# RELATIONSHIP MERGING IN SCHEMA INTEGRATION

S. B. Navathe(1)*, T. Sashidhar(1),
R. Elmasri(2)*

(1)University of Florida (2)University of Houston

## Abstract

Merging of relationships among data is an important activity in schema integration. The latter can arise as integration of user views in logical database design or as the creation of a global schema from existing databases in a distributed or centralized environment. During the "view integration" phase of design, separate views of data held by different user groups are integrated into a single conceptual schema for the entire organization. In this paper we use a variant of the entity relationship model to represent schemas or user views and discuss the problem of integrating relationships from different schemas. Using three major criteria for comparing relationships, we develop a hierarchical comparison scheme. Each case represented by the terminal nodes of this hierarchy is discussed separately and rules of integration are developed. The problem is dealt with in a general sense so that the qualitative discussion is applicable to several other semantic data models. After a paper on object class integration at COMPDEC 84, this work constitutes our next step in the research on schema integration.

## 1.0 INTRODUCTION

With the diversity of data models and database management systems, the problem of schema integration is becoming increasingly important. Schema integration arises in two different contexts:

a) Global schema design.

b) Logical database design.

In global schema design, several databases already exist and are in use. The objective is to design a single global schema which represents the contents of all these databases. This global schema can then be used as an interface to the diverse databases. User queries and transactions can then be specified against this global schema, and the requests are mapped to the relevant databases. The schema integration is applicable for global schema design in both centralized and distributed environments. We do not make a specific reference to global schema design aspect of integration in the rest of this paper; rather, we focus on the use of schema integration in logical database design. The reader should keep in mind that the overall philosophy of schema integration discussed here applies to both.

Database design has been described as an art rather than a science. Conventional database design is heavily dependent upon the designer's experience and intuition. The sophistication of the applications of databases has advanced tremendously but the development of techniques to support database design has not advanced comparably beyond what we found in 1978 [NYU78]. Database design as mentioned in the 1978 New Orleans Data Base Design Workshop [LUM79] is still "an art practiced by few with few tools other than the designer's experience and intuition". In the past few years attempts have been made to treat database design in a scientific and structured manner. There have essentially been two schools of thought:

A. In the first, the database design starts with the individual data items. All functional and other types of dependencies among individual data items must be specified before constructing the schema. The relational synthesis approach [BERN76], approaches using the functional data model [YAO82] and formal approaches using various types of dependencies [CASA83] fall in this category. This method of design is appealing since it appears "foolproof". However as argued by Bubenko [BUBE79], the assumptions of this approach are "not always realistic". Moreover, the approach generally works with the idea that the minimal set of relations is the best database design. Semantic considerations are almost completely ignored. In addition, the resulting schema is dependent on the order in which data dependencies may be given to the synthesis algorithms.

*Work partially supported by Honeywell Computer Sciences Center, Bloomington, Minnesota.

The most serious drawback of this method is its user friendliness. The users (as well as designers) of databases in the real world of commercial, industrial and government applications tend to be people with only average skills in mathematics or database models. The initial input in terms of an array of functional/partial/conditional relationships is very hard to obtain for large databases used in practice simply "by interviewing the users" as is normally claimed.

B. The second school of thought advocates operation at the level of entities, relationships and attributes. Following this approach one divides the database design process into the following four phases:

I. View modeling.

II. View integration.

III. Schema analysis and mapping.

IV. Physical schema design and optimization.

The use of the word "view" needs some elaboration. This term has been used in the database design context to represent the view of data held by a user group or a designer of the entire enterprise. Terms such as user view or enterprise view make it clearer as to who holds or owns the view. Two aspects are covered by this term:

- structure of data.
- processing of data.

In this paper, view will be used to refer to structure only. The transactions on views are considered during the analysis and optimization phases. (Note: this use of the term view is not to be confused with external views which are supported in the form of external schema definition in some DBMS like System R).

During view modeling, the users' views of data are expressed in a high level semantic model such as the E-R model or a variant. All the user views are integrated during the view integration phase to generate the schema. The work of Navathe, Elmasri and Wiederhold [NAVA78, WIED79, ELMA79, NAVA82, ELMA84], Batini and Lenzerini et al. [BATI82, BATI83, BATI84, ATZ81] falls into this category. View integration as a process has been further analyzed in [NAVA82, BATI83]. The overall integration is looked upon as an incremental process of integration and conflict resolution. This school is of the opinion that conflict resolution in view integration can be accomplished by interactive design tools. The integration is the result of a constant dialogue with the user about conflict resolution.

Three distinguishing features of this school of thought from the previous one are as follows:

A. In this approach the semantics of the objects and of relationships among objects drives the integration. The methodology tries to reach a global representation of the data which is the "best compromise" for the given set of local views considering all the conflicts. The 'semantics' we are referring to are over and above functional dependencies which are typically included in the other school. The approaches driven purely by dependencies cannot have any "compromise" notion.

B. The methodology relies heavily upon designer's intervention. The first school of thought expects the design activity to be fully automated whereas the latter assumes that the semi-automated approach should help the designer by identifying the conflicts and proposing alternatives to merging. An automatic merging of views is considered not only unrealistic but also meaningless in the absence of semantic information that integrates multiple views of data.

C. The design optimization under the second school of thought is typically based on an evaluation of alternative designs in the context of the given processing load. Approaches based on pure dependency information [CASA83] generally assume that minimal redundancy is the primary objective of design. A few functional model advocates, e.g. Yao et al. [YAO82], do however use the notion of transaction in their method. In the present paper we have not included transaction specification at all since we are not addressing design optimization.

### 1.1 OBJECTIVE AND SCOPE

This paper addresses the view integration phase of database design under the second school of thought. [BATI82, BATI84] have proposed extensions to the E-R model to facilitate view integration. They have considered the naming problems (homonyms & synonyms) in detail and have approached view integration as an incremental modification of the nucleus of a global view to accommodate all otherviews considered one by one. They do not fully consider a global analysis of all views or the merger of a class of views. Elmasri and Wiederhold [ELMA79] specify different types of binary relationships among object classes and consider how they can be made to coexist. They

do not consider relationships of higher degree. By using a variant of the E-R model called the Entity-Category-Relationship model [WEEL80, ELMA84], we shall specifically focus on the problem of relationship integration. We believe that neither the above two referenced groups of works nor any others have addressed the problem in a general sense as we do in this paper. In particular, integration of relationships considering cardinality ratios and roles explicitly has not been dealt with before.

### 1.2 ENTITY-CATEGORY-RELATIONSHIP MODEL REVIEW:

The extended E-R model we use here is called the Entity-Category-Relationship (E-C-R) model. The E-C-R model includes extensions to the E-R model in two main areas:

1. The category concept is used to represent sub-classes [HAMM81]. Besides, categories can also be used to group entities playing the same role in a relationship.

2. Cardinality and dependency constraints on relationships are specified precisely.

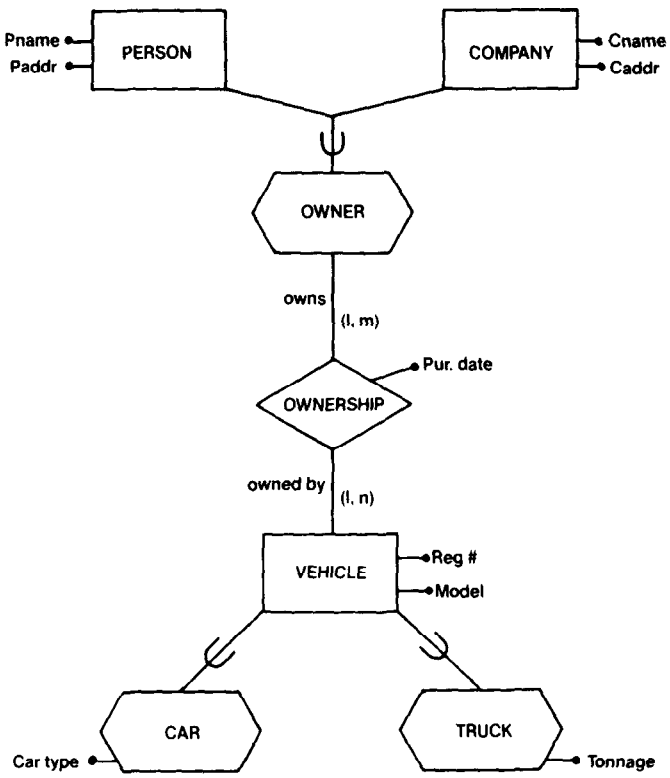



Figure 1.

The E-C-R model uses the following constructs: entity sets, relationship sets, categories and attributes. Entity sets represent sets of entities that have the same attributes. Categories represent additional groupings of entities from one or more entity sets. The E-C-R model supports n-ary relationships directly. Similarly, a category can be used to model a subset of one entity set. An example of the E-C-R model is shown in figure 1; the E-C-R diagram is an extension to the E-R diagram [CHEN76]. Rectangular boxes represent entity sets, hexagonal boxes represent categories and diamond shaped boxes represent relationship sets.

Two types of categories exist in the E-C-R model. A sub-class categtory is a grouping of entities from one entity set or category. Members of any entity set may belong to any number of sub-class categories. Sub-class categories could be either attribute defined or user defined [HAMM81]. The category concept allows one to model generalization. In the example of figure 1, CAR and TRUCK are defined to be sub-classes of the VEHICLE set. A VEHICLE can be a CAR or TRUCK or both.

The second type of category is used to group entities playing the same role in a relationship. The category OWNER includes the entities COMPANIES and PERSONS playing the owner role in the OWNER relationship. Thus owners are a subset of the union of COMPANIES and PERSONS.

## 2.0 OUR APPROACH TO VIEW INTEGRATION

We shall review some of the important features of view integration covered in our previous work [NAVA82, ELMA84] and some underlying assumptions that set the stage for the problem of relationship integration.

### 2.1 INTEGRATION PROCEDURE

In this paper we assume the view integration framework to be essentially similar to that defined by Navathe and Gadgil [NAVA82] (see figure 2). The outline of the integration procedure consists of:

1. Specification and interactive modification of inter-view and intra-view assertions by the database designers. This is a "pre-integration" phase to specify the exact correspondences among the attributes, object classes and relationships in the different views. Techniques such as those discussed in [MANN84] may be used.

2. Interactive integration of object classes and relationship classes based on the specified assertions. Object integration rules are given in [ELMA84], and relationship integration rules are presented in this paper. Some assertions may be modified during this phase.

3. Generation of mappings from the global schema to the local views.

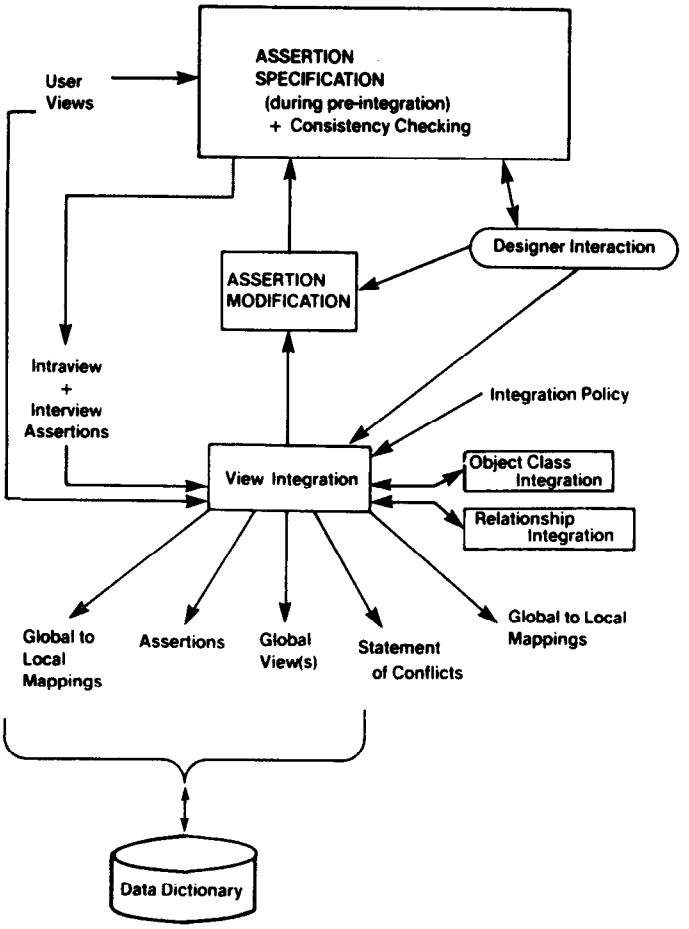



Figure 2.

Relationship integration applies during step 2 above. Our overriding philosophy behind view integration is one of arriving at a "compromise structure". When presented with alternative views of the same situation, we accept the more general view. It is our belief that in real life database design, the design activity proceeds essentially in this manner. Any additional constraints locally applicable to a user view can always be enforced on top of the more general view.

In the context of the E-C-R model, the integration of user views results in a combination of the following changes to one view so that the semantics of the other view can be accommodated:

a. Creating new categories from one or more entity classes.

b. Defining new relationships among entity classes and newly created categories.

c. Making new relationships sub-relationships of others.

## 2.2 PRE-INTEGRATION PHASE

A pre-integration phase is necessary to specify a correspondence among objects in different views. This is necessary to deal with synonym-homonym problems.

The pre-integration phase "establishes" the following among the individual user views:

1. Class name correspondences.

2. Attribute name correspondences.

3. Candidate keys for each class.

4. Correspondences between entity classes in several views.

5. Correspondences among role names and relationships.

By "establishes" we mean that the given specifications are analyzed and additional designer input in terms of new assertions statements is sought to define the above correspondences.

The notion of object class similarity is hard to define precisely. For example, if view 1 has class ENGINEERS and view 2 has class SECRETARIES, the two classes include the same type of entities (EMPLOYEES), but will not have any entities in common. This conclusion cannot be drawn just by comparing namestring ENGINEERS with namestring SECRETARIES. Another comparison of the namestrings ENGINEERS and SUPERVISORS would indicate that they are also dissimilar; however the relationship among them is not quite the same as in the previous case, since they may share some common member entities.

It is possible to develop an automated tool to assist the designers in establishing class correspondences by comparing the strings for class names and attribute names. This method can detect homonyms where the namestring matches but the meaning is different. However, when different views have completely different names for similar entity classes (synonyms), the entire responsibility of denoting which classes are similar will fall on the designer. Thus, the following three factors contribute toward establishing the degree of similarity or degree of match among views:

1. A specification of which classes are similar, identical to, overlapping, or contained in one another. It may be done in a specially designed constraint specification language.

2. A system-guided comparison of namestrings.

3. Designer's input to confirm and resolve correspondences.

## 2.3 A REVIEW OF OBJECT INTEGRATION

Before getting into the integration of relationships, we shall briefly address integration of object classes. Object integration is a precursor to relationship integration, since only after object integration is it possible to determine the similarity of relationships. Once the similarity among object classes is established during the pre-integration phase, similar objects from all the views can be integrated. Integrating two object classes from different views depends on the extensions of these database objects when the database is populated. We refer to the extension of an object class in a view as the domain of the object class. Two similar object classes A and B from different views could be related in the following ways:

1. DOM(A)=DOM(B)

2. DOM(A)$\supseteq$DOM(B) or DOM(B)$\subseteq$DOM(A)

3. DOM(A)$\cap$DOM(B)$\neq$0 and DOM(A)$\not\supseteq$DOM(B) and DOM(B) $\not\subseteq$ DOM(A)

4. DOM(A)$\cap$DOM(B)=0

When integrating object classes, the above mentioned constraints will have to be reflected in the global schema. Elmasri and Navathe [ELMA84] have discussed the process of object integration in detail. We shall only mention the results briefly for each of the above cases.

1. DOM(A) = DOM(B): The objects in domain A are identical to objects in domain B, i.e. the extensions of the objects in A and B are identical. In this case the object classes A and B are integrated and a single object class C is created. Consider the example shown in figure 3.

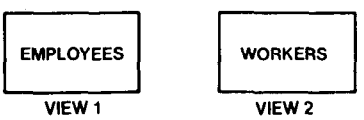



Figure 3.

After an assertion by the designer that the domain of EMPLOYEES in view 1 is identical to the domain of WORKERS in view 2, an integrated object class of EMPLOYEES is created. The attributes of the newly created class are the union of the attributes of the merged identical classes. If the keys of both classes are not the same, then the designer will select a primary key for the class C, and other keys will become secondary keys in the conceptual schema.

2. DOM(A)$\supseteq$DOM(B): When the domain of a class B is a subset of the domain of class A, the object B is represented as a category to denote the sub-class relationship. Consider two objects, STUDENTS and GRADUATE-STUDENTS in two views. The domain of class GRADUATE-STUDENTS is a subset of the object class STUDENTS. The result after integration is shown below in Figure 4.

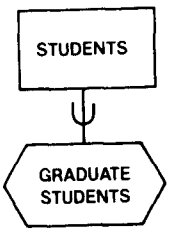



Figure 4.

When integrating n object classes, a hierarchy of sub-class relationships are established.

3. DOM(A)$\cap$DOM(B) $\neq$ and DOM(A)$\not\supseteq$DOM(B) and DOM(B) $\not\subseteq$DOM(A):

In this case, even though the objects A and B are related, neither is a subset of the other. In integrating classes A and B, an object class AB whose domain is the union of both classes A and B is created. Consider two objects TRUCKS and TRACTOR-TRAILER in two different views. Clearly the object classes do not contain each other but are related. Integrating the object classes results in a generalization hierarchy. The result is a VEHICLE entity class and two sub-classes TRUCKS and TRACTOR-TRAILER as shown in Figure 5.

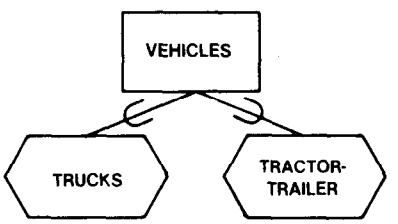



Figure 5.

The domain of the VEHICLES class is the union of domains of TRUCKS and TRACTOR-TRAILER.

4. DOM(A)$\cap$DOM(B) = 0:

In this case, even though the classes are specified to be similar by the designer, they have not objects in common. The integration of such objects is left to the designer. Those object classes which are not integrated will be retained without any modification.

## 3.0 RELATIONSHIP INTEGRATION

The problem of integrating relationships is closely tied to the problem of integrating object classes. In this section, we shall primarily address ourselves to the problem of integrating relationships and make reference to entity class integration only when necessary. So far we have tried to separate the above two kinds of integration in our work; later we propose to put these two under a single framework. Figure 2 shows our intended framework of the view integration system which is similar to what we proposed in [NAVA82]. The additional feature relevant to this paper is that the processes of object class integration and relationship integration are involved independently, but possibly iteratively. The exact nature of this interaction is yet to be investigated.

Given an entity class A1 in view V1, whenever there is a "similar" or "compatible" entity class A2 in the view V2, all relationships in which A1 participates become potentially subject to an integration with all the relationships in which A2 participates. The actual merging of the relationships, however is subject to the semantics which are determined by the roles of the entity classes, the degree, and the cardinality ratios of the relationships, etc. (definitions follow in Section 3.1). Furthermore, when entity classes A1 and B1 in V1 respectively match A2 and B2 in V2, there is a potential for various types of relationships to exist among the pairs of entities. Only after a semantic scrutiny of those relationships, can some of them be integrated.

An external specification relating constructs between views, as well as the designer input is supposed to aid in the determination of which relationships should be subjected to merging. One implicit assumption is that relationships are being considered two at a time for intergration. The ideas are applicable to n-ary intergration; however, the detailed mechanics of n-ary integration are yet to be investigated.

Our approach differs from the previous approaches [ELMA79, BATI82] in that we analyze integration based on both structural and semantic information and that we include integration of relationships of different degree whenever possible.

### 3.1 CRITERIA FOR RELATIONSHIPS COMPARISON

For the purpose of view integration, relationships can be classified by three different characteristics as follows:

1. Degree of a relationship: The degree of a relationship is the number of entities (not necessarily distinct) participating in that relationship. An entity by itself may be treated as a zero-degree relationship for purposes of comparison with other relationships. Unless otherwise specified, an n-degree relationship would have n entity classes participating in it.

2. Roles in a relationship: A role name signifies the role played by an entity class in a relationship [BACH77, CHEN76]. There is a distinct role name for every entity class participating in a relationship.

   The semantic equivalence of relationships is based on the correspondence between rolenames and the relationship instances. During pre-integration phase, the above correspondences are established. Besides, many database query languages directly support role names in their selection expressions: e.g., the GORDAS language [ELMA81]. Hence both view modeling and data retrieval are facilitated for the end user if role names are used.

3. Structural constraints: A relationship between two classes of objects is a mapping that associates the objects from one entity class with objects from other entity classes. Any specification rules supported by the model to express the constraints are called structural constraints [ELMA80]. We shall consider cardinality constraints as the primary type of structural constraints in the integration process.

   The cardinality constraints in a binary relationship place restrictions on the number of objects of one entity class that may be related to an object of the other entity class. We associate two numbers: the MAX CARDINALITY ratio and MIN CARDINALITY ratio to specify the maximum and minimum number of relationship instances per an instance of the given entity type. A MIN CARDINALITY ratio of 0 implies a partial participation and 1 implies a total participation of the entity of a relationship. For an n-ary relationship, the cardinality ratio concept is normally extended to specify the number of relationship instances which may be related to an entity of the participating class. Thus, in a ternary relationship (A,B,C), the three cardinality ratios refer to the ratios of occurrences A:(B,C), B:(A,C) and C:(A,B). These ratios are used by Lenzerini and Santucci [LENZ83] in their cardinality specification technique. Alternately, the ratios (A,B):C, (B,C):A, (C,A):B are also significant and play an important role during integration (See Figure 21).

## 3.2 CLASSIFICATION OF RELATIONSHIPS

In order to develop a systematic approach to the integration of relationships, we consider the above three aspects of relationships in the form of a tree structure. The top node of the tree is called the degree of a relationship. At the second level, we recognize differences in roles. This has two possible outcomes: same role or different role. The last level of the tree has the nodes called structural constraints. The edges are named as shown in figure 6. They indicate that the structural constraints may be disjoint, one constraint may be a subset of the other, or the two sets of constraints may be overlapping.

The leaf nodes of this tree structure represent possible outcomes when relationships from different views are compared. The left half of the tree yields cases which are easier to deal with than the right half. In what follows we discuss these cases in turn.

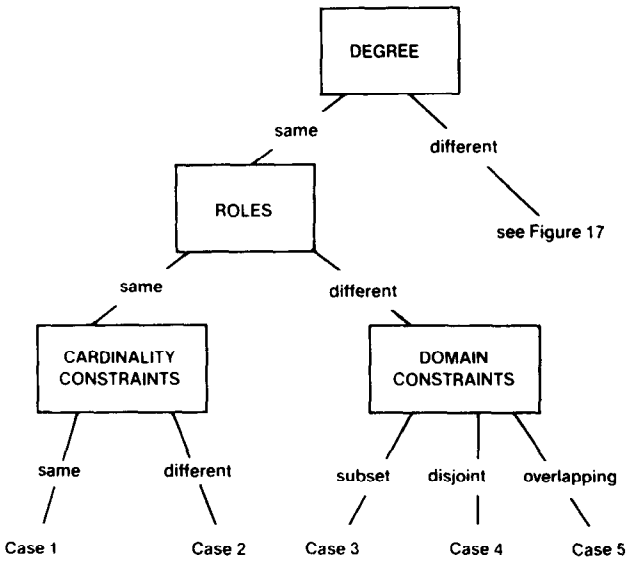



Figure 6.

## 3.3 MERGING RELATIONSHIPS OF THE SAME DEGREE

a. Case 1 - Same role and same structural constraints (or absence of structural constraints):

This is the simplest case of relationship integration. Two views V1 and V2 contain a relationship between two similar entity classes. Only one view is retained in the schema after integration. If an entity class in view V1 is a subset of the entity class in view V2, then the corresponding sub-class is created to hold the entities of view V1. The reader should refer to our previous work on object integration for more details [ELMA84]. An example of this kind is shown in Figures 7 and 8. Figure 7 shows the two input views where the domain of the entity classes GRAD-STUDENT and CS_STUDENT is different in both the views. The integration schema (Figure 8) uses two categories called GRAD_STUDENT and C.S._STUDENT.

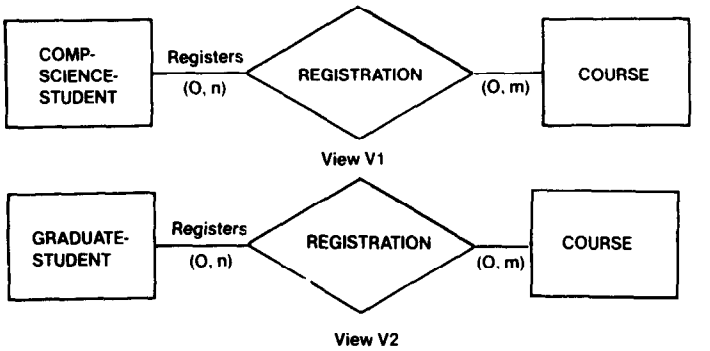



Figure 7.

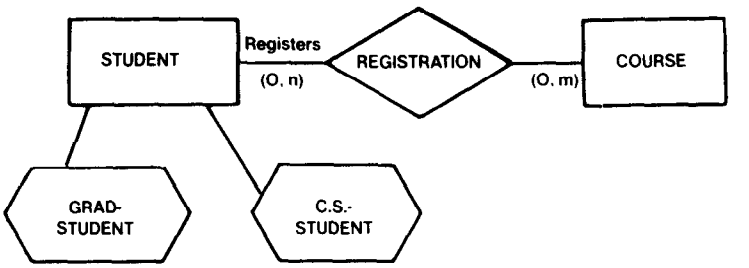
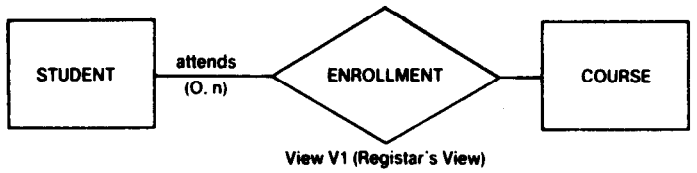



Figure 8.

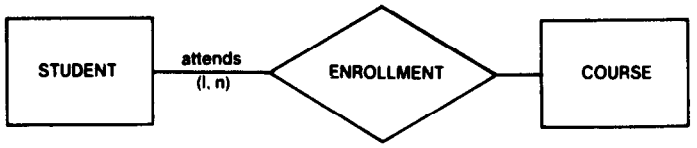
b. Case 2 - Same role and different structural constraints:

In this case one of the views is more constrained than the other. Consider the example where, for the relationship being integrated, the participation of an entity in view V2 may be total, while the participation of the same entity in view V1 may be partial. Since the constraints from both the views are conflicting, we let the tighter constraint apply - that the view with the total participation remain in the schema. The integration process is illustrated in Figure 9. The first view is held by the Registrar's office, where some students are possibly not currently enrolled in any courses. The second view is held by the accounting office, which only includes currently enrolled students.

The participation of STUDENTS in view V2 is total, while it is partial in view V1. The integrated schema is shown in Figure 10. A category termed REGISTERED-STUDENT is created to contain all the students participating in the enrollment relationship. Any STUDENT instance which is participating in the ENROLLMENT relationship is automatically made a member of the REGISTERED-STUDENT category.




View V1 (Registar's View)




View V2 (Accounting View)

Figure 9.

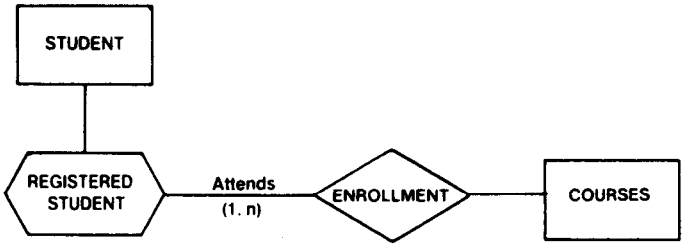


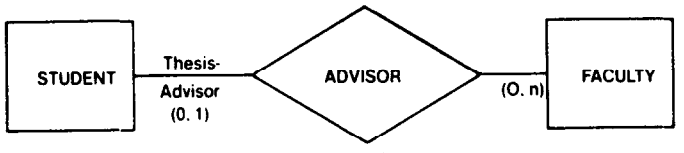

Figure 10.

c. Different roles:

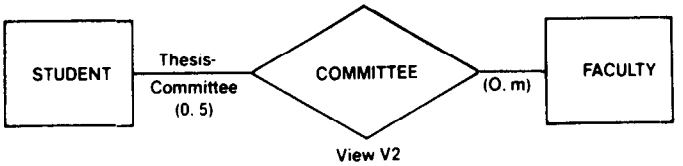
When different roles are used in relationships, the relationships are not identical since they convey different semantics. In many cases, the object classes participating in the relationship will not be identical. During object integration, additional sub-classes may have been created (see Figure 16, with sub-classes home-borrower and auto-borrower). We shall consider three cases:

Case 3. Containment:

In this case the relationship in view V2 contains all the instances of view V1. (See Figure 11).




View V1




View V2

Figure 11

The relationship in V1 is a subset (sub-relation) of the relationship in V2. The integrated schema is shown in figure 12. The schema reflects the fact that the ADVISOR relationship is a subset of the COMMITTEE. This would help maintain the integrity of the database namely, that all instances in ADVISOR relationship should be contained in the COMMITTEE relationship.

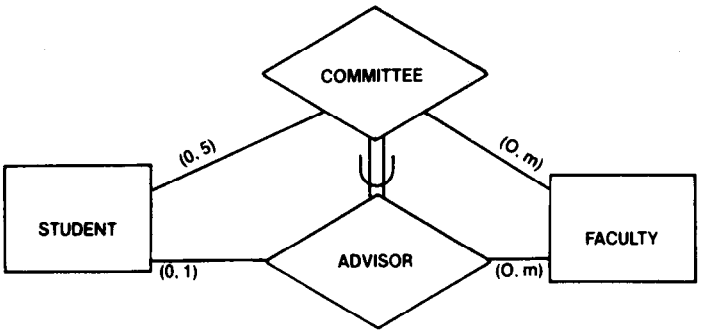
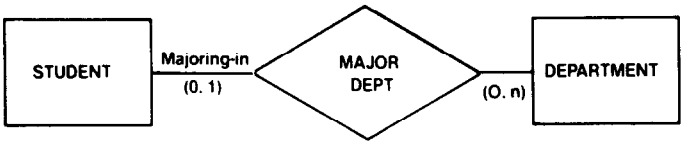


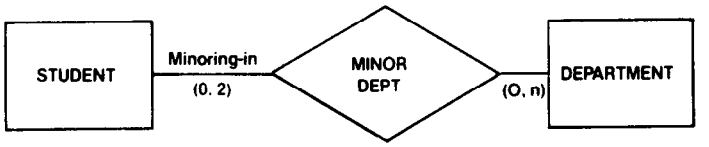

Figure 12.

Case 4. Disjoint relationships:

When the relationships in the views convey different semantics and are unrelated, then the relationships are included in the schema without any modifications.




View V1




View V2

Figure 13.

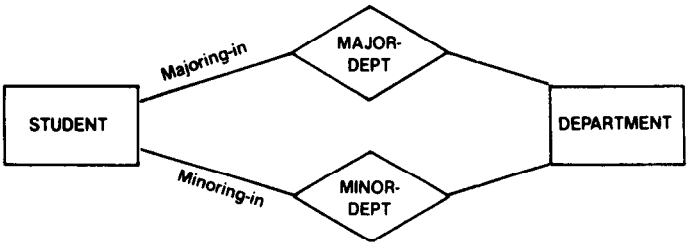



Figure 14.

In the above example, the relationships in view V1 and V2 are disjoint in terms of their contents. The two views are maintained in the schema without any modifications. The resultant schema is shown in Figure 14.

Case 5. Intersecting relationships:

Relationships described between the same set of entities in two views could have some common instances, but neither of the relationships in the views is a sub-set of the other. Consider the views shown in Figure 15.

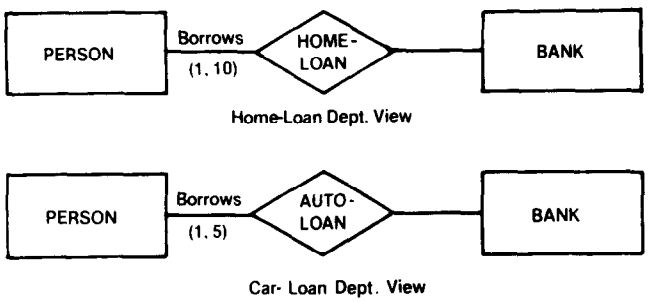



Figure 15.

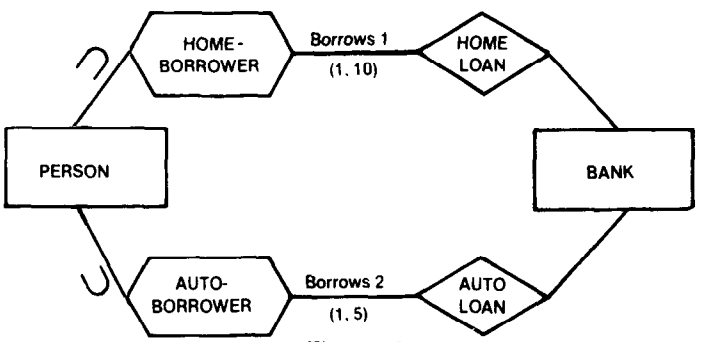



Figure 16.

A PERSON could be both a home borrower and an auto borrower. The PERSON instances in both the views potentially overlap. Hence two categories are created from the PERSON entity class during object integration. These categories could be either attribute defined sub-classes or user defined sub-classes or both. If these categories are attribute defined then an insertion of a PERSON instance automatically creates an instance of the respective category. In case of user defined category, the user has to specify inclusion into the categories. The schema after relationship integration is shown in Figure 16.

### 3.4 MERGING OF RELATIONSHIPS OF DIFFERENT DEGREE

When integrating relationships of different degree, the following possibilities exist:

a. The relationship of lower degree is derivable from the relationship of higher degree. Derivability may be defined in the context of a given DBMS in which the integrated schema is going to be implemented. A possible informal definition is as follows: A view V1 is derivable from V2 if all entity and relationship occurrences in V1 either occur directly in V2 or can be generated from V2 by the use of higher level languages (like SQL or GORDAS [ELMA81]).

b. The different relationships may be made compatible by enforcing additional semantic constraints. These constraints are of the following type:

   1. Cardinality constraints among three or more entity types; e.g. for the relationship (a,b,c), cardinality ratios (a,b):c, (a,c):b and (b,c):a, which are typically absent from graphic notations are important to enforce compatability. Lenzerini and Santucci [LENZ83] provide some good examples of different cardinality constraints.

   2. Set-subset constraints among entity classes.

   3. Derivability constraints to make one view derivable from the other. These constraints would take the form of more complex specifications which can be given using higher level languages.

c. The different relationships cannot be dealt with as in a) or b) above. Then we let them co-exist in the integrated view and let the designer have the final say in merging.

Merger of relationships of different degree is complicated since relationships of higher degree always contain more semantic information [DATE81]. Therefore two considerations are kept in mind.

1. When merging relationships of different degree, the higher degree relationship is always retained.

2. Any attempt to "derive" the higher degree relationships by combining lower degree relationships using operators from high level languages like GORDAS, without considering the semantics of the data, may yield spurious results [DATE81]. A classic example of this type was presented as the "connection trap" by Codd [CODD70].

We classify the integration of relationships of different degree into three different kinds as per the structure shown in figure 17. These three cases are discussed below:

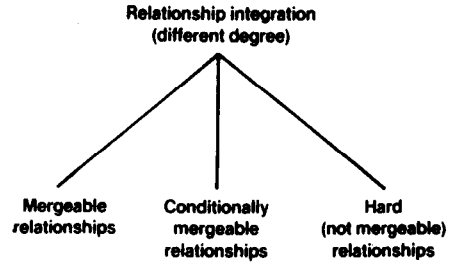



Figure 17.

1. Mergeable relationships:

Two relationships are mergeable when one relationship can be derived from the other. This applies to cases where a relationship in one view is represented as an entity (i.e. a relationship with degree zero) in another view. Consider two views V1 and V2 as shown in Figure 18.

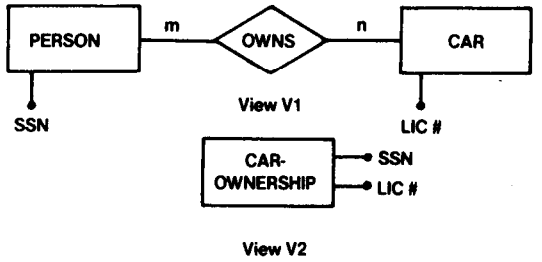



Figure 18.

We refer to the entity in view V2 as a "compressed entity" [NAVA76]. In view V2, the compressed entity CAR-OWNERSHIP does not have any implicit cardinality constraints as far as the relationship between PERSON and CAR is concerned i.e., it may be considered to be many to many. The view V2 can be derived from V1. The mapping constraints (1:1, 1:n, N:1, or M:N) in view V1 can be enforced in V2 by appropriate choice of keys. For example, if SSN and LIC# are the keys of entities PERSON and CAR respectively, then the possible keys of CAR-OWNERSHIP are: SSN alone (for N:1), LIC# (for 1:N), either SSN or LIC# (for 1:1) or the combined key SSN, LIC# (for M:N). Merging is feasible if the key in V2 matches the mapping constraint in V1. Similar arguments apply in the general case to n-ary relationships where an entity in view V2 could be considered to be equivalent to the compression of several entities. The result of integration in the above case is view V1.

2. Conditionally mergeable relationships:

In some other cases of relationship merging, two relationships could be merged only when additional semantic information is specified. Consider two relationships defined in view V1 and V2 in Figure 19.

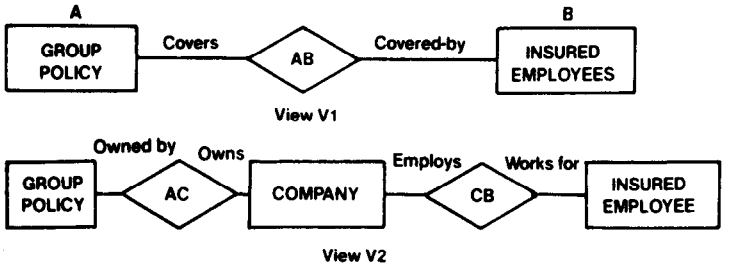



Figure 19.

The "derivability of view V1 from V2 is dependent on the entity C. Under certain cases, when the semantics of the two views is the same, the view V1 may be "derivable" from V2. Here, V2 is retained as the global view. But the semantic interpretation of the relationship in view V1 (as represented by the role) may not be equivalent to the relationship in view V2. For example, in Figure 20 the MAJORS-IN relationship is not a composition of the relationships ATTENDS and OFFERED-BY. Hence all relationships are retained.

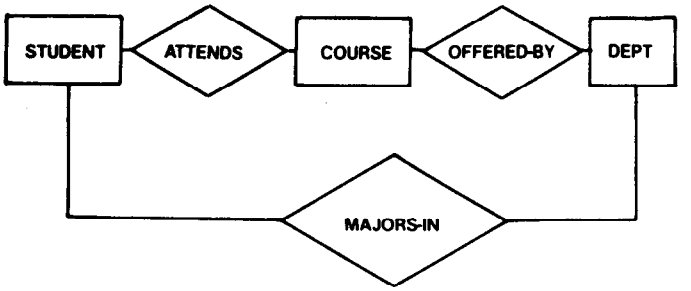



Figure 20.

It is possible in some other cases for a lower degree relationship to be derivable from a higher degree relationship.

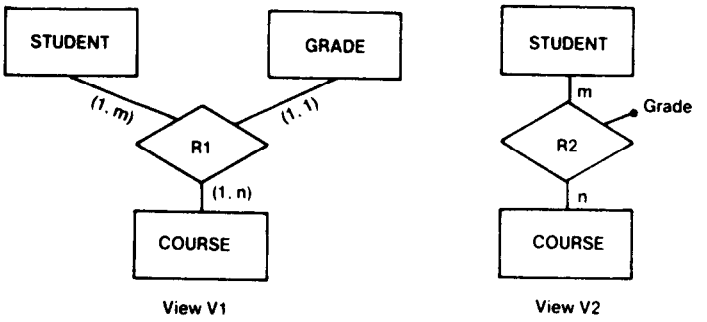



Figure 21.

Consider Figure 21, in view V1, where the cardinality ratios along the arcs are as follows:

| | Min,Max |
|---|---|
| (S,G):C | = (1,n) |
| (S,C):G | = (1,1) |
| (C,G):S | = (1,M) |

(Note: the above ratios are different from the ones used in the previous examples).

The (1,1) ratio between (S,C) and G above implies a functional dependency:

(Student,Course) -----> Grade

Due to this, a binary relationship shown in view V2 is equivalent to the ternary relationship in view 1. In general, a binary view is derivable from a ternary view whenever a (1,1) or (0,1) ratio of the above form exists (see [JAJ083]). The (1,1) above can be modeled by making Grade an attribute of the relationship. A (0,1) ratio would mean an optional attribute. If Grade has its own attributes, they could all be considered as the attributes of the new relationship.

Consider another case of derivable views in Figure 22. The binary relationship in R2 will be derivable from R1 only if the following constraint is specified:

R2 = Projection of R1 on DEALER, CUSTOMER.

The above constraint may be expressed as a derivation specification in a high level language. When two views are conditionally mergeable, the view of higher degree should be kept as the integrated view together with the derivation specifications.

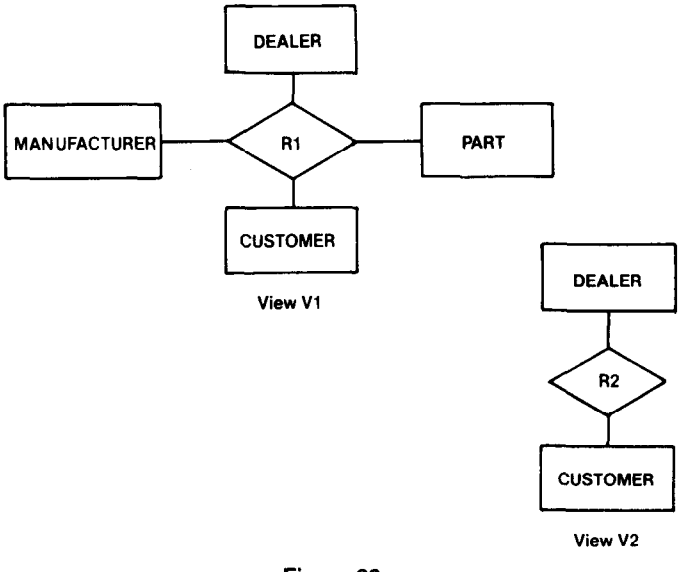



Figure 22.

3. Non-mergeable relationships:

This category includes relationships from two views where some or all entity classes involved may be common to both the views, yet the degree of relationships are dissimilar and the semantics are not exactly the same. Consider a simple example (Figure 23): In the view V1, there are two relationships R1 and R2. They independently describe the relationship of a dealer supplying a part and the customer buying a part. The view V2, however describes the relationship R3 which is a ternary association describing the fact that a customer buys a certain part from a certain dealer.

Although the same entity classes are involved in two views, even if the set of instances of each entity class involved in view V1 and V2 may be identical, it is not possible to consider relationships R1, R2 as being mergeable with or derivable from each other. This is the classical case of a connection trap [CODD70]. Hence the resultant integrated schema will contain the entity classes DEALER, PART, CUSTOMER , and all the relationships R1, R2, R3. Many examples of a connection trap can be constructed.

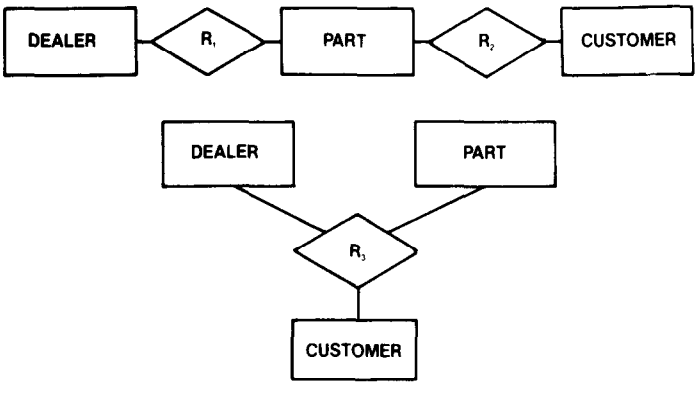



Figure 23.

## CONCLUSIONS AND FUTURE WORK

In the paper we have attempted to classify the problem of relationship integration by taking into account the degree of relationship, roles and structural constraints as the main features to guide the comparison of relationships. If was possible to give definitive rules of integration in cases where the relationships have a matching degree. The problem becomes harder when relationships of different degree have to be merged. Although the discussion was done in the context of the Entity-Category-Relationship Model, we believe that the results can be easily applied to other models.

The work on object integration presented in [ELMA84] and the discussion here on relationship integration constitutes a basis for developing the framework of a database design system. We strongly believe in an interactive approach to view integration where the aim is to arrive at the best compromise solution. With the popularity of Bachman diagrams and the E-R model in practice [see CHIL83], it is reasonable to expect that for the users and developers of databases at large, only the approaches based on semantic data models will seem appealing. In addition semantic models are the only ones which allow meaningful distinctions among views.

Integration tools are also needed for a different problem than database design. This is the problem of integration of existing databases, possibly in a distributed environment, in order to provide a uniform interface to the diverse databases. The techniques presented here are also applicable to the latter problem.

Among the areas we wish to address in our further research are the following:

a. Assertions:

Development of methods for specification & representation of intra-view and inter-view assertions. This constitutes a major portion of the pre-integration phase.

b. The order of integration:

Even with an interactive design tool, it is not clear how we should control the merging of object classes, relationships and any modifications of views. An implied question is whether integration should proceed in a binary or an n-ary fashion i.e., whether two or more views should be simultaneously considered for integration.

c. Language Issues:

These are related to constraint specification, and subsequently, a query specification on views that may guide the designer in selecting among alternatives. Presently we are considering the use of GORDAS [ELMA81] for this purpose.

We hope to develop a prototype design tool for the average practitioner designer as we continue to resolve some of the above issues.

Acknowledgement:

We wish to acknowledge the technical contribution of Jim Larson of Honeywell to the above work. Navathe and Elmasri's work was supported by the Honeywell Computer Sciences Center, Bloomington, MN.

## REFERENCES

[ATZE81] Atzeni, P., Batini, C., and Lenzerini, M., "INCOD-DTE: A System for Conceptual Design of Data and Transactions in the Entity Relationship Model," Proceedings of Second International Conference on Entity Relationship Approach, Washington D.C., October 1981.

[BACH77] Bachman, C., and Daya, M., "The Role Concept in Database Models," Proceeding of the Data Bases Third International Conference on very Large Data Bases, September 1977.

[BATI82] Batini, C., Lenzerini, M., and Santucci, G., "A Computer Aided Methodology for Conceptual Database Design" Information Systems, Volume 7, Number 3, 1982.

[BATI83] Batini, C., and Lenzerini, M., "A Conceptual Foundation for View Integration," Proceedings of IFIP Working Conference, Budapest, Hungary, May 1983.

[BATI83] Batini, C., and Lenzerini, M., "A Methodology for Data Schema Integration in the Entity Relationship Model," IEEE Transactions on Software Engineering, to appear.

[BERN76] Bernstein, P., "Synthesizing Third Normal Form Relations from Functional Dependencies," ACM Transactions on Database Systems, Volume 1, Number 4, December 1976.

[BUBE79] Bubenko, J., "On the Role of 'Understanding Models' in Conceptual Schema Design," Proceedings of the Fifth International Conference on Very Large Databases, October 1979.

[CASA83] Casanova, M. A., and Vidal, V.M.P., "Towards a Sound View Integration Methodology," Proceedings of the Second Symposium on Principles of Database Systems, ACM, New York, March 1983.

[CHEN76] Chen, P.P.S., "The Entity Relationship Model- Towards a Unified View of Data," ACM Transaction on Database Systems, Vol. 1, No. 1, 1976.

[CHIL83] Childers, "Database Design: A Survey of Logical and Physical Design Techniques," Database, Vol. 15, No. 1, March 1983.

[CODD70] Codd, E. F., "Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Vol. 13, No. 6, June 1970.

[DATE81] Date, C. J., "An Introduction to Database Systems," Addison Wesley, New York, 1981.

[ELMA79] Elmasri, R., and Wiederhold, G., "Data Model Integrating Using the Structural Model," Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM, May 1979.

[ELMA80] Elmasri, R., and Wiederhold, G., "Properties of Relationships and their Representation," Proceedings of the National Computer Conference, AFIPS, Volume 49, 1980.

[ELMA81] Elmasri, R., "GORDAS: A Data Definition, Query and Update Language for the Entity-Category Relationship Model of Data" Honeywell Technical Report HR-81-250: 17-38, Honeywell Corporate Technology Center, January 1981.

[ELMA84] Elmasri, R., and Navathe, S.B., "Object Integration in Database Design," Proceedings of the IEEE COMPDEC Conference, Los Angeles, April 1984.

[HAMM81] Hammer, M., and McLeod, D., "Database Description with SDM: A Semantic Database Model," ACM Transactions on Database Systems, Volume 6, Number 3, September 1981.

[JAJO83] Jajodia, S., Ng, P. A., and Springsteel, N. F., "The Problem of Equivalence for Entity Relationship Diagrams," IEEE Transactions on Software Engineering, Vol. SE-9, No. 5, September 1983.

[LENZ83] Lenzerini, M., and Santucci, G., "Cardinality Constraints in the Entity Relationship Model," Proceedings of the Third International Conference on Entity Relationship Approach, October 1983.

[LUN79] Lum, V. Y., et al "1978 New Orleans Database Design Workshop Report," Proceedings of the Fifth International Conference on Very Large Databases, ACM, New York, October 1979.

[MANN84] Mannino, M. V., and Effelsberg, W., "Matching Techniques in Global Schema Design" Proceedings of the IEEE COMPDEC Conference, Los Angeles, April 1984.

[NAVA76] Navathe, S. B., and Fry, J. P., "Restructuring for Large Databases: Three Levels of Abstraction," ACM Transactions on Database Systems, Vol. 1, No. 2, 1976.

[NAVA78] Navathe, S. B., and Schkolnick, M., "View Representation in Logical Database Design," Proceedings of the ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 1978.

[NAVA82] Navathe, S. B., and Gadgil, S., "A Methodology of View Integration in Logical Database Design," Proceedings of the Eighth International Conference on Very Large Data Bases, Mexico City, September 1982.

[NYU78] "Database Design Techniques I: Requirements and Logical Structures" Proceedings of the NYU Symposium on Logical Database Design, (Yao, S. B., Navathe, S. B., Kunii, T., Weldon, J. L., eds), Published as Lecture Notes in Computer Science, Volume 132, Springer Verlag, New York, 1982.

[WIED79] Wiederhold, G., and Elmasri, R., "The Structural Model for Database Design," Proceedings of the International Conference on Entity Relationship Approach, Los Angeles, December 1979.

[YAO82] Yao, S., Waddle, V., and Housel, C., "View Modeling and Integration Using the Functional Data Model," IEEE Transactions on Software Engineering, Volume SE-8, November 1982.