relationship set is always updatable. ■

We refer to the class of view relationship sets that are determined to be insertable by theorem 5 as *Type 1 insertable*. We can always find the mapping to translate any insertion requests on these Type 1 insertable relationship sets. For example, Figure 4 shows a view relationship set R_v obtained from a join of two conceptual schema relationship sets R_1 and R_2, that is, derivation is $<R_1, R_2>$. A', B' and C' are the view participating entity types of R_v whose base entity types are A, B and C respectively. R_v is functionally equivalent to both R_1 and R_2. Hence, R_v is Type 1 insertable with respect to both R_1 and R_2. 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.

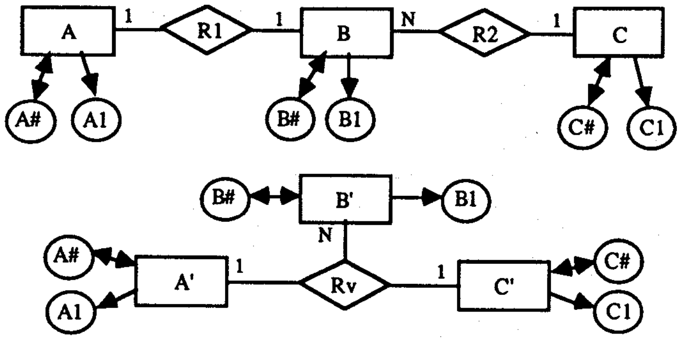

Figure 4: A view relationship set Rv obtained from a join of the conceptual schema relationship sets R1 and R2.

However, the class of view relationship sets which are Type 1 insertable is very restrictive. For example, Figure 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 Figure 4, that is, derivation is $<R_1, R_2>$. Here, the common entity type B has been projected out from the view. Although R_w is not Type 1 insertable, but 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.

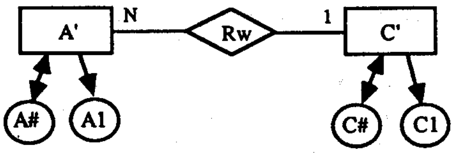

Figure 5: A view relationship set Rw obtained from a join of the conceptual schema relationship sets R1 and R2 with the common entity type B projected out.

We have a few observations from the second example.

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 is 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 b 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 Type 1 insertable according to theorem 5, but 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, but the base entity type A of A' is functionally equivalent to B w.r.t. $<R_1>$. R_w is *Type 3 insertable* by the definition following theorem 6.
3. In the ER approach, the existence of an entity in a relationship could be defined as either *mandatory or optional*. 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. R_w is *Type 2 insertable* by the definition following theorem 6.

Definition 5 : Suppose an entity type E_{i0} is involved in a relationship set R_{i0} with another entity type E_{i1}, and E_{i1} is involved in a relationship set R_{i1} with an entity type E_{i_2}, and so on, and eventually we have an entity type $E_{i_{j-1}}$ involved in a relationship set $R_{i_{j-1}}$ with an entity type E_{i_j}. If the existence of E_{i_k} is mandatory in R_{i_k} (which may be n-ary) for all k, $0 <= k < j$, then we say that the existence of E_{i_0} is *transitively mandatory* in the relationship R which is obtained from a natural join of all the relationship sets R_{i_k}.

We will now define the concepts of Type 2 insertable and Type 3 insertable formally.

Theorem 6 : Let R be a view relationship set with the relationship derivation $<R_1, R_2, ..., R_n>$. R is insertable for those attributes which are also the attributes of some relationship set R_i where $i \in \{1, 2, ..., n\}$, if R is functionally equivalent to R_i w.r.t. $<R_1, R_2, ..., R_n>$, 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 either $<R_1, R_2, ..., R_{i-1}>$ or $<R_{i+1}, R_{i+2}, ..., R_n>$. ■

We call the class of view relationship sets that are determined to be insertable by the above theorem as *Type 3 insertable*. Moreover, if the existence of the entity type F in the above theorem is transitively mandatory in the relationship set which is obtained from a join of a set of relationship sets in the derivation T, then we call this class of view relationship sets as *Type 2 insertable*. For example, the view relationship set R_w in Figure 4 has a relationship derivation $<R_1, R_2>$ and the entity type A is functionally equivalent to B w.r.t. $<R_1>$. R_w is Type 2 insertable if A is mandatory in R_1. Otherwise, R_w is Type 3 insertable. In both cases, R_2 is the base relationship set of R_w, that is, R_w is insertable w.r.t. R_2. Note that if a view relationship set is Type 2 insertable, then any relationship insertion request is subjected only to domain and key constraint checks. On the other hand, if a view relationship set is Type 3 insertable, then any relationship insertion request is not only subjected to domain and key constraint checks, but is also dependent on the contents of the database.

Corollary 6: 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. ■

We conclude in the following theorem that if a view relationship set is not Type 3 insertable, then it is not insertable.

Theorem 7 : If a view relationship set is not Type 3 insertable, then it is *not* insertable.
Proof : We will give an outline of the proof here.
Let R be a view relationship set with the relationship derivation $<R_1, R_2, ..., R_n>$. If R is not Type 3 insertable, then by theorem 6, for each of the relationship sets R_i, $1 <= i <= n$, either R is not functionally equivalent to R_i w.r.t. $<R_1, R_2, ..., R_n>$, or there exists some participating entity type E of R_i such that E is not the base

entity type of any participating view entity type of R, and E is not functionally equivalent to any entity type F w.r.t. derivation T such that F is the base entity type of some participating view entity type of R and T is either $<R_1, R_2, ..., R_{i-1}>$ or $<R_{i+1}, R_{i+2}, ..., R_n>$.
Now if R is not functionally equivalent to R_i w.r.t. $<R_1, R_2, ..., R_n>$, 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 the base relationship set of R.
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 either $<R_1, R_2, ..., R_{i-1}>$ or $<R_{i+1}, R_{i+2}, ..., R_n>$, then there is no way we can obtain the identifier value of E during an insertion of R. Therefore, R_i is not the base relationship set of R.
Hence, if R is not Type 3 insertable, then for each of the relationship sets R_i, $1 <= i <= n$, R_i is not the base relationship set of R. Therefore, 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, 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.

For example, we may want to insert a new doctor into the view entity type DOCTOR in Figure 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. We have a one-to-one correspondence between the relationships in ATTACHTO and the entities in the view entity type DOCTOR since the identifier of ATTACHTO is functionally equivalent to the key of 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. 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 $<R_1, R_2, ..., R_n>$ 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. the derived relationship set R of A is functionally equivalent to R_n w.r.t $<R_1, R_2, ..., R_n>$, and
2. E is functionally equivalent to E_n w.r.t. $<R_1, R_2, ..., R_{n-1}>$, and
3. E is transitively mandatory in the relationship set which is obtained from a join of the relationship sets $R_1, R_2, ..., R_{n-1}$.

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 $R_1, R_2, ..., R_n$ 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. $<R_1, R_2, ..., R_n>$. If E is functionally equivalent to E_n w.r.t. $<R_1, R_2, ..., R_{n-1}>$, and E is transitively mandatory in the relationship set obtained from a join of the relationship sets $R_1, R_2, ..., R_{n-1}$, then R is Type 2 insertable with respect to R_n. 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 a 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 7: 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. ■

Corollary 8: An inherited attribute is insertable. ■

Based on the above theory developed, we have an algorithm to systematically determine the updatability of view entity types and view relationship sets in a view. In addition, this algorithm also determines the different types of insertability for view relationship sets. Interested readers can refer to [20] for details of this View Updatability Algorithm. We also have a View Update Translation Algorithm [20] 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 these information during any view update request translation.

We have seen that it is trivial to delete an entity from a deletable view entity type. However, to insert a new entity into an insertable view entity type, we may 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 a corresponding relationship into some relationship set. For example, to insert a new doctor into the view in Figure 2, we have the Prolog goal

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

The View Update Translation Algorithm will translate this view insertion request into the following three facts to be inserted into the database.

```
doctor (116790, ['MBBS', 'MMed']).
        /* Base attributes of EMPNO and QUAL are in base entity type */
employee (116790, 'H. Goh', 35).
        /* NAME & AGE are inherited attributes with derivation <UNION>*/
attachto (116790, surgery).
        /* DNAME is a derived attribute with derivation <ATTACHTO> */
```

To modify a particular doctor in the view entity type DOCTOR in Figure 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. If the attribute DNAME is given a new value, we modify the corresponding attachto relationship. Similar forms of translations can be carried out for view update requests on relationship sets. View relationship sets deletions and modifications are trivial. For view relationship set insertions, we need to consider the three types of insertability.

5. Conclusions

We have proposed a theory within the framework of ER approach which characterizes the conditions under which there exist mappings from view updates into conceptual schema updates. 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. [16] 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. F. Bancilhon and N. Spyratos: Update semantics and relational views, ACM Trans. Database Systems 6 (4), 1981.
2. T. Barsalou, et. al: Updating Relational Databases through Object-Based Views, Proc. of the 1991 ACM SIGMOD Int. Conf. on Management of Data, May 1991.
3. C.R. Carlson and A.K. Arora: The updatability of relational views based on functional dependencies, Third International Conputer Software and Applications Conference, IEEE Computer Society, 1979.
4. M.C. Chan: Translation templates for updates issued on relation views, Tech. Report 35, Dept. of Comp. Science, Monash University, Melbourne, Australia, April 1983.
5. P.P. Chen: The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems vol 1, no 1, 1976, pp 166-192.
6. E.F. Codd: Recent Investigations in a Relational Database System, Information Processing 74, North Holland, Amsterdam, 1974, pp 1017-1021.
7. U. Dayal and P.A. Bernstein: On the correct translation of update operations on relational views, ACM Trans. Database Systems 7 (3), 1982.
8. A.L. Furtado, C.K. Sevcik and C.S. Santos: Permittting updates through views of databases, Information Systems 4 (4), Pergamon Press, Great Britain, 1979.
9. J. Grant and T.W. Ling: Database Representation and Manipulation Using Entity-Relationship Database Logic, Proc. of Methodologies for Intelligient Ststem IV, Elsevier Science Pub. Co., 1989, pp 102-109.
10. J. Guttag: Abstract data types and the development of data structures, Communications of ACM 20 (6), 1977, pp 396-404.
11. A.M. Keller: Algorithms for translating view updates to database updates for views involving selections, projections and joins, 4th PODS, ACM, March 1985.
12. A.M. Keller: Choosing a view update translator by Dialog at view definition time, Proc. of the 12th International Conference on Very Large Databases, 1986.
13. R. Langerak: View Updates in Relational Databases with an Independent Scheme, ACM Transactions on Database Systems, Vol 15, No 1, March 1990, pp 40-66.
14. 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.
15. 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.
16. T.W. Ling: A Normal Form for Entity-Relationship Diagrams, Proc. 4th International Conference on Entity-Relationship Approach, 1985.

17. T.W. Ling: A Three Level Schema Architecture ER based Database Management Systems, in: March, S.T. (ed), Entity-Relationship Approach, North Holland, Amsterdam, 1987, pp 205-220.
18. T.W. Ling and M.L. Lee: A Graphical Entity-Relationship Based Database Management System Workbench, Proc. 4th International Workshop on Computer-Aided Software Engineering, 1990, pp 480-495.
19. T.W. Ling. and M.L. Lee: A Prolog Implementation of an ER based DBMS, Proc. 10th Int. Conf. on ER Approach, 1991, pp 587-605.
20. T.W. Ling and M.L. Lee: View Update in Entity-Relationship Approach, to be submitted for publication, 1992.
21. D. Maier: Theory of Relational Databases, Computer Science Press, 1983.
22. L. Rowe and K.A. Schoens: Data abstractions, views and updates in RIGEL, in Proc. ACM-SIGMOD International Conf. on Management of Data, 1979, pp 71-81.
23. K.C. Sevcik and A.L. Furtado: Complete and compatible sets of update operations, in International Conf. on Management of Data (ICMOD), 1978.
24. M. Stonebraker: Implementation of integrity constraints and views by query modification, Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose, 1975, pp 65-78.