DATA MODEL INTEGRATION USING THE STRUCTURAL MODEL

Ramez El-Masri
Gio Wiederhold

Computer Science Department
Stanford University
Stanford, California 94305

ABSTRACT:

One approach to the design of a logical model for an integrated database requires each potential user or application to specify its view as a data model. An integration phase follows, where these user data models are integrated into a global database model. We address the problem of view integration when user data models are expressed using the structural model [Wi77,WE79].

The structural model is built from relations in Boyce-Codd normal form [Co74]. A basic set of integrity assertions is implicit in the model. The integrity assertions are defined by classification of relations into types, and are represented by connections between relations. We will show how to integrate different representations of two related real-world entity classes.

KEY WORDS AND PHRASES:

Logical database design, data model integration, relational model, structural model, entity classes and relationships, ANSI/SPARC DBMS architecture, conceptual and external schema, data semantics.

1. INTRODUCTION:

An integrated database is expected to be used by a number of users and their applications, not all of whom have the same view of the database. Two approaches exist that allow different user views of the same database. The first approach assumes the existence of a model for the entire database, and allows users to define their views by specifying the parts of the database model that interest them. This approach is used in both network (or CODASYL) [COD74, TF76] and relational [CGT75] databases.

In network databases, a user may define his view as a subschema which is identical to the part of the database model that interests him. Relational databases allow user views that are not identical to parts of the database model. For example, a relation in a user view may be a 'JOIN' of two relations in the database model. However, such relations may not be updated. It has recently been shown that in a general relational model, with integrity constraints specified by assertions, updatability of relational views is limited [DB78].

Recent work in logical database design [Wi77, NS78, WE79] suggests a second approach. Each user first specifies a data model which represents his requirements. An integrated database model is then created by the combination of user data models. If conflicts arise that do not permit integration, contrary data models will have to be changed.

In this paper, we show how to integrate two data models that represent a relationship between two real-world entity classes different ways. The data models, and the integrated database model, are specified in an extended relational model which implicitly represents a limited set of basic integrity constraints. Relations are classified into types, and connections between relations specify the existence dependencies of tuples from separate relations.

All relations in the structural model are in Boyce-Codd normal form [Co74]. In section 2, we define this structural model. A more complete discussion of the origin and use of the structural model is given in [WE79]. In section 3, we show how two related real-world entity classes can be represented in the structural model, and in section 4 we show how the different representations may be integrated.

2. THE STRUCTURAL MODEL:

In our discussion of the structural model, we will often refer to entity classes and relationships. An ENTITY CLASS is a set of real-world objects or events of the same type, such as "CARS IN CALIFORNIA" and "CAR MANUFACTURERS". A relationship between two entity classes is a mapping that associates with each object of one entity class a number of objects (possibly none) of the other entity class. For example, a relationship CAR : MANUFACTURER relates a car with a manufacturer such that the car was made by this manufacturer.

Our model is constructed from relations which are used to represent entity classes and some types of relationships among entity classes. Other types of relationships are represented by connections between relations. Both relations and connections are categorized into several types, according to the structure they represent in a data model.

Connections between relations allow the representation of additional semantic information in the data model. In particular, existence dependencies among tuples in separate relations are clearly represented. The data model designer has several choices for representation of a relationship, and can choose the one best suited to his needs.

Relational concepts are well known. For completeness, we will concisely define relations and relation schemas as we use them in the structural model. We then formally define the concept of connections between relations.

We use A, B, C, to denote single attributes; X, Y, Z, to denote sets of attributes; a, b, c, to denote values of single attributes; and, x, y, z, to denote values of sets of attributes. We assume all sets of attributes are ordered for convenience.

2.1. RELATIONS:

DEF.1: An ATTRIBUTE B is a name associated with a set of values, DOM(B). Hence, a VALUE b of attribute B is an element of DOM(B).

For an ordered set of attributes Y= <Bl, ... Bm> we will write DOM(Y) for the set { <bl, ... bm>| bi is an element of DOM(Bi); i = 1, ... m}. Hence, DOM(Y) is the cross product DOM(Bl) X ... DOM(Bm).

DEF.2: A TUPLE (or value) y of a set of attributes Y= <Bl, ... Bm> is an element of DOM(Y).

DEF.3: A RELATION SCHEMA, Rs, of order m, m> Ø, is an (ordered) set of attributes Y= <Bl, ... Bm>. The RELATION R is an instance (or current value) of the relation schema Rs, and is a subset of DOM(Y).
Each attribute in the set Y is required to have a unique name.
The set Y is partitioned into two subsets, K and G. The RULING PART, K, of relation schema Rs is a set of attributes K= <Bl, ... Bk>, k <= m, such that every tuple y in R has a unique value for the (sub)tuple that corresponds to the attribute set K. (For simplicity, we assume K is the first k attributes of Y.) The DEPENDENT PART, G, of relation schema Rs is the set of attributes G = Y - K. (The - is the set difference operation).
All relations are in Boyce-Codd normal form (see [Co74]).

We will write R[Y] or R[Bl, ... Bm] to denote that relation R is defined by the relation schema Y= <Bl, ... Bm>.

Also, K(Y) will denote the ruling part of relation schema Y, and G(Y) will denote the dependent part. Similarly, for a tuple y in relation R, k(y) will denote the tuple corresponding to attributes K(Y), and g(y) likewise.

A relation R[Y] may have several attribute subsets Z which satisfy the uniqueness requirement for ruling part. In the structural model, the ruling part of a relation will be defined according to the relation type (see section 2.3).

2.2. CONNECTIONS:

We now define the concept of connections between relations. A connection is defined between two relation schemas. Instances of the connection exist between tuples from the two relations. Two major connection types will be distinguished in the structural model: ownership and reference.

DEF.4: A CONNECTION between relation schemas Xl and X2 is established by two sets of attributes Yl and Y2 such that:
a. Yl is a subset of Xl.
b. Y2 is a subset of X2.
c. DOM(Yl) = DOM(Y2).
We then say that Xl is connected to X2 through (Yl, Y2).

The definition of connection is symmetric with respect to Xl and X2, and thus it is an unordered pair (Xl,X2). An instance of the connection exists between every two tuples that have matching values (yl = y2) for the sets of attributes Yl and Y2.

Connections may also be defined for dissimilar but related attributes. Condition (c) above then becomes DOM(Yl) = f(DOM(Y2)), where f is a function that defines matching values. We need the following types of connections to define the structural model.

DEF.5: A REFERENCE CONNECTION from relation schema Xl to relation schema X2 through (Yl, Y2) is a connection between Xl and X2 through (Yl, Y2) such that:
a. Y2 = K(X2).
b. Yl is a subset of K(Xl), or Yl is a subset of G(Xl) (but Yl may not contain attributes from both K(Xl) and G(Xl)).

DEF.5a: A reference connection is an IDENTITY REFERENCE if Yl = K(Xl).

DEF.5b: A reference connection is a DIRECT REFERENCE if it is not an identity reference.

Reference and direct reference are not symmetric with respect to Xl and X2, and are ordered pairs <Xl,X2> when the reference is FROM Xl TO X2. The identity reference is defined symmetrically, but we still consider it to be ordered. This is because identity references will be used to represent subrelations of a relation, defined in section 2.3. We consider the reference to be from the subrelation to the relation.

DEF.6: An OWNERSHIP CONNECTION from relation schema Xl to relation schema X2 through (Yl,Y2) is a connection between Xl and X2 through (Yl,Y2) such that:
a. Yl = K(Xl).
b. Y2 is a proper subset of K(X2).

The ownership connection is also an ordered pair <Xl,X2> when the connection is from Xl to X2.

The connections defined above may be represented graphically as in Fig. 1. They are represented by directed arcs, with the • representing the TO end of the connection.

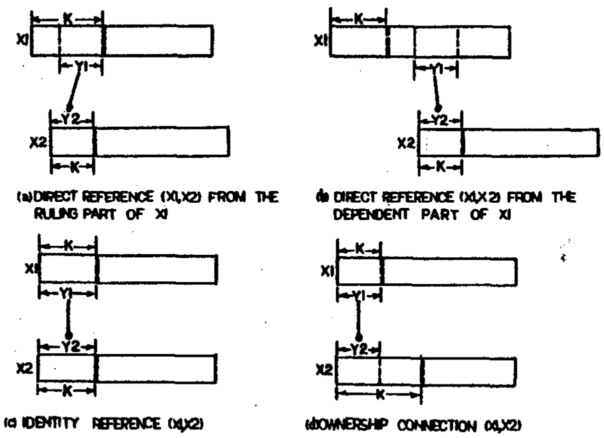


Fig. 1. Types of connections

2.3. TYPES OF RELATIONS:

We now present the types of relations in the structural model. The origin and use of these relations in data model design are discussed in [Wi77] and [WE79]. We will only briefly mention the use of each relation type here.

For the rest of the paper, we will use the term relation for both relation schema and relation, since the meaning is clear from the context.

DEF.7: An ENTITY RELATION is a relation R[X] which defines a correspondence between objects of a class of entities E and the tuples in R.

DEF.8: A REFERENCED RELATION is a relation which has reference connections to it from some relations in the data model.

We now define the five basic types of relations in the structural model.

DEF.9: A PRIMARY ENTITY RELATION is an entity relation that has no direct references or ownership connections to it from any other relations in the data model.

DEF.10: A REFERENCED ENTITY RELATION is an entity relation which has direct references to it from some relations in the data model.

Entity relations are used to represent entity classes. The existence of objects from the class in the relation is determined externally relative to the model. The ruling part defines the tuple that represents an object uniquely. The dependent part attributes describe properties of the object.

The ruling part attributes of a referenced entity relation R are also used for referencing R. Each relation R' that references R will have a set of referencing attributes, having the same domain as the ruling part of R. This constrains insertion and deletion of tuples in both R and R' as we shall see in section 2.4. No such constraints exist for primary entity relations.

DEF.11: A NEST RELATION is a relation, R2, which has an ownership connection to it from exactly one other relation, R1, in the data model. Relation R1 is the OWNER of R2.

The ruling part of a nest relation R2 consists of two parts: attributes which define the connection with the owner relation R1, and additional attribute(s) that uniquely identify tuples owned by the same owner tuple in R1. Existence of tuples in R2 depends on the existence of an owner tuple in R1.

DEF.12: An ASSOCIATION RELATION of order i, i>1, is a relation R that has i ownership connections to it from i other relations in the data model, R1, ... Ri, such that:

a. each Rj has an ownership connection to R through Xj, Yj; j=1, ... i.
b. Yj intersection Yk = empty set for j ≠ k.
c. K(R) = Y1 union ... union Yi.

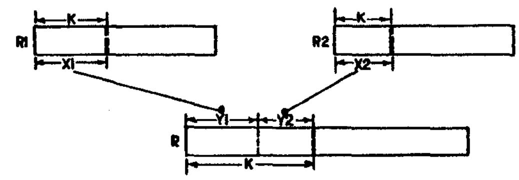


Fig. 2. An association relation, R, of order 2

The ruling part of an association consists of i disjoint sets of attributes; each defines a connection to one of the owner relations. Every tuple in the association is owned by i tuples, one from each of the owner relations.

DEF.13: A LEXICON RELATION R[X] between two sets of attributes Y1 and Y2 defines a 1:1 correspondence between DOM(Y1) and DOM(Y2) such that:

a. Y1 = K(X).
b. the set of attributes Y2 does not appear in any relation other than R.
c. Y1 intersection Y2 = empty set, and Y1 union Y2 = X.
d. R is referenced by one or more relations in the data model.

The main use of lexicons in data model design is to reduce the number of attributes in the core of the data model by removal of equivalent attribute sets to a lexicon. Only one of the attribute sets is maintained in the remainder of the data model. This is particularly useful when several candidates exist for ruling part of a relation.

The above definitions define the five main types of relations: primary entity, referenced entity, nest, association and lexicon. A subrelation may be defined on any relation.

A subrelation R' of some relation R defines a subset of the tuples in R as belonging to the subrelation. This subset of tuples either has a semantic significance in the data model, or has certain additional properties that have to be described, but are not represented in other tuples of the relation. The relation R is called the base relation of the subrelation.

A subrelation has the same ruling part attributes as the base relation, and is connected to the base relation through an IDENTITY REFERENCE. The identity reference reflects the fact that a tuple in the subrelation with the same value for ruling part as a tuple in the base relation actually represents the same object in the data model.

No attributes from the base relation are duplicated in the subrelation other than the ruling part attributes that define the connection.

DEF.14: A (non-restriction) SUBRELATION of relation R[X] is a relation R'[Z] such that:

a. an identity reference exists from R' to R.
b. for every tuple z in R', there exists a corresponding tuple x in R such that k(x) = k(z).
c. (Z - K(Z)) intersection (X - K(X)) = empty set.

R is the BASE RELATION for subrelation R'.

Def.14a: A RESTRICTION SUBRELATION of a relation R[X], restricting the set of attributes Y, Y subset of X, to the domain D', D' subset of DOM(Y), is a subrelation R'[Z] of R such that: for every tuple x in R that has as value for the set of attributes Y a tuple y in D', there exists a corresponding tuple z in R' such that k(z) = k(x).

An example of a restriction subrelation is a relation 'TECHNICAL EMPLOYEES', a subrelation of an 'EMPLOYEES' relation, restricting the attribute "JOB" of 'EMPLOYEES' to the subdomain {engineer, researcher, technician}, say. Existence of tuples in such a restriction subrelation is totally dependent on existing tuples in the base relation. All employee tuples with job value engineer, researcher or technician must also exist in the 'TECHNICAL EMPLOYEES' subrelation. All other employee tuples cannot exist in this subrelation.

An example of a non-restriction subrelation is a relation 'EMPLOYEES IN SPECIAL PROJECT X'. Tuples in this subrelation are determined externally from the data model, but confined to tuples in the base relation of all employees.

We use subrelations to represent three cases:

1. When a subset of a relation has a semantic significance to the data model, or has additional attributes that need to be represented in the model.
2. When integrity constraints require a subset of a relation to own a nest relation or an association, or to be referenced from another relation.
3. When we combine data models to form an integrated database model (see section 4), some data models may represent a subset of a relations from other data models. This will be reflected in the integrated database model.

2.4. MAINTAINING THE STRUCTURAL INTEGRITY OF THE DATA MODEL:

The structural model contains a basic set of integrity constraints that govern existence dependencies of tuples in distinct connected relations. These constraints are expressed implicitly by the connections between relations. Structural integrity exists in our model when the tuples in the database do not violate these constraints.

These constraints are quite useful when a relationship is represented. Many properties of the relationship can be captured in the model.

We now give the constraints associated with each connection. For two connected relations R1 and R2, we say two tuples x and y from R1 and R2, respectively, are connected if the values for the connection attributes in x and y match.

A direct reference from relation R1 to relation R2 specifies the constraints:

1. Every tuple in R1 must be connected to a tuple in R2.
2. Deletion is restricted for tuples in R2. Only tuples that are not connected to any tuple from a referencing relation may be deleted.

An ownership connection from R1 to R2 specifies the constraints:

1. Every tuple in R2 must be connected to an owner tuple in R1.
2. Deletion of an owner tuple from R1 requires deletion of all tuples connected to it in R2.

An identity reference from a subrelation R' to its base relation R specifies the constraints:

1. Every tuple in R' must be connected to a tuple in R.
2. Deletion of a tuple from R requires deletion of the connected tuple in R'.
3. If R' is a restriction subrelation, every tuple in R that belongs to the subrelation (specified by the value of the restricting attributes) must exist in R'.

We will use the following notation to represent connections in our diagrams:

———*	———>	>>>>>>
ownership connection	direct reference	identity reference

The ownership connection is similar to the Bachman arrow [Ba69] of data structure diagrams.

3. DATA MODEL REPRESENTATION OF A RELATIONSHIP BETWEEN TWO ENTITY CLASSES:

In this section, we consider how the structural model represents two related entity classes. This is important when we discuss data model integration in section 4.

Consider two entity classes, A and B, related in some way. One property of the relationship is its CARDINALITY. The cardinality of the relationship places restrictions on the number of entities of one class that may be related to an entity of the other class. The cardinality of the relationship between A and B may be:

a. 1:1, an entity in A may be related to at most one entity in B, and vice versa.
b. 1:N, an entity in A may be related to N entities in B, $N \geq 0$, but an entity in B may be related to at most one entity in A.
c. M:N, an entity in A may be related to N entities in B, $N \geq 0$, and an entity in B may be related to M entities in A, $M \geq 0$.

Additional restrictions may be specified. For example, one may specify that each entity in A is related to exactly one entity in B, or the values for M and N could be more stringently specified ($N > 0$, or $0 < N < 5$, say).

Entity classes may be represented in the structural model by a primary entity relation, a referenced entity relation, or a nest relation. A direct relationship between two entity classes A and B may be represented structural model as one of four choices (Fig. 3):

1. a reference connection: entity class A is represented as a relation Ra, referencing the relation Rb that represents entity class B (Fig. 3.a). The cardinality of the relationship A:B is N:1, and each entity in A must be related to one entity in B.

2. an ownership connection: entity class A is represented by a relation Ra that owns the nest relation Rb which represents entity class B (Fig. 3.b). The cardinality of the relationship A:B is 1:N, and each entity in B must be related to one entity in A.

3. an association relation: relations Ra and Rb represent entity classes A and B, and an association relation Rab represents the relationship (Fig. 3.c). In this case, the cardinality of the relationship A:B is M:N; M >= 0, N >= 0.

4. a nest of references: relations Ra and Rb represent the entity classes A and B. A nest relation Rab owned by Ra, and a reference connection from Rab to Rb represent the relationship (Fig. 3.d). The cardinality of A:B is M:N; M >= 0, N >= 0.

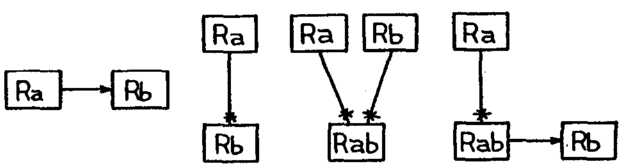


Fig. 3. Representation of two directly related entity classes

Other relationships may exist indirectly between two entity classes. For example, if entity classes A and B, and B and C are directly related, an indirect relationship exists between A and C. We only consider direct relationships. Note that we can further restrict the cardinality of a representation by specifying M or N to be a positive integer.

Data models that represent the same two related entity classes may choose different representations for the relationship according to the way they view the update constraints. Two reasons for such a discrepancy can be distinguished: difference of understanding and difference of representation. We illustrate these differences by example.

1. The two data models may differ in their understanding of the same situation. Consider the two entity classes 'DEPARTMENTS' and 'EMPLOYEES'. One user may consider the relationship between 'DEPARTMENTS' and 'EMPLOYEES' to be 1:N (each employee may work in only one department). A second user is aware of exceptions and considers the relationship to be M:N (an employee may work in more than one department). Here, a disagreement exists about the situation being modelled, and one of the data models is in error. It may be that the first user only knows about employees that work in one department. If such a conflict occurs between the two data models, the situation being modelled will have to be re-examined to determine its actual properties. We will not consider this problem further.

2. The two data models represent the same situation differently, each user choosing the representation that best suits his integrity control requirements. Consider the 'DEPARTMENTS' and 'EMPLOYEES' example, and suppose that the relationship cardinality is 1:N. It may be represented in several different ways in the structural model:

a. an association (Fig. 4.a).
b. a reference connection from 'EMPLOYEES' to 'DEPARTMENTS' (Fig. 4.b).
c. a nest of references from 'EMPLOYEES' to 'DEPARTMENTS' (Fig. 4.c).
d. an ownership connection from 'DEPARTMENTS' to 'EMPLOYEES' (Fig. 4.d).
e. a nest of references from 'DEPARTMENTS' to 'EMPLOYEES'.

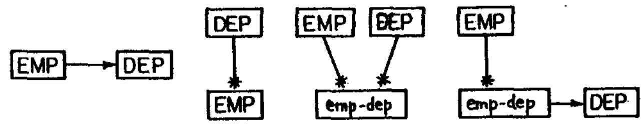


Fig. 4. Some possible representations of DEP:EMP, a 1:N relationship

The different representations reflect different integrity requirements.

a. The association does not place any constraints on the existence of the actual entities, but it can only be made between existing entities.
b. The reference representation requires each employee to belong to a department, and restricts deletion of a department from the database while it is referenced by some employee.
c. The first nest of references representation restricts the deletion of an employee while he is referenced from his department, but allows both employees and departments to exist that are not related to an object of the other class.
d. The ownership connection representation requires that each employee have one department, and that deletion of a department from the database results in the deletion of all the employees who work in that department.
e. The second nest of references representation restricts the deletion of a department while referenced from some employee, but allows both employees and departments to exist independently.

4. INTEGRATION OF DATA MODELS:

We now present the integration of data models. First we briefly define our terminology for logical database design.

A DATA MODEL is a representation of the requirements of a particular potential database user or application. The definition of data models for individual users or groups that expect to use the database is the first step in the design of an integrated database.

The DATABASE MODEL is the integrated model created by merging the individual data models. During merging, differences in view are bound to appear. The differences may be resolved by transformations of the original datamodels. If unresolvable conflicts emerge, managemant decisions have to be made to force data model changes, or to abandon the integration with respect to some data models.

A DATABASE SUBMODEL is the user or application view that is consistent with the integrated database model. If the integrated database model can directly support a user data model, the database submodel for that user will be the same as his data model. If some conflict had arisen during integration, some differences may exist between the data model and the database submodel.

4.1. INTEGRATION OF DATA MODELS THAT REPRESENT A RELATIONSHIP BETWEEN TWO ENTITY CLASSES:

In the following sections (4.1.1 - 4.1.4), we assume that we have two data models, data model 1 (dm1) and data model 2 (dm2). Both data models represent two entity classes A and B, and a relationship A:B between them. Other classes of data will be represented, but we only consider entity classes A and B, and the relationship between them. If dm1 and dm2 use the same representation, there will be no need for any transformation, and the integrated database model (idbm) will use the same representation. If the representations differ, we must create an idbm that supports both dm1 and dm2 correctly.

We use Ra and Rb to denote the relations that represent entity classes A and B. If the representation involves an association relation between Ra and Rb, we will designate it Rab. If a nest of references, owned by Ra and referencing Rb, is represented, we will designate the nest relation Rab also. After integration, the idbm will support database submodel 1 (dbsm1) and database submodel 2 (dbsm2), corresponding to dm1 and dm2 respectively.

In some cases, a subset of the relation Ra (or Rb) in the idbm will correspond to the Ra (or Rb) relation represented in dbsm1 or dbsm2. We will then use a subrelation to represent this subset, and an identity connection will connect it to Ra. Hence, if the Ra relation of dbsm1 corresponds to a subrelation of Ra in the idbm, we denote this subrelation Ra1 in the idbm, and Ra1 will have an identity reference to Ra. The subrelation Ra1 of Ra contains only the ruling part attributes of Ra, so no duplication of information occurs in the representation of the idbm. All other attributes of Ra can be accessed through the identity reference to Ra.

We do not address the problem of authorization of users to perform insertion and deletion. We assume that every database submodel has complete insert, delete, and update authorization over the part of the database model it represents. Hence, if one submodel, dbsm1 say, inserts a tuple that does not violate the integrity constraints of dbsm2, the tuple is inserted in both. If the tuple violates the integrity constraints of dbsm2, it is inserted but remains invisible to dbsm2. For deletion, if deletion of a tuple is legal in dbsm1, say, but the tuple may not be deleted from dbsm2 because of integrity constraints, the tuple will be kept in the idbm and in dbsm2, but will become invisible to dbsm1.

After integration, the idbm will support both dbsm1 and dbsm2. A mapping will exist from each submodel to the idbm. This mapping (Fig. 5) will contain additional integrity rules, derived from the integration process, which apply to the idbm

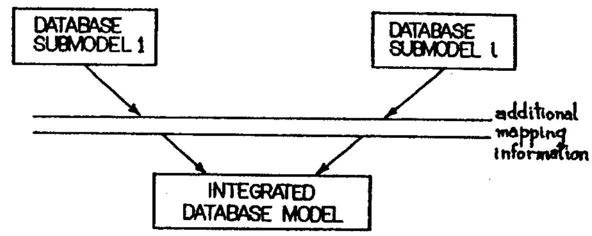


Fig. 5. Additional constraints on the idbm

whenever one of the database submodels performs an insertion, deletion, or update. We list these rules with each case of integration.

There are four ways of representing a relationship A:B between two entity classes in the structural model (Sec. 3). The set of possible cases for combining two different representations is 2 X (4 + 3 + 2 + 1) = 20. We remove 4 cases where the relationship is represented in the same way, and 4 cases because the association is symmetric with respect to Ra and Rb. Then 12 cases remain to be considered. We first consider integration with the association (3 cases, sec. 4.1.1). We then consider the cases that remain with nest of references (5 cases, sec. 4.1.2), reference (3 cases, sec. 4.1.3), and nest (1 case, sec. 4.1.4).

Our assumption (sec. 3) that both original data models accurately represent the same situation implies that the cardinality of both representations is the same. Hence, the data model that can represent more general cardinalities is restricted to the cardinality of the relationship represented in the other data model.

Following each integration case, we give a simple example with attributes. In these examples, some attributes are required to have unique values in tuples of a relation at all times to enforce the specified cardinality of a relationship. These attributes will be marked (U). Attributes will be separate by lines ( ), and the ruling part attributes are shown to the left of a double line ( ).

In order to demonstrate how two different data models are integrated, we present the integration of an association with the nest of references (Fig. 6a). Here, the only difference is that in dm1, deletion of tuples from Rb is unconstrained, while in dm2 such a deletion is restricted by referencing tuples from Rab.

To concile this difference, we create two subrelations Rb1 and Rab1 in the idbm to represent the tuples in Rb and Rab of dbsm1. Some tuples in Rb and Rab of the idbm may have been deleted by dbsm1. If these tuples were referenced, they are only deleted from Rb1 and Rab1 in the idbm, but not from Rb and Rab due to the deletion constraint of the reference in dbsm2. These tuples become invisible to dbsm1.

The database submodels now obey the following rules. Insertion and deletion in Ra from either dbsm1 or dbsm2 is unrestricted, as is deletion of Rab tuples, and of unreferenced Rb tuples. If dbsm1 deletes a referenced Rb tuple (dbsm2 may not perform such a deletion), it is only deleted from

Rbl (and the owned tuples are automatically deleted from Rabl). These rules accurately reflect the constraints of the original data models.

For brevity, we will use a standard format for each integration case listed below. We first give the differences between the two data models, then the additional integrity constraints that have to exist in the mapping information from each database submodel to the integrated database model.

When listing these additional constraints, ("relation name") will mean 'do the specified insertion or deletion if allowed by the integrity constraints of the idbm'. Also, the relation name to the left of the '-' refers to the database submodel, while those to the right refer to the database model. We only consider cases which need additional control from the mapping information.

We now present the demonstration case again in the brief notation to clarify these conventions.

4.1.1. INTEGRATION WITH AN ASSOCIATION:

Here, dm1 represents the relationship A:B by an association relation, and dm2 uses a different representation. The cardinality of the relationship A:B is hence M:N, possibly restricted to that of the representation in dm2.

(a) Association and nest of references (Fig. 6a):

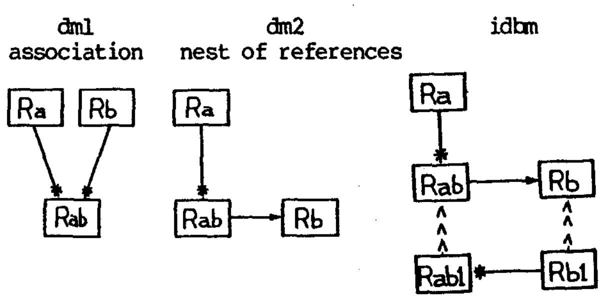


Fig. 6.a. Integration of association and nest of references

Differences:
In dm2, deletion of Rb tuples is restricted by references. Dm1 has no such restriction.

Additional mapping information:

dbsm1: ins Rb - Rb,Rbl
Rab- (Rab,Rabl)
del Rb - (Rb),Rbl

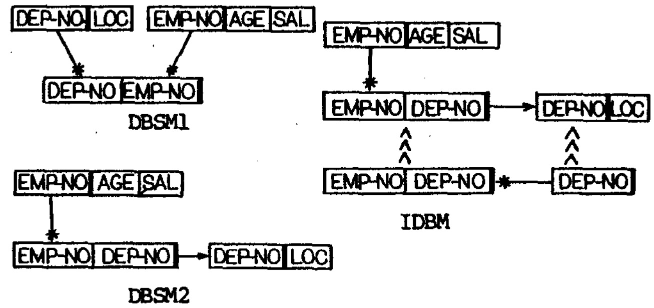
dbsm2: ins Rb - Rb,Rbl
Rab-(Rab,(Rabl))

To clarify our notation, we discuss the additional mapping information for this example.

Insert a tuple in Ra in dbsm1 (or dbsm2) means insert it in Ra in the idbm, since it is not listed. In dbsm1, insert in Rb requires insertion in Rb and Rbl in the idbm. Insert in Rab requires insertion in (Rab,Rabl), the () brackets meaning if the integrity check of the idbm will allow it (here if both owner tuples exist). In dbsm2, insert in Rab requires insertion in (Rab,(Rabl)) which means insert in Rab if the integrity check of the ibdm holds (both the owner tuple in Ra and the referenced tuple in Rb exist), then also insert in Rabl (if the other owner tuple exists in Rbl).

In dbsm1, delete a tuple from Rb means delete it from Rbl, and also from Rb if it is not referenced.

Example:



(b) Association and reference (Fig. 6.b):

The cardinality of the relationship A:B is restricted to N:1, since the reference cannot represent an M:N relationship.

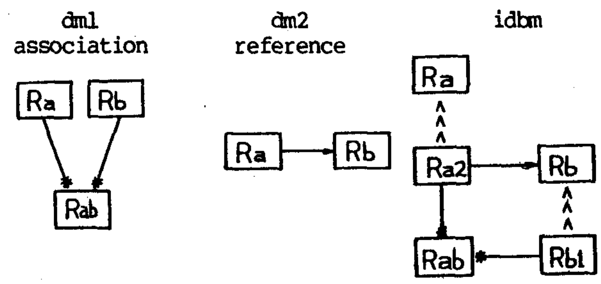


Fig. 6.b. Integration of association and reference

Differences:
1. In dm2, every Ra tuple must reference an Rb tuple, while in dm1 not all Ra tuples have to be related to Rb tuples.
2. In dm2, deletion of Rb tuples is restricted by references.

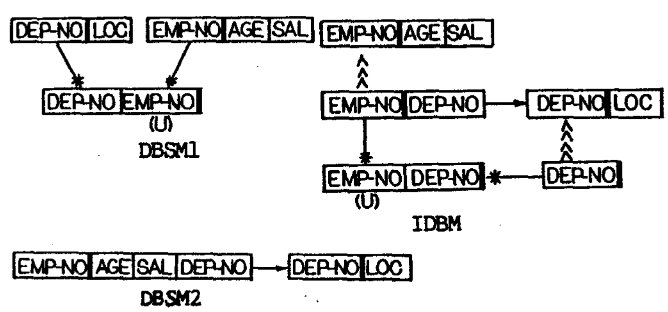
The unrestricted deletion of Rb tuples in dbsm1 leads to the creation of the subrelation Rbl, and the requirement that every Ra tuple must reference an Rb tuple in dbsm2 leads to the creation of Ra2.

Additional mapping information:

dbsm1: ins Rb - Rb,Rbl
Rab- (Ra2,Rab)
del Rb - (Rb),Rbl
Rab- Rab,Ra2

dbsm2: ins Ra - (Ra,Ra2,(Rab))
Rb - Rb,Rbl

Example:



(c) Association and nest (Fig. 6.c):

The cardinality of the relationship A:B is restricted to 1:N.

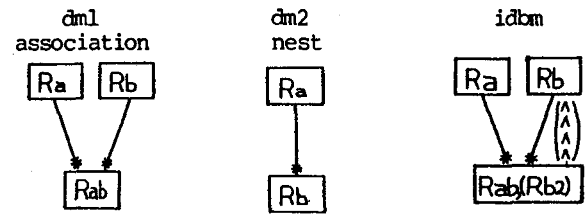


Fig. 6.c. Integration of association and nest

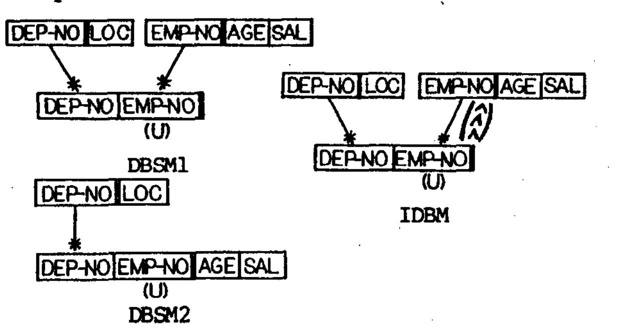
Differences:
1. In dm2, existence of a tuple in Rb requires the existence of the owner tuple in Ra. In dml, Rb tuples can exist independently.
2. In dm2, deletion of a tuple from Ra requires the deletion of the owned tuples in Rb. Dml does not require these deletions.

The Rb tuples in dbsm2 are only those in Rb2, since they require the existence of the owner tuple. Those are the same as the tuples in Rab.

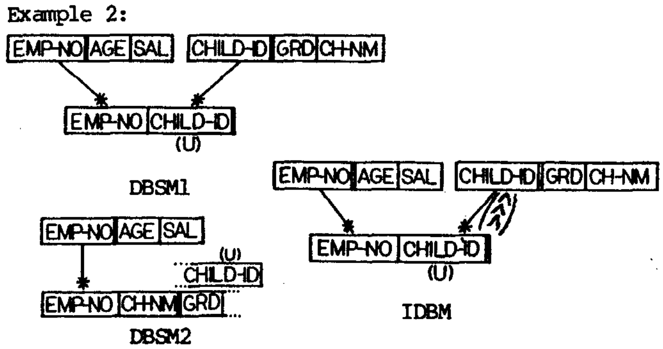
Additional mapping information:
dbsml: dbsm2: ins Rb - (Rb,Rb2)

Here, we must consider two examples, since the nest relation may represent different tuple identification attributes than the association. We first consider the case where the identification is the same. The attribute "EMP-NO" identifies the employee in both dbsml and dbsm2. Since the cardinality of DEPARTMENT:EMPLOYEE is 1:N, "EMP-NO" must have unique values for the attributes marked with a (U). Note that this does not violate Boyce-Codd normal form. In this case, the integration is straightforward (Example 1).

Examplel:



Example 2:



In the second example, identifying attributes are different. Dbsm2 uses the combination <"EMP-NO", "CHILD-NAME"> as ruling part, and dbsml uses only "CHILD-ID". "CHILD-ID" uniquely identifies a child tuple, while "CHILD-NAME" does not. Here, if dm2 did not represent the attribute "CHILD-ID", it has to be made aware of it to maintain the correct mapping between "CHILD-ID" and "CHILD-NAME". Hence dbsm2 will be different from dm2.

4.1.2. INTEGRATION WITH A NEST OF REFERENCES:

Dml represents the relationship A:B as a nest of references from A to B, and dm2 represents it differently. The cardinality of the relationship A:B is M:N, but may again be restricted to the representation in dm2. The nest of references is not symmetric with respect to entity classes A and B, so we must consider it twice with each nonsymmetric representation.

(a) Nest of references and nest of references (Fig. 7.a):

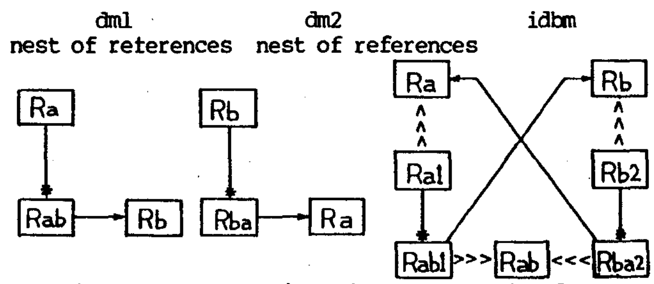


Fig. 7.a. Integration of two nests of references

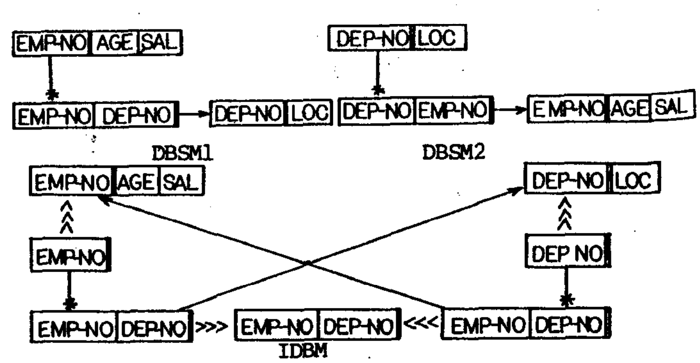
Differences:
1. Deletion of Rb (Ra) is restricted in dml (dm2).
2. Deletion of Ra (Rb) in dml (dm2) requires deletion of owned tuples in Rab (Rba).

Additional mapping information:

dbsml:	ins Ra - Ra,Ral		dbsm2: ins Ra - Ra,Ral	
	Rb - Rb,Rb2		Rb - Rb,Rb2	
	Rab- (Rab,Rabl,(Rba2))		Rba- (Rab,Rba2, (Rabl))	
	del Ra - (Ra),Ral,(Rab)		del Ra - (Ra,Rab)	
	Rb - (Rb,Rab)		Rb - (Rb),Rb2, (Rab)	

When dbsml attempts to delete an Ra tuple that is referenced in the idbm from Rba2, it is only deleted from Ral. If the tuple is not referenced from Rba2, it will also be deleted from Ra. In the

Example:



latter case, the tuples in Rab that correspond to those deleted from Rab1 (due to the deletion of Ra and the ownership connection) should also be deleted, since they no longer exist in either Rab1 or Rba2. Rab exists to ensure the consistency of the tuples associating Ra with Rb (in Rab1 and Rba2).

(b) Nest of references and reference (Figs. 7.b, 7.c):

Both nest of references and reference are non-symmetric, so we must examine two cases.

Case 1 (with reference from A to B):
The cardinality of the relationship A:B is restricted to N:1, since the reference from A to B can only represent an N:1 relationship.

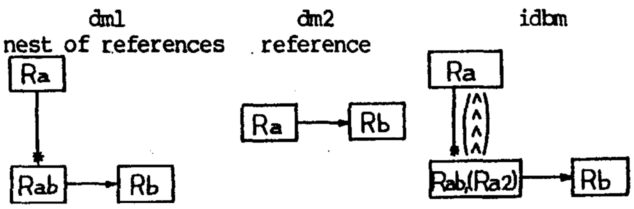
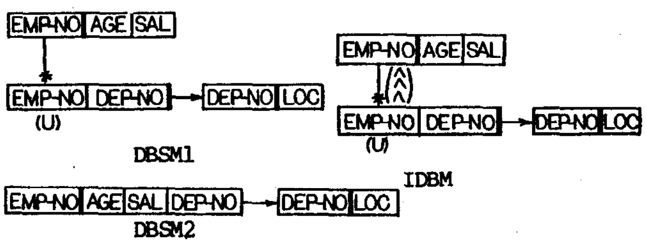


Fig. 7.b. Integration of nest of references and reference (case 1)

Differences:
In dm2, a tuple in Ra must be related to an Rb tuple.

Additional mapping information:
dbsm1: dbsm2: insert Ra - (Ra,Ra2)

Example:
Since the relationship is N:1, "EMP-NO" must have unique values where marked (U).



Case 2 (with reference from B to A):
The cardinality of the relationship A:B is restricted to 1:N.

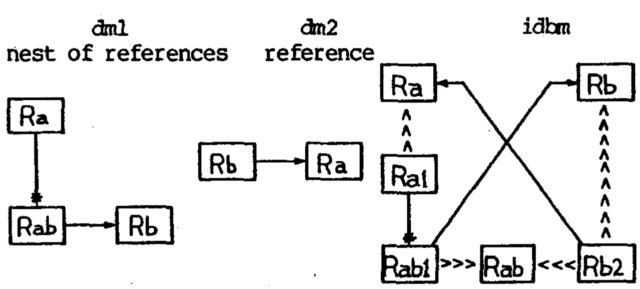


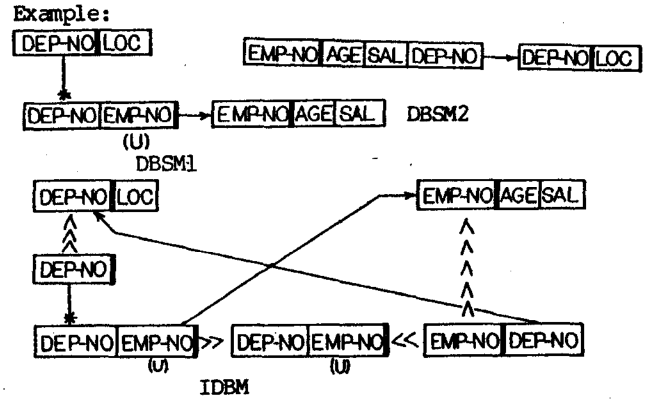
Fig. 7.c. Integration of nest of references and reference (case 2)

Differences:
1. Deletion of Rb (Ra) tuples is restricted in dm1 (dm2).
2. Every Rb tuple in dm2 is related to an Ra tuple.

Additional mapping information:

dbsm1:			dbsm2:		
ins	Ra -	Ra,Ra1	ins	Ra -	Ra,Ra1
	Rab-	(Rab1,Rab,Rb2)		Rb -	(Rb,Rb2, Rab,Rab1)
del	Ra -	(Ra),Ra1,(Rab)	del	Ra -	(Ra,Rab)
	Rb -	(Rb,Rab)		Rb -	(Rb,Rab), Rb2
	Rab-	Rab1,(Rab)			

Example:



(c) Nest of references and nest (Figs. 7.d, 7.e):

Again, both nest of references and nest are non-symmetric, so we must examine two cases.

Case 1 (with nest ownership from A to B):
The cardinality of the relationship A:B is restricted to 1:N, since the nest cannot represent an M:N relationship.

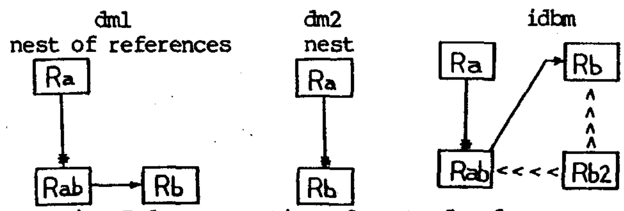
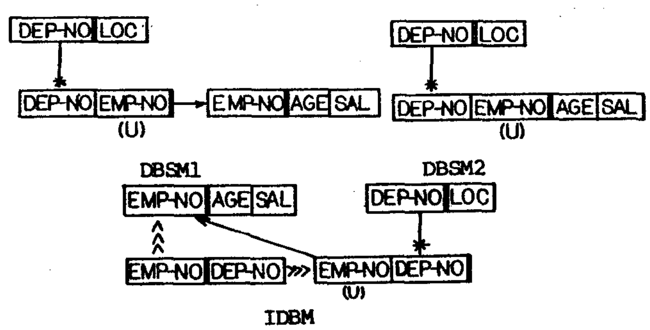


Fig. 7.d. Integration of nest of references and nest (case 1)

Differences:
1. Rb tuples may exist independently in dm1.
2. Deletion of Rb tuples is restricted in dm2.

Additional mapping information:
dbsm1: ins Rab - (Rab,Rb2) dbsm2: ins Rb - (Rb,Rab, Rb2)
del Rb - (Rb),Rb2

Example:
We consider an example with similar identification.



Case 2 (with nest ownership from B to A):
The cardinality of the relationship A:B is restricted to N:1.

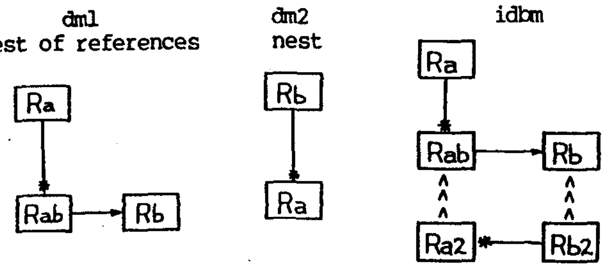
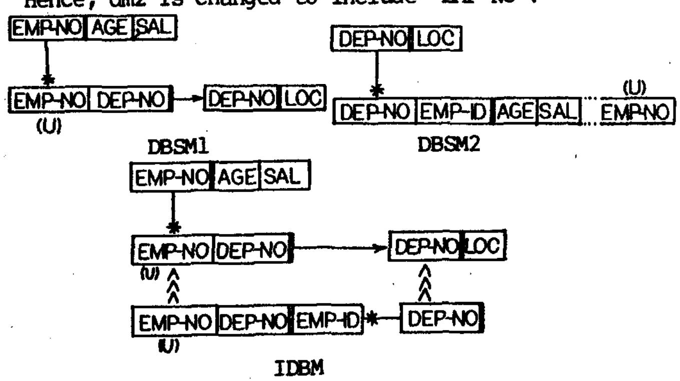


Fig. 7.e. Integration of nest of references and nest (case 2)

Differences:
1. In dml, Ra tuples can exist independently, but in dm2 an owner tuple must exist in Rb.
2. In dml, deletion of Rb tuples is restricted by references, while in dm2, deletion of an Rb tuple requires deletion of related Ra tuples.

Additional mapping information:
dbsml: ins Rb - Rb,Rb2 dbsm2: ins Ra - (Ra,Ra2, Rab)
Rab- (Rab,Ra2)
Rb - Rb,Rb2
del Rb - (Rb),Rb2

Example:
This example has different identification. Hence, dm2 is changed to include "EMP-NO".



4.1.3. Integration with a reference:

Dml represents the relationship A:B as a reference connection from A to B. The cardinality of the relationship is N:1, possibly restricted by the representation in dm2.

(a) Reference and reference (Fig. 8.a):

The cardinality of A:B is restricted to 1:1, since it is N:1 in dml and 1:N in dm2. It would be unusual to encounter these two representations of a 1:1 relationship. However, it can be integrated.

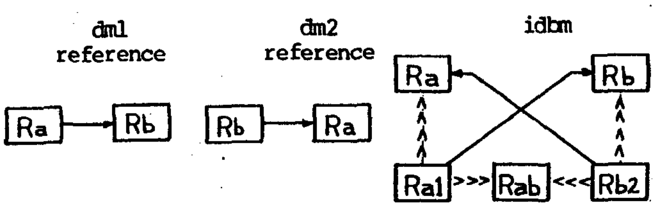
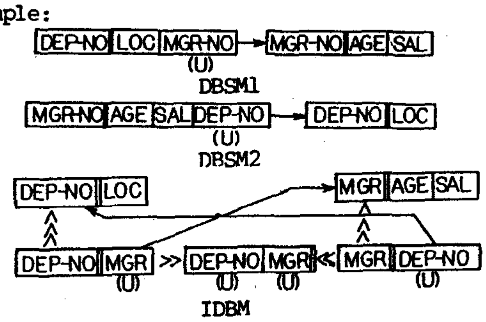


Fig. 8.a. Integration of two references

Differences:
1. In dml (dm2), every Ra (Rb) tuple must reference an Rb (Ra) tuple.
2. Deletion of Rb (Ra) tuples is restricted in dml (dm2).

Additional mapping information:
dbsml: ins Ra - (Ra,Ral, Rab,Rb2) dbsm2: ins Rb - (Rb,Rb2, Rab,Ral)
del Ra - (Ra),Ral,(Rab) del Rb - (Rb),Rb2, (Rab)
Rb - (Rb,Rab) Ra - (Ra,Rab)

Example:



(b) Reference and nest (Fig. 8.b, 8.c):

Case 1 (with nest ownership from B to A):
The cardinality of the relationship A:B is N:1.

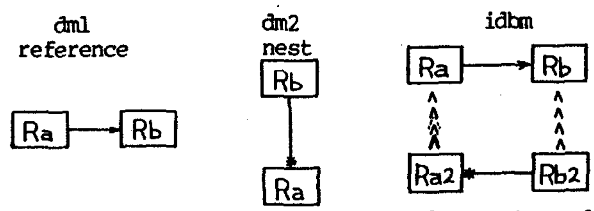
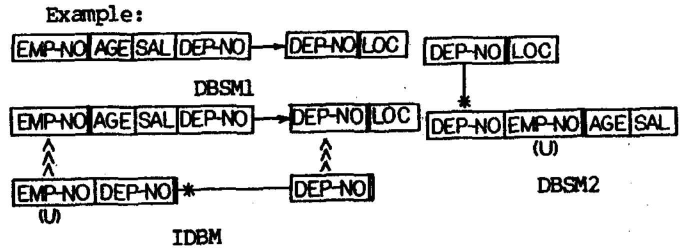


Fig. 8.b. Integration of reference and nest (case 1)

Differences:
1. In dml, deletion of Rb tuples is restricted by references.
2. In dm2, deletion of an Rb tuple requires deletion of owned tuples in Ra.

Additional mapping information:
dbsml: ins Ra - (Ra,(Ra2)) dbsm2: ins Ra - (Ra,Ra2)
Rb - Rb,Rb2 Rb - Rb,Rb2
del Rb - (Rb),Rb2

Example:



Case 2 (with nest ownership from A to B):
The cardinality of the relationship A:B is restricted to 1:1, since in dml it is N:1, and in dm2 it is 1:N.

Differences:
1. Every Ra tuple in dml must reference an Rb tuple. In dm2 every Rb tuple must be owned by an Ra tuple.
2. Deletion of Rb tuples is restricted in dml.

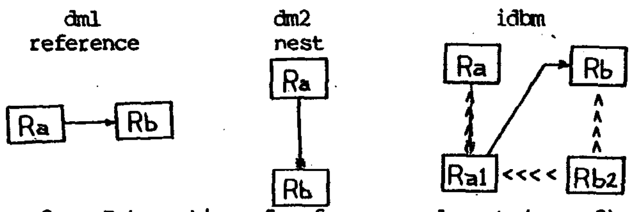
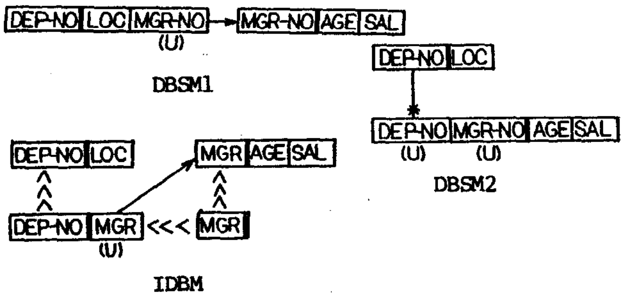


Fig. 8.c. Integration of reference and nest (case 2)

3. Deletion of an Ra tuple in dm2 requires the deletion of owned Rb tuples.

Additional mapping information:
dbsm1: ins Ra - (Ra,Ra1, Rb2) dbsm2: ins Rb - (Rb,Ra1, Rb2)
del Rb - (Rb),Rb2

Example:



4.1.4. Integration with a nest:

Nest and nest (Fig. 9):

The final case to consider is also unlikely. The cardinality of A:B is restricted to 1:1.

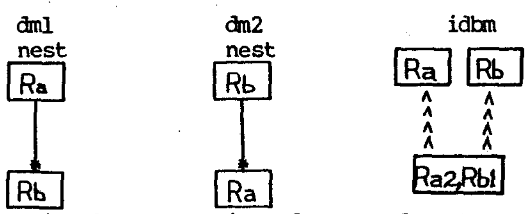
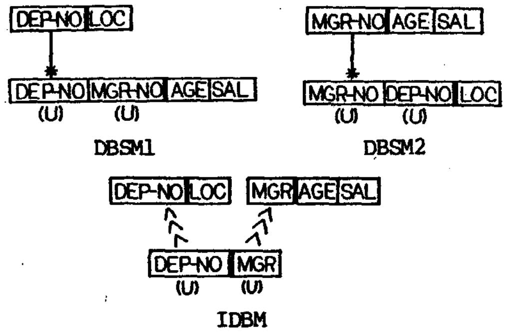


Fig. 9. Integration of nest and nest

Differences:
1. In dm1 (dm2), every Rb (Ra) tuple must be owned by an Ra (Rb) tuple.
2. Deletion of an Ra (Rb) tuple in dm1 (dm2) requires deletion of the owned Rb (Ra) tuple.

Additional mapping information:
dbsm1: ins Rb - (Rb,Rb1) dbsm2: ins Ra - (Ra,Ra2)

Example:



4.1.5. Summary:

In sections 4.1.1-4.1.4, we showed how two different representations of a relationship can be integrated. An integrated database model to support the original data models correctly was created. In some cases, when one of the relationships was represented by an ownership connection, one of the data models had to be changed slightly when the identifying attributes were different in the two data models.

This approach ensures the updatability of both database submodels. The integrated database model sometimes looks quite complex due to the creation of several subrelations. However, implementation of subrelations can be quite simple. A single record type can be created for a relation and all of its subrelations. One additional bit for each subrelation is included in each record to indicate whether that record belongs to the subrelation or not. Fields that exist only in a subrelation have values only if the record is in the subrelation.

4.2. DATA MODELS THAT REPRESENT PARTS OF A RELATION IN ANOTHER DATA MODEL:

We now consider the cases where one data model, dm1 say, represents part of a relation represented in the other data model, dm2 here. Two cases may be distinguished:
1. Dm1 represents a relation represented in dm2, but only represents some of the attributes in that relation.
2. Dm1 represents a subrelation (some of the tuples) of a relation represented in dm2.

4.2.1. Representation of subsets of the attributes:

Suppose both dm1 and dm2 represent a relation 'EMPLOYEES', and it is determined that both are representing the same set of employees. Each data model defines his own set of attributes. A common set of attributes is represented in both models, which includes the tuple identification attributes. Then, the idbm will have to represent the complete set of attributes. Dbsm1 and dbsm2 will each be allowed access to the subset of attributes it represents.

To integrate the two data models, we include in the idbm a base relation 'EMPLOYEES' which contains the common attributes, and two subrelations 'EMP1' and 'EMP2' of 'EMPLOYEES'. Each subrelation will represent the additional attributes in one of the data models (and the identifying attributes).

For example, dm1 could represent a relation 'EMPLOYEES' defined by the attribute set <"NAME", "ADDRESS", "H-PHONE", "OFFICE", "O-PHONE", "DEPT"> (representing a directory of the employees). Dm2 represents a relation 'EMPLOYEES' defined by the attribute set <"NUMBER", "NAME", "AGE", "JOB", "SAL", "DEPT"> (representing job information). The idbm then represents a base relation 'EMPLOYEES' <"NAME", "DEPT"> and two subrelations 'EMP1' <"NAME", "ADDRESS", "H-PHONE", "OFFICE", "O-PHONE"> and 'EMP2' <"NAME", "NUMBER", "AGE", "JOB", "SAL">. If dbsm1 inserts a tuple, it is only inserted in the base relation and 'EMP1' until dbsm2 inserts it.

4.2.2. Representation of subsets of tuples:

Here, we have two possibilities. The first is when one data model, dm1 (say), represents a relation R, while dm2 represents a subset of the tuples in R. In this case, the idbm will represent R and a subrelation R' of R. The subrelation R' may be a restriction subrelation if the subset of tuples in R' is determined by attribute values in R, or a non-restriction subrelation if the subset of tuples in R' is determined externally, independently from the model. For example, the data model for the payroll department could represent all employees of a company in an 'EMPLOYEES' relation, while the data model for the sales department of the company represents the relation 'SALES EMPLOYEES', the employees that work for the sales department. In the idbm, the 'EMPLOYEES' relation is represented, with a subrelation 'SALES EMPLOYEES'. If 'EMPLOYEES' has an attribute "DEPT" represented, the subrelation 'SALES EMPLOYEES' would be a restriction subrelation, restricting the "DEPT" attribute to the value 'sales dept'. If the "DEPARTMENT" attribute is not represented in 'EMPLOYEES', 'SALES EMPLOYEES' would be a non-restriction subrelation. After integration, the database submodel for the sales department would only be allowed access to tuples in the subset, while the database submodel would be allowed access to all employee tuples.

The second possibility is that dm1 represents a relation R1 and dm2 represents a relation R2 such that:

R1 intersection R2 $\neq$ empty set

R1 - R2 $\neq$ empty set and R2 - R1 $\neq$ empty set

The idbm will then represent a relation R = R1 union R2, and two subrelations of R, R1 and R2. Again, R1 and/or R2 could be restriction or non-restriction subrelations. For example, refering to a university database, dm1 (representing the computer science department of the university) could represent a relation 'CSD PROFESSORS', and dm2 (representing information about permanent faculty) could represent the relation 'TENURED PROFESSORS'. The idbm would then represent a relation 'PROFESSORS', and two subrelations of 'PROFESSORS', 'CSD PROFESSORS' and 'TENURED PROFESSORS'. Each database submodel is then allowed access to his subset of tuples, and the base relation assures integrity of common information.

5. CONCLUSIONS:

The structural model is an extended relational model that provides tools for a more extensive representation of the data semantics. In particular, many properties of relationships between entity classes are captured.

The structural model can be used to represent different user views of a database. These views must be integrated in order to have a global conceptual model.

We have shown here how different representations of two related entity classes in two data models can be integrated. The methodology for integration has to be extended to more general cases, and we are pursuing further work in this area.

ACKNOWLEDGEMENTS:

This work was supported by ARPA grant MDA903-77-C-0322. We wish to thank Sheldon Finkelstein, Daniel Sagalowicz, Bharat Bhargava, and other members of the KBMS project for helpful discussions.

REFERENCES:

[Ba69] Bachman, C.W., "Data Structure Diagrams", Database (ACM SIGBDP), Vol.1, No.2, Summer 1969

[CGT75] Chamberlin, D.D., J.N.Gray, and I.L.Traiger, "Views, Authorization, and Locking in a Relational Database System", Proc. NCC, AFIPS, 1975, pp.425-430

[Ch76] Chen, P.P.S., "The Entity-Relationship Model - Towards a Unified View of Data", ACM Trans. on Database Systems, Vol.1, No.1, March 1976, pp.9-36

[Co70] Codd, E.F., "A Relational Model for Large Shared Data Banks", Comm. ACM, Vol.13, No.6, June 1970, pp.377-387

[Co72] Codd, E.F., "Further Normalization of t[illegible] Data Base Relational Model", in R.Rustin (e[illegible]), "Data Base Systems", Courant Comp. Sci. Symp., Volume 6, Prentice-Hall, 1972, pp.33-64

[Co74] Codd, E.F., "Recent Investigations in Relational Data Base Systems", Information Processing 74, North-Holland, 1974, pp.1017-1021

[DB78] Dayal, U. and P.A. Bernstein, "On the Updatability of Relational Views", in S.B.Yao (ed.), Proc. Fourth Intl. Conf. on VLDB, West-Berlin, 1978, pp.368-377

[EC75] Eswaran, K.P. and D.D.Chamberlin, "Functional Specifications of a Subsystem for Database Integrity", in D.S.Kerr (ed.), "Very Large Data Bases", (Proc. Intl. Conf. on VLDB), ACM, 1975, pp.48-68

[HM78] Hammer, M. and D.McLeod, "The Semantic Data Model: A Modelling Mechanism for Data Base Applications", in E.Lowenthal and N.B.Dale (eds.), ACM SIGMOD Intl. Conf. on Management of Data, Austin, Texas, 1978, pp.26-36

[NS78] Navathe, S.B. and M.Schkolnick, "View Representation in Logical Database Design", in E.Lowenthal and N.B.Dale (eds.), ACM SIGMOD Intl. Conf. on Management of Data, Austin, Texas, 1978, pp.144-156

[SS75] Schmid, H.A. and J.R.Swenson, "On the semantics of the relational model", in W.F.King (ed.), ACM SIGMOD Intl. Conf. on Management of Data, San Jose, California, 1975, pp.211-223

[SS77] Smith, J.M. and D.C.P.Smith, "Database Abstractions: Aggregation and Generalization", ACM Trans. on Database Systems, Vol.2, No.2, June 1977, pp.105-133

[WE79] Wiederhold, G. and R. El-Masri, "A Structural Model for Database Systems", Stanford University, Computer Science Dept. Technical Report CS-79-722, April 1979

[Wi77] Wiederhold, G., "Database Design", McGraw-Hill, 1977, Chapter 7, pp.329-367