A STRUCTURAL MODEL FOR DATABASE SYSTEMS

by

Gio Wiederhold and Ramez El-Masri

STAN-CS-79-722
February 1979

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

A STRUCTURAL MODEL FOR DATABASE SYSTEMS

Gio Wiederhold
Ramez El-Masri

Computer Science Department
Stanford University

ABSTRACT

This report presents a model to be used for database design, Because our motivation extends to providing guidance for the structured implementation of a database, we call our model the *Structural Model*. Wc derive the design using criteria of correctness, relevance, and performance from semantic and operational specifications obtained from multiple sources These sources typically correspond to prospective users or user groups of the database. The integration of such specifications is a central issue in the development of an integrated structural database model.

The structural model is used for the design of the logical structures that represent a real-world situation. However, it is not meant to represent all possible real-world semantics, but a subset of the semantics which arc important in database modclling,

The model uses relations as building blocks, and hence can be considered as an extension of Codd's relational model [Codd70]. The main extensions to the relational model arc the explicit representation of logical connections between relations, the inclusion of insertion-deletion constraints in the model itself, and the separation of relations into several structural types.

Connections between relations are used to represent existence dependencies of tuplcs in different relations. These existence dependencies arc important for the definition of semantics of relationships between classes of real-world entities. The connections **between relations arc used** to specify these existence dependencies, and to ensure that they remain valid when the database is updated. Hence, connections implicitly define a basic, limited set of integrity constraints on the database, those that identify and maintain existence dependencies among tuplcs from different relations. Consequently, the rules for the maintenance of the structural integrity of the model under insertion and deletion of tuplcs arc easy to specify.

Structural relation types arc used to specify how each relation may be connected to other relations in the model. Relations arc classified into five types: primary relations, referenced relations, nest relations, association relations, and lexicon relations. The motivation behind the choice of these relation types is discussed, as is their usc in data model design.

A methodology for combining multiple, overlapping data models — also called user views in the literature — is associated with the structural model. The database model, or conceptual schema, which represents the integrated database, may thus be derived from the individual data models of the users, Wc believe that the structural model can be used to represent the data relationships within the conceptual schema of the ANSI/SPARC DBMS model since it can support database submodels, also called external schema, and maintain the integrity of the submodels with respect to the integrity constraint6 expressable in the structural model.

We then briefly discuss the usc of the structural model in database design and implementation. The structural model provides a tool to deal effectively with the complexity of large, real-world databases.

We begin this report with a very short review of existing database models. In Chapter 2, wc state the purpose of the model, and in Chapter 3 wc describe the structural model, first informally and then using a formal framework based on extensions of the relational model. Chapter 4 defines the representations wc usc, and Chapter 5 covers the integration of data models that represent the different user specifications into an integrated database model. Formal descriptions and examples of the prevalent cases arc given.

The work is then placed into context first relative to other work (Chapter 6) and then briefly within our methodology for database design (Chapter 7).

CONTENTS

1. CURRENT STATE OF DATA MODELS

Database systems have become a major topic of interest because of their widespread use in industry, commerce, government, and educational institutions [Steel74, Sibley76, Fry76]. Several data models have been proposed to represent the structure of databases. The most widely discussed models arc the relational model [Codd70], the hierarchical model [Tsichritzis76], and the network model (derived from the CODASYL database system specification [CODASYL74]). The majority of implemented database systems use one of the above models. For an excellent introduction to these three database models, see [CompSurv76].

1.1. The relational model:

The relational model is formed from relations. Each relation is composed of a set of structurally identical tuplcs. Tuplcs arc composed of related data elements. For each relation, a relation description, or schema, defines the attributes and the possible values for the data elements that each tuplc in the relation may take. The sets of tuplcs in a relation is described using the mathematical theory of relations, augmented with the concept of functional dependency among attributes. The mathematical basis of the relational model, the uniform representation of all structures as relations, and the syntactic clarity of the data model schema provide important advantages for model and query analysis.

The relational model ha6 been subjected to intensive theoretical scrutiny. Third normal form [Codd72], and Boyce-Codd normal form [Codd74] have been defined to design relations with favorable update properties. Bernstein [Bernstein75] describes an algorithm for synthesis of third normal form relations from functional dependencies. Fagin [Fagin77] introduced multivalued depcndcncics and a fourth normal form for relations to extend the understanding of the logical design of relational databases.

When relations arc built solely from the functional or multivalued dependencies among all attributes in the data model, several possible logical data models can be derived [Bernstein75, Fagin77, Chang78, Delobel78]. Further, some of the data models will not have a direct correspondcncc with the actual real-world situation being modcllcd [Schmid75]. Then the database designer, or some automatic procedure, ha6 to choose the *most* suitable model,

A drawback of the basic relational model is that known relationships among entities of the situation being model arc not explicitly represented but have to be recognized at query processing time by matching attributes that have the same domain. This requires recognition of similar domains, using the schema, as well as some computation within the database to match data elements. Also, logical integrity constraints arc not defined within the model, but arc left to be defined by the database implementors. In one approach, integrity constraints arc described by assertions [Stonebraker74, Eswaran75].

1.2. The hierarchical model:

The hierarchical model represents classes of entities and hierarchical relationships among different entity classes. A class of entities is represented as a record type, and the hierarchical relationships arc represented by a tree structure, with record types as nodes in the tree. The record type represents the attributes of a class of entities, while each record represents a particular entity of the class, and is composed of data items that describe the entity.

Each record is owned by only one record of the record type at the level above it in the tree, and can own in turn any number of records of the record types below it, if any. Many real world situations arc naturally hierarchical, and arc thus well represented by a hierarchical model. In particular, individual user views, or data models, are often hierarchical. Databases used by multiple users often need a more complex model. In the hierarchical model, non-hierarchical relationships arc represented in an awkward and non-symmetric fashion by defining duplicate record types and using pointers.

1.3. The network model:

The network model allows representation of non-hierarchical relationships among entity classes. A record type may be owned by more than one record type, leading to a network rep rcscntation of relationships among entity classes. This permits a direct representation of m:n relationships among entity classes. The concept of a link-set between two record types is introduced. A link-set groups together records of one record type, the member record type, that arc owned by a particular record of a different record type, the owner record type. Existence dependencies to govern occurrences of owner and member records of a link-set arc specified by different types of link-sets, such as manual and automatic.

The database administrator may specify the access structure used for implementing a link-set as a chain of pointers, a pointer array, or he may specify that the records be stored physically adjacent, Thus access to the records in a particular link-set via the owner record can be very efficient. However, the database designer has to recognize and define the link-set and its acccs6 structure a priori, and queries based on structures not directly implemented may be quite costly to process.

A drawback of the network model is that only implemented relationships can be exploited, and that, due to implementation constraints, certain relationships arc difficult to express (such a6 recursive sets [Taylor76], which are relationships between records of the same record type). Another criticism is that it is too implementation oriented, and thus provides limited data independence [Engles69].

1.4. Some other data models:

The problems with the relational, hierarchical and network models have led to active research in data models. Chang [Chang78] has dcvcloped an approach with a 'database skeleton" which includes semantic information about the relationships bctwccn database relations, and defines the relationships over a time frame using the concept of the "state" of the database. The semantic information is used by the system in query translation, and incomplete *or* "fuzzy" queries may be processed. Manachcr [Manacher75] differentiates relationships into several semantic categories. Abrial [Abrial74] goes further by distinguishing every relationship according to its particular semantic notion, but states that his model would bc too complicated for database construction.

Chen [Chen76] has proposed a model based on the relational model which clearly distinguishes relations into two types: entities and relationships among the entities. Integrity rules for logical consistency arc considered for the relation types, but arc not part of the model. Schmid and Swenson [Schmid75] develop the semantics of the relational model, and show that, in the context of their model, relations in third normal form can be differentiated into five semantic types. Rules for insertion and deletion of tuplcs arc given.

More recently, models have been introduced that provide a more detailed semantic description of the situation being modcllcd [Smith77, Hammar78, Navathe78]. In these papers, constructs arc introduced to represent subsets of entity classes in the data model. These subsets have a semantic significance in the data model, such as certain identifying properties that make them different from other entities in the class, The requirement to have a model which describer the data relationships independently of implementation concerns was recognized when standardization of the CODASYL model wa6 suggested. The ANSI/X3/SPARC committee [Steel75] her described a DBMS architecture in response to the perceived long range needs. A principal component of the architecture is the concept ual schema, which is to contain essential information about the database itself. The conceptual schema would be augmented by an internal schema to define the implementation, and by possibly several external schemas to represent the transformations of the database to the views desired by the users.

2. PURPOSE OF THE STRUCTURAL MODEL

The numerous data models presented in the literature have given insight into the process of logical data model design, and the implemented relational, hierarchical and network database systerns have provided experience on both logical and physical database design and implementation. The model presented here is intended to assist in the development of a conceptual data model indepcndcnt of any implementation, but also to provida a framework for database implementation. Wc propose that the model satisfies the criteria [Kent77] for representing the relationships within the conceptual schema of a database system that has an architecture similar to the ANSI/X3/SPARC DBMS architecture.

The structural model which wc present here:

(1) avoids the storage structure dependency and the limitations of the hierarchical and network models,

(2) introduces semantic information to the relational model by the representation of logical connections between relations which also define structural integrity constraint6 in the model itself,

(3) allow6 a precise representation of the semantics of relationships between entity classes, and

(4) provides a framework for the design of a database system starting with the design of individual users data models, to the integration of the data models to form a global database modal, and finally the guidance of the choice of database implementation structures.

Associated with this structural model is a methodology to combine multiple, related data models to form an integrated database model, and to design the data models to match closely the real-world situation being rcprcscntcd. The individual data models also allow the user to specify 6omc of hi6 requirements of the database system.

The model wc present is built from relations, augmented with two additional basic concepts. First wc associate a relation type with each relation. Second wc associate connection types with the relation types which define the structural integrity of this relation with respect to other relations that arc logically related to it in the model. Wc define structural integrity to be the maintenance of a consistent relationship among tuplcs in different relations of the data model as defined by the connections among relations.

During the design and integration process, the relations will be manipulated. To assure manipulatability, wc require all relations to be in Boyce-Codd normal form. However, it is not necessary to build the relations from the functional dependencies between attributes. Rather, as also argued by Chen [Chen76], if wc first define the logical entities and relationships from the real-world, then simple transformations will create a model where all relations arc in third normal form. Once a relation is defined with all its attributes, one can check the functional dependencies between the attributes of the relation. If a relation is not in third normal form, it may be transformed into two or more relations in third normal form [Wiederhold77, sec.7.2]. The structural model prescribes how the data model relation and connection types will represent the entities and relationships of a particular real-world situation, and hancc limits the number of possible data models that may represent a real-world situation.

Wc note here that the structural model is completely independent of implementation considerations. While the structural modal does represent connections between relations, it does not mandate implementation of these connections. Rather, the connections arc used for definition of some logical integrity constraints, An implementation can be chosen based upon an existing relational, hierarchical or network database management system, or possibly by using some other approach.

3. THE STRUCTURAL MODEL

3.1, Real-World Structures:

A database system is used to model some aspect of the real world. People approach real-world data in several phases. First, they observe the situation and collect existing data that describe the situation. Then, from their observations, they classify the data into abstractions. Next, they assess the value of their abstractions in terms of how much it helps them manage the world with a minimum of exceptions. Finally, if they have to implement a system, they describe the real-world situation by a data model, Such a model may be stored on some physical medium (computer or paper files), and used as a guide for data processing. Wc hence introduce a model which can be used to represent the majority of real-world situations rather than a model which may be used to represent all possible real-world semantics.

The main building blocks of the data model arc *classes of entities,* such as PEOPLE, CARS, HOUSES,. . . etc. An entity class is described by the primitive components that arc used to describe each of it6 members, the *properties.* For example, the entity class CARS can have the properties ***LICENSE-NUMBER, COLOR, MODEL, YEAR.*** The properties that identify a specific entity within the entity cla66, in this case the single property ***LICENSE-NUMBER,*** arc called the *ruling* properties. The properties that describe characteristics of an entity, in this case COLOR, MODEL, and ***YEAR,*** arc called the dependent properties.

Associated with each property is a domain, the set of values the it can take in any of the entities that have this property. Some properties may bc repeating. For example, consider the class of entities EMPLOYEES. One of the properties we may represent is the SALARY-HISTORY of an employee. Each employee will have several entries of the salary history, one for each salary he had during hi6 previous employment period. The number of entries is variable from one employee to the next. The SALARY-HISTORY is also an example of a *compound* property, one which is formed of several, more basic, other properties. In this case, SALARY-HISTORY is formed from two more basic attributes, ***YEAR*** and SALARY-VALUE. However, such compound properties can always be decomposed into several *of* the basic properties,

Wc also have to model the relationships that exist bctwccn entity classes. A *relationship* is a mapping among classes. Thus, a relationship defines a rule associating an entity of one class with entities of other (not necessarily different) classes. Most relationships wc encounter arc between two entity classes. An example of such a relationship is CAR:OWNER between the entity classes CARS and PEOPLE. Such relationships may bc 1:1 (for example **COUNTRY:PRESIDENT),** 1:N (for example MANAGER:EMPLOYEE), or M:N (for cxampla STUDENT: CLASS). Other relationships may be among more than two classes. For example, the relationship **SUPPLIER:PART:PROJECT** is among three entity classes SUPPLIERS, PARTS, and PROJECTS.

A relationship between two entity classes has two important characteristics: the cardinality and the dependency . The *cardinality* of a relationship places constraints on the number of entities of one class that can be related to a single entity of the other class. The *dependency* characteristic of a relationship places constraints on whether an entity of one class can exist that is not related to any entities of the other class. Wc will discuss these characteristics more fully in section 4.1.

Finally, some classes of entities may be sub-classes of other entity classes, For example, the entity class EMPLOYEES is a sub-class of the entity class PEOPLE.

The data model should reflect the real-world structure as closely as possible. This makes it easier for the users to understand the model, and allows useful semantic information from the real world to be included in the data model.

In the structural model, relations arc used to represent entity classes, and some types of relationships between entity classes, Other relationships between entity classes arc represented by connections between relations. Relations will be categorized into several types, according to the structure they represent in a data model. Connections bctwccn relations will also be classified into types, and possible connections between relation types arc a part of the model.

Simple properties arc represented by attributes of relations. Wc will always dccompoec compound properties into the simple properties from which they arc formed.

3.2. Relations and Connections:

Relational concepts arc well known, but for conciseness we now define relation6 and relation schemes a6 wc use them in the structural model. Then wc formally define the concept of connections between relations.

In order to define a relation, we first define attributes, tuplcs of attributes, and relation schemes. Relation schemas specify the attributes of a relation. Attributes define the domain6 from which data elements that form the tuplcs of the relation can take values.

Wc will use B, C, D, to denote single attributes; X, Y, Z, to denote sets of attributes; b, c, d, to denote values of single attributes; and, x, y, z, to denote tuples of sets of attributes. For simplicity, wc assume that all sets of attributes arc ordered.

3.2.1. Relations:

Definition 1: An *attribute B* is a name associated with a set of values, $DOM(B)$. Hence, a *value b* of attribute B is an clement of $DOM(B)$.

. For an (ordered) set of attributes Y $= \langle B_1, \ldots, B_m \rangle$, wc will write $DOM(Y)$ to denote $DOM(B_1) \times \ldots$ X $DOM(B_m)$, where X is the cross product operation. Hence, $DOM(Y)$ is the set $\{\langle b_1, \ldots, b_m \rangle \mid b_i \in DOM(B_i)$ for $i = 1, \ldots, m\}$.

Definition 2: A *tuple* y of a set of attributes Y $= \langle B_1, \ldots, B_m \rangle$, is an clement of $DOM(Y)$.

Definition 3: A *relation schema*, R_s, of order m, $m > 0$, is a set of attributes Y $= \langle B_1, \ldots, B_m \rangle$. The *relation*, R, is an instance (or current value) of the relation schema R_s, 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 $K = \langle B_1, \ldots, B_k \rangle$, $k \leq m$, such that every tuplc y in $\mathbf{R}$ ha6 a unique value for the tuplc corresponding to the attribute set K. For simplicity, wc assume the set K is the first k attributes of Y. The *dependent part*, G, of relation schema $R_s (= Y)$ is the set of attributes G $=$ Y- K, where $-$ is the set difference operator.

All relations arc in Boyce-Codd normal form. (For definitions of functional dependcncy and Boyce-Codd normal form, see section 8.1.)

We will write $\mathbf{R}[Y]$ or $\mathbf{R}[B_1, \ldots, B_m]$ to denote that relation $\mathbf{R}$ is defined by the relation schema Y $= \langle B_1, \ldots, B_m \rangle$.

Also, *K(Y)* will denote the ruling part of relation schema Y, and G(Y) will denote the depcndcnt part. Similarly, for a tuplc y in relation $\mathbf{R}$, defined by the relation schema Y, k(y) will denote the tuplc of values that correspond to the attributes *K(Y)* in y, and g(y) will denote the tuplc of values that correspond to G(Y) in y.

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 schema is defined according to the type of the relation (see sec. 3.4).

3.2.2. Connections:

We now define the concept of a connection between two relations, then define the types of connections that are used in the structural model, A connection is defined between two relation schemes, An instance of the connection exists between two tuplcs, one from each relation.

Definition 4: ***A connection*** between relation schemes X_1 and X_2 is established by two sets of ***connecting*** attributes Y_1 and Y_2 such that:

a. $Y_1 \subseteq X_1$.

b. $Y_2 \subseteq X_2$.

c. $DOM(Y_1) = DOM(Y_2)$.

Wc then say that X_1 is connected to X_2 through (Y_1, Y_2).

Two tuplcs, one from each relation, arc ***connected*** when the values for the connecting attributes arc the same in both tuples.

The definition of connection is symmetric with respect to X_1 and X_2, and thus it is an unordered pair.

Connections may be more complex. For example, if wc desire a connection between two sets of attributes with dissimilar, but related, domains, condition (c) above may by changed to DOM($Y_1) = f(DOM(Y_2))$. The function j will relate values of data elements from the two domains. The equality condition in (c) above is the simplest case.

The structural model uses three basic types of connections, which we now define. Associated with each of the connection types arc a set of integrity constraints that define the existence dependency of tuplcs in the two connected relations, These constraints define the conditions for the maintancnce of the structural integrity of the model, We will define structural integrity, and discuss these constraints in section 3.5.

Definition 5: A ***reference connection*** from relation schema X_1 to relation schema X_2 through (Y_1, Y_2) is a connection between X_1 and X_2 through (Y_1, Y_2) such that:

a. $Y_2 = K(X_2)$.

b. $Y_1 \subseteq K(X_1)$, or $Y_1 \subseteq G(X_1)$, but Y_1 may not contain attributes from both $K(X_1)$ and $G(X_1)$.

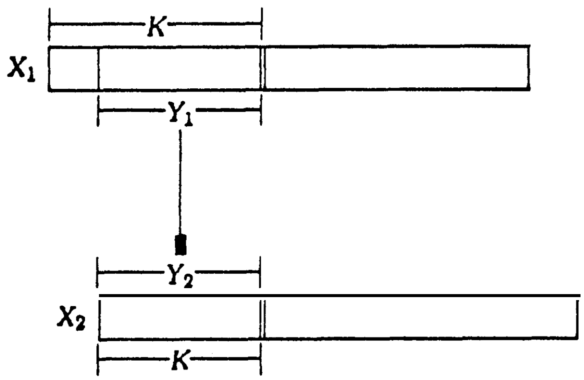
Definition 5a: A reference is an identity ***reference*** if $Y_1 = K(X1)$.

Definition 5b: A reference is a ***direct reference*** if it is not an identity reference.

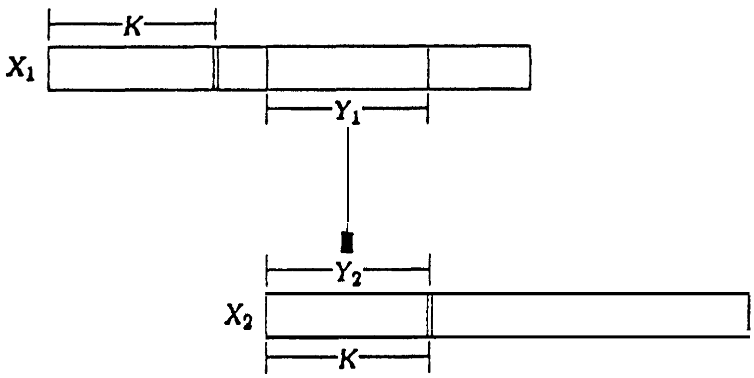
Reference and direct reference are not defined symmetrically with respect to X_1 and X_2, and thus arc ordcrcdcd pairs $\langle X_1, X_2 \rangle$ when the reference is from X_1 to X_2. The identity reference is defined symmetrically, but wc still consider it to be ordered. This is because identity references arc used to represent a subrelation of a relation, defined in section 3.4.2, and wc consider the reference to be directed ***from*** the subrclation to the relation.

Definition 6: An ***ownership connection*** from relation schema X_1 to relation schema X_2 through (Y_1, Y_2) is a connection between X_1 and X_2 through (Y_1, Y_2) such that:

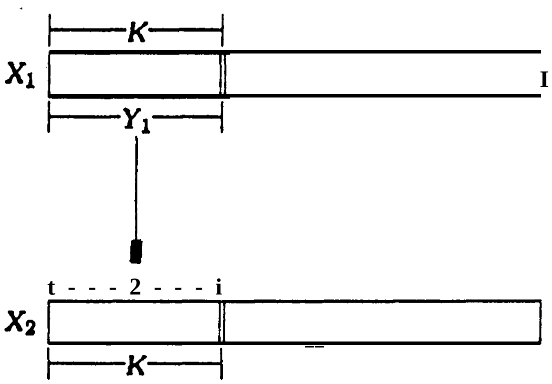
a. $Y_1 = K(X_1)$.



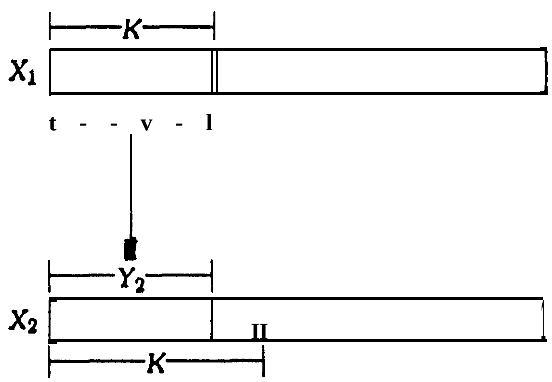
(a) Direct reference $\langle X_1, X_2 \rangle$ from the ruling part of X_1



(b) Direct reference $\langle X_1, X_2 \rangle$ *from* the dependent part of X_1



(c) Identity reference $\langle X_1, X_2 \rangle$



(d) Ownership connection $\langle X_1, X_2 \rangle$

Figure 1. Types of connections

b. $Y_2 \subset K(X_2)$.

The ownership connection is also non-symmetric with respect to X_1 and X_2, and is an ordered pair $\langle X_1, X_2 \rangle$ when the ownership connection is from X_1 to X_2.

The connections defined above may be represented graphically as in figure 1. They are represented by directed arcs, with the ▮ representing the *to* end of the connection. The ruling part attributes in each relation arc marked K, and separated from the dependent part attributes by double lines ($\|$).

3.3. Type8 of relations:

Relations in the structural model arc classified into structural types, which define their interaction with other relations in the data model. Relations can also be classified semantically according to the concept they represent from the real-world situation. One should be careful to distinguish between the semantic and **the** structural role of a relation in a data model,

Semantically, WC distiguish between classes of entities, properties of classes of entities, and relationships among classes of entities. Classes of entities can be represented by several structural relation types, depending upon their relationship with other classes of entities. Hence, entity classes may **be** represented by either primary entity relations, referenced entity relations, or nest relations, as we shall see.

Non-repeating properties of a ***class*** of entities arc represented as attributes of the relation that represents **the** entity class. Repeating properties of a class of entities arc represented by a nest relation owned by the relation that represents the entity class (see section 3.3.3).

Relationships among entity classes can also **be** represented using different structures, **depending** upon the characteristics of **the** relationship. **A** relationship between two entity classes may be represented by an ownership connection, a reference connection, or two connections and **an** auxiliary relation. This auxiliary relation ***may*** be a primary relation, **a** nest relation, or an association relation (see section 4.1).

Structurally, relations arc categorized into five types: primary relations, referenced relations, nest relations, association relations, and lexicon relations. These **are** all relations which **have** the same form, but are classified according to their connections to other relations.

In this section, WC informally present **the** rationale behind **the** choice of **the** different structural relation types. WC give formal definitions for the relation types in section **3.4.**

A relation in the data model which represents a class of entities in the real-world situation is termed an ***entity relation***. The choice of entity classes is a fundamental aspect of the data model design **process.** The goal is to match entity relations as **closely** as possible to real-world entity **classes.**

Structurally, entity relations may be primary, referenced or nest relations. The choice of structural type to represent an entity relation depends upon its role in the data model. In the following three sections, we discuss the criteria for this choice.

3.3.1. Primary entity relations:

An important objective of the data model is to represent real-world entities. The existence of a tuplc in the data model which represents such an entity is hence determined by the existence of the actual entity, independently ***from*** other modclling considerations. Classes of such entities arc represented in the data model by primary entity relations. Examples of primary entity relations are **EMPLOYEES** and CARS, 'Primary entity relations should **be** chosen to **be** update-independent of other relations in the data model. An update of another relation should not require an update of **a** primary entity relation. An update of an entity relation, however, may require updates to other relations connected to it, as WC shall **see** later.

An example of **a** primary entity relation is **the** relation EMPLOYEES in a model that **represents** a company. Updates to the EMPLOYEES relation occur only from outside the database. An employee tuple is inserted whenever a **new** employee is hired by the company, and deleted whenever an employee leaves. This potentially **affects** several other relations in the database such **as** CHILDREN and EMPLOYEES-DEPARTMENT, Thus, insertion of an employee tuplc involves the possible addition of tuplcs to other relations in the database that arc connected to the employee relation, such as tuplcs that represent the employee's children in the CHILDREN relation, and tuplcs associating the employee with **the** departments **he** works for in the **EMPLOYEES-DEPARTMENT** relation. Note that the number of additional tuplcs added to the database because of the insertion of a new primary entity tuple is variable, and determined externally; the model only presents the user with guidelines to follow when inserting a primary entity tuplc.

The deletion of a tuplc from a primary entity relation may imply the deletion of related tuplcs from other relations in the database , Thus, the deletion of an employee tuplc will involve the deletion of tuplcs for his children from the CHILDREN relation, as well as tuplcs associating him with the department he worked in from the EMPLOYEES-DEPARTMENT relation. Such a deletion does not involve any additional checking before the tuplc is deleted, since a primary entity relation may not be referenced by any other relation in the data model.

3.3.2. Referenced entity relations:

When representing a real-world situation, one often encounters abstractions that arc used mainly to describe properties of other entities. Such entities arc referenced by other entities in the model. This type of entity is a *referenced* entity, and classes of such entities arc represented in the data model by referenced entity relations. Examples of referenced entity relations arc CAR MODEL SPECIFICATIONS, referenced by the attribute MODEL in the relation CARS, and JOB DESCRIPTION, referenced by the JOB attribute of the relation EMPLOYEES. The USC of these referenced entities greatly reduces redundancy in the data model. As we shall see, the main difference between a primary entity relation and a referenced entity relation in the structural model is in their update characteristics.

A direct reference connection will exist from some relations in the data model, termed the *referencing* relations, to the referenced entity relation. The reference connection restricts the deletion of tuplcs in the rcfcrcnccd entity relation, as well as the insertion of tuplcs in the referencing relations. WC discuss these restrictions here in terms of an example, and will define them precisely in section 3.4.

An example of a referenced entity relation is presented with respect to a company database. Suppose the company wishes to keep track of current and possible suppliers for inventory items. The SUPPLIERS relation is a referenced entity relation. The existence of supplier tuplcs is determined by a selection from the real-world, since the company maintains a list of its current and possible suppliers. However, a supplier tuplc may not be deleted while it is being referenced from the INVENTORY relation within the data model. Thus, the deletion of tuplcs from a referenced entity relation requires checking the tuples in all relations in the data model which reference this referenced entity relation. Addition of tuplcs to the referencing relation, the INVENTORY relation in this case, is restricted to those tuples that reference an already existing supplier, represented by a tuple in the SUPPLIERS relation in the database. Thus, the name of a supplier for a new inventory item should exist in the SUPPLIERS relation before the new referencing tuplc is added to the INVENTORY relation.

Tuplcs of referenced entity relations may be referenced from more than one relation. For example, the SUPPLIERS relation, may be referenced from the ACCOUNTS-PAYABLE relation, describing unpaid bills, as well as from the INVENTORY relation. Note that supplier tuples may exist which arc not currently referenced from other tuples in the database, but one cannot delete a supplier tuple without checking tuples in all relations that may reference the SUPPLIERS relation.

All other update characteristics for referenced entity relations arc the same as the update characteristics for primary entity relations. In the rest of this paper, when WC USC the term entity relation without qualification, WC will mean primary or referenced entity relation.

3.3.3. Nest relations:

Hierarchical dependencies occur frequently in real-world situations. Hence, real-world entities will be represented in the data modcl whose existence directly depends upon the existence of another entity, For example, in a company database, the **CHILDREN** relation represents children of employees currently working in the company, The existence of children tuplcs in the Company database is justified while their parent works for the company, and the tuplc representing the parent exists in the EMPLOYEES relation. Such entities will be represented in the data model by *nest relations*.

A nest relation always corresponds to a 1:N relationship between two data model relations, the *owner relation* and the nest relation, In our example, the EMPLOYEES relation is said to own the **CHILDREN** relation, This 1:N relationship is represented in the data model by an ownership connection from the owner relation to the nest relation.

For each tuplc in the owner relation, a set of zero or more tuplcs will exist in the nest relation that arc connected to this tuplc. The existence of this set of tuplcs depends upon the existence of the *owner tuple* in the owner relation. The term 'nest relation has been chosen because each owner tuplc will own a 'nest' of tuplcs in the nest relation. The existence of individual tuplcs of the nest is determined by the real-world requircmcnts.

Hierarchical dependencies also occur when a class of entities has a repeating property, where the number of repetitions is variable for each entity in the class. Wc then represent the repeating properties by attributes in a nest relation that is owned by the relation representing the entity class. An example is the education history attributes of on employee in the company database. Here, the EMPLOYEES relation owns the nest relation EDUCATION **HISTORY.** In the structural model, the normalization to first normal form forces the usc of distinct nest relations, but the Connection to the owner relation remains recognized.

Insertion of a tuplc in a nest relation is contingent upon the existence of the owner tuplc in the owner relation. Thus, one may not insert a child or an education history tuplc without a corresponding owner employee tuplc in the EMPLOYEES relation, The deletion of a tuplc from a nest relation is not restricted by the ownership connection. The deletion of a tuplc from the owner relation requires deletion of the nest of tuplcs owned by it in the nest relation. Insertion of tuplcs in the owner relation may involve the creation and insertion of a nest of tuplcs in the nest relation.

3.3.4. Lexicon Relations:

A *lexicon relation* is used to represent a one-to-one correspondence between two act6 of attributes. Most frequently, the one-to-one corrcspondcncc will bc between only two single attributes, but sets of attributes may also bc involved. Examples are the one-to-one correspondence between the two attributes DEPARTMENT-NAME and DEPARTMENT-NUMBER in a company data model, or that between the two sets of attributes ***(INSTRUCTOR,*** CLASS, ***SECTION}*** and ***(ROOM, HOUR, DAYS)*** in a university data model. This one-to-one correspondence reflects a similar correspondence between properties.

Such one-to-one correspondences bctwccn two sets of attributes occur frequently, and isolating lexicons simplifies the data model considerably by transferring attributes that serve the same function into a lexicon relation. One sct of attributes can represent all instances of either set outside of the lexicon itself. Which set of attributes remains in the core of the data model is left to the judgment of the model designer.

The lexicon relation will have a reference connection to it from every relation in the data model that includes either one or both of the sets of attributes in the lexicon. The reference connection may be a direct reference or an identity reference, depending on the situation.

Lexicons serve another important function in the data model, Frequently, relations will have more than one sat of ruling (or *key)* attributes. A set of ruling attributes is guaranteed to have a unique value for any tuplc in the relation, and thus any such set of ruling attributes may be used for tuplc identification, In our model, each relation has one primary set of ruling attributes, the ruling part of the relation. Other equivalent sets of ruling attributes arc transferred to lexicon relations.

The usc of lexicons can greatly reduce the number of possible alternatives for the data model, leading to a significant simplification of the model design process. The two sets of attributes in a lexicon relation can be treated conceptually as a single attribute in intermediate processes which lead to the design of the data model, and can thus bc considered as equivalent *in* the data model. Hence, lexicon relations can be seen as a means of reducing the number of attributes in the core of the data model, leading to the creation of a clearer, simpler model,

3.3.5. Association relations:

Wc finally consider relations used to represent the interaction between two or more relations in the data model. Such relations will bc termed *association relations*. An association relation between two relations associates with each tuplc of one relation a number of tuplcs from the other relation (possibly none). It does not rcprcscnt any existence dependency between the tuplcs in the different relations, but only an association between existing tuplcs.

An association relation of *order* i relates tuplcs from i owner relations. Each of the owner relations has an ownership connection to the association relation.

An example of an association of order 2 is the relation EMPLOYEE-PROJECT which relates an employee to the projects he works in, and vice-versa. Each project tuplc and each employee tuple have an existence of their own, independently from the tuplcs in the association relation. A tuple in the association only relates an employee with a project,

An example of an association of order 3 is the SUPPLIER-PART-PROJECT relation, which relates tuplcs from three owner relations.

An association relation is used to represent information relevant to a relationship between entity classes. Usually, the entity classes arc represented by the i independent relations. Thus, in our example, the EMPLOYEE-PROJECT association may include information about the job the employee does for the project, the percentage of time ho works on the project, . . . etc. It is also possible for association relations to have no dependent information. In this case the association relation is used only for relating tuplcs from the owner relations together,

The update rules for an association relation and its owner relations are now self-evident: no tuplc in the association relation may be created if there arc *no* corresponding owner tuplcs in the owner relations, and deletion of a tuplc from any owner relation causes the deletion of all tuplcs affiliated with it from the association. Note that the deletion rule does not affect the existence of the tuples related to the deleted tuple in the other owner relations: it only affects those tuplcs in the association relation that serve to relate these tuplcs together. Thus, deletion of an employee would not affect the existence of any of the projects hc works for.

3.4. Formal definition of relation types:

In this section, wc formally define the different types of relations discussed in section 3.3 in terms of their connections with other relation types in the data model. Wc then define subrelations of existing relations, and how a subrelation is connected to its base relation in section 3.4.2.

For the remainder of the paper, wc will usc the term relation for both the relation schema and the relation, since the meaning is clear from the context.

3.4.1. Basic relation **types:**

Semantically, relations arc classified into entity and non-entity relations.

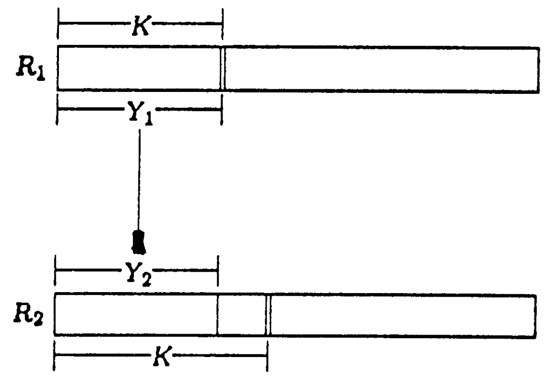


Figure 2. A nest relation, $\mathbf{R_2}$

Definition 7: An *entity relation* is a relation $R[X]$ which defines **a correspondence** between members of **a class** of real-world entities and **the** tuplcs in $R[X]$.

The ruling part of an entity relation **defines** the correspondence to the **class** of real-world **entities,** while the dependent part includes the attributes that describe basic properties of **the entities.**

Structurally, wc define five basic **types** of relations:

Definition 8: A *primary relation* **is a** relation that has no direct references or ownership connections to it from any other relation in the data model.

Primary relations arc required to **have** no **references** or ownership connections to them. Thus, deletion of tuplco from primary relations is unconstrained by **the** data model.

Definition 9: A *referenced relation* is a relation which has direct ***references*** to it from some relations in **the** data model.

The ruling part attributes K(R) of a **referenced** relation, **R,** arc **used** for referencing **R** from **other** relations. Hence, each relation **R'** that references **R** will have a set of referencing attributes that define **the** reference connection to **R.** This constrains insertion and deletion of tuplcs in both **R** and **R'.**

Insertion of **a** tuplc in **R** should prcccdc any reference to it ***from*** a tuplc in a referencing relation. Deletion of a tuplc from **R** involves checking that it is not referenced by any tuplcs from any of the relations that reference **R.** Insertion of a tuplc in **R'** requires the existence of all tuplce that it references.

Definition 10: A *nest relation* is a relation, $\mathbf{R_2}$, which has an ownership connection to it from exactly one other relation, $\mathbf{R_1}$, in **the** data model. $\mathbf{R_1}$ is the *owner* of $\mathbf{R_2}$.

A **nest** relation $\mathbf{R_2}$ has an ownership connection to it from the owner relation, $\mathbf{R_1}$. Hence, the ruling part $K(\mathbf{R_2})$ will consist of two parts: a **set** of attributes to define **the** connection with $\mathbf{R_1}$, and additional attribute(a) which must uniquely identify tuplcs owned by the **same** owner tuplc in $\mathbf{R_1}$.

Insertion of tuplcs in $\mathbf{R_2}$ requires **the** existence of the **owner** tuplc in $\mathbf{R_1}$. Deletion of tuplcs from the **nest** relation may occur based on conditions determined externally from **the** database, but may also be **the** result of deleting an **owner** tuplc from $\mathbf{R_1}$, which requires deletion of all tuplce owned by it in $\mathbf{R_2}$.

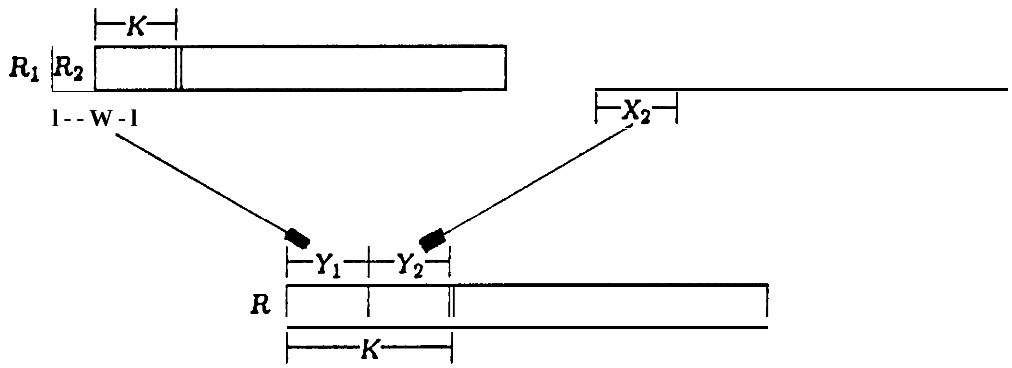


Figure 3. An association relation, R, of order 2

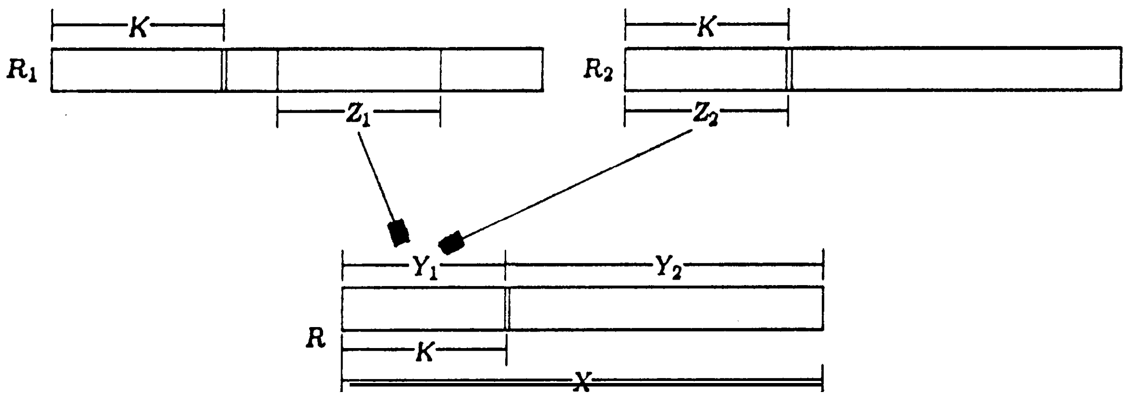


Figure 4. A lexicon relation, $\mathbf{R}[X]$

Definition 11: An ***association relation*** **R** of order $i, i > 1$, is a relation **R** that has i ownership connections to it from i other relations in **the** data model, $\mathbf{R_1}, \ldots \mathbf{R}_i$ **such** that:

a. **each** $\mathbf{R}_j$ has an ownership connection to **R** through $\mathbf{X}_j, \mathbf{Y}_j$ for j - $1, \ldots i$.

b. $\mathbf{Y}_j \bigcap \mathbf{Y}_k = \emptyset$ **for** $j \neq k$.

c. $K(R) = \mathbf{Y}_1 \bigcup \ldots \bigcup \mathbf{Y}_i$.

An association relation of **order** i has i ownership connections to it, one from each of the i owner relations. Hence, **the** domain of the ruling part attributes of an association relation is a **catenation** of i **sets** of attributes, each **set** defining **the** connection to one of **the** owner relations. A tuplc in the association is owned by **one** tuplc from **each** of **the** owner rclatione. For each tuple in an owner relation, there **may** exist **zero, one** ***or*** many **owned** tuplcs in the association.

Deleting a tuplc from an **owner** relation will thus **require** the deletion of all tuplcs owned by it in the association, Insertion of a tuplc in the association will require the existence of the i owner **tuples.**

Definition 12: A ***lexicon relation*** $\mathbf{R}[X]$ **between** two acts of attributes Y_1 and Y_2 defines a **1:1** correspondence **between** $DOM(Y_1)$ and $DOM(Y_2)$ such that:

a. $Y_1 = K(X)$.

b. the set of attributes Y_2 does not appear in any relation other than **R.**

c. $Y_1 \bigcap Y_2 = 0$, and $Y_1 \bigcup Y_2 = X$.

d. **R** is referenced **by one** or **more** relations in the data model by identity or direct references.

A lexicon will have reference connections to it from all the relations in the data model that contain the set of attributes in the lexicon. The ruling part of a lexicon is the attribute set that exists in the other relations in the model, and the dependent part is the other attribute act in the lexicon. For example, if it is necessary to identify the dapartmcnt in several relations of the data model, then either DEPARTMENT-NUMBER or DEPARTMENT-NAME would be chosen. To simplify the model, an arbitrary single choice is made, say to usc the attribute DEPARTMENT-NUMBER in all relations of the model. Then, DEPARTMENT-NUMBER will be the ruling part of the lexicon, and DEPARTMENT-NAME will be the dependent part. Every relation containing the attribute DEPARTMENT-NUMBER will reference the lexicon.

The above definitions define the five structural types of relations: primary, referenced, nest, association, and lexicon. Connections can exist at any level in the model: nest relations can be owned by other nest relations, by associations, or by referenced entity relations as well as by primary entity relations. Similar choices exist for referenced relations, associations, and lexicons. A subrelation may be defined on any relation. In the following section, wc define subrelations.

3.4.2. Subrclationo:

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

Wc will not allow duplication of information in the representation of a subrclation, other than the information needed for tuplc identification, Hence, a subrclation will have the same ruling part attributes as the base relation, and will be connected to the base relation through an identity reference connection. The identity reference reflects the fact that a tuplc in the subrelation that has the same value for the ruling part as a tuplc in the base relation represents the same entity in the data model.

All attributes other than the ruling part attributes of the subrelation have to be different from the attributes of the base relation.

Definition 13: A (non-restriction) *subrelation* of relation $\mathbf{R}[X]$ is a relation $\mathbf{S}[Z]$ such that:

a. an identity reference exists from S to **R**.

b. for every tuplc z in S, there exists a corresponding tuplc x in **R** such that $k(x) = k(z)$.

c. $Z - K(Z) \bigcap X - K(X) = \emptyset$.

The relation **R** is called the *base relation* for subrelation S.

Definition 13a: A *restriction subrelation* of a relation $\mathbf{R}[X]$, restricting the set of attributes Y, $Y \subseteq X$, to the subdomain D, $D \subseteq DOM(Y)$, is a subrclation $\mathbf{S}[Z]$ of **R** such that: for every tuple x in **R** that has as value for the set of attributes Y a tuplc y in D, there exists a corresponding tuplc z in S such that $k(z) = k(x)$.

An example of a restriction subrelation is a relation TECHNICAL EMPLOYEES, a subrelation of the EMPLOYEES relation, restricting the attribute JOB of EMPLOYEES to the subdomain {engineer, researcher, technician}, say.

Existence of tuplcs in a restriction subrclation is totally dependent on the existing tuplcs in its base relation. In our example, all employee tuplcs with job value engineer, researcher or technician must also exist in the TECHNICAL EMPLOYEES subrelation, while all other employee tuplcs cannot exist in this subrelation.

An example of a non-restriction subrelation is a relation EMPLOYEES IN SPECIAL PROJECT X. Existence of tuplcs in this subrclation is determined externally of the data model, but confined to tuplcs in the base relation of all employees.

WC will USC subrelations to represent three cases:

(1) When a subset of a relation has a semantic significance within the data model, or ha6 additional attributes that need to be represented in the model.

(2) When integrity constraint6 require a subset of a relation to own a nest relation or an association, or to be referenced from another relation.

(3) When WC combine data models to form an integrated database model (see section 5), some data models may represent subsets of relations represented in other data models. This ha6 to be reflected in the integrated database model,

The update rules for the base relation and the subrelation arc: when a tuplc that belongs to the subset represented by the subrelation is inserted in (deleted from) the base relation, the corresponding tuplc (having the same ruling part value) is inserted in (deleted from) the subrelation. Also, if an update to a tuplc in the base relation results in the removal of the tuplc from the subset, the corresponding tuplc should be deleted from the subrclation. For example, if the job of an employee tuplc is changed from engineer to manager the corresponding tuplc in the TECHNICAL EMPLOYEES subrclation should bc deleted.

3.5. Maintaining the structural integrity of the data model:

Structural integrity exists in our model when the tuplcs in the data model do not violate the constraints specified by the connections bctwccn relations, One can consider that the structural model contain6 a basic set of integrity assertions as part of the model. The integrity assertions arc those expressed implicitly by the connections between rclatione, and arc used to specify the existence dependencies, and hence the update constraints, of tuplcs in connected relations.

WC do not specify in the model when or how the integrity constraints arc to be maintained in an implementation of the data model, The purpose of the model is that integrity constraints can be recognized, and that implementors can refer for guidance to the model. In practical implementations, there may be intervals where the structural integrity rules do not hold. It should be known however which structural integrity constraint6 have been violated and arc awaiting correction. Hierarchical and network databases tend to require that all integrity constraints be satisfied for those connections that arc actually implemented. Techniques dealing with temporary integrity violation6 using artificial reference tuplcs arc indicated in [Wiederhold77].

Our model may appear less powerful than the original relational model since update integrity violation6 can occur, In the pure relational model, inter-relation connections arc not described, but arc left to be discovered at query-processing time. The lack of recognition of logical connections between relations in a database model will simplify certain technical problems during update, but does not eliminate semantic inconsistencies relative to knowledge models of the database administrator or the user. Furthermore in many situation6 it is best to discover and correct integrity violation6 at the time of update rather than to try and cope with an inconsistent database at query processing time.

In section 3.5.1, WC list the integrity constraint6 specified by each connection type, then give a summary of rules for maintenance of the structural integrity for each of the relation types. WC then show in section 3.5.2 how these rules may be expressed a6 simple algorithm6 for maintaining the structural integrity of the database upon insertion and deletion of tuplcs, and update of attribute values.

3.5.1. Update constraints in the structural model:

The integrity constraints specified by the connection types arc the following:

A ***direct reference connection*** from relation R_1 to relation R_2 specific6 the **constraints:**

(1) Every tuplc in R_1 must reference an existing tuplc in R_2.

(2) Dcletion is restricted for tuplcs in R_2. Only tuplcs that arc not rafarcnced from any relation in the data model may be deleted.

An *ownership connection* from relation R_1 to relation R_2 specifies the constraints:

(1) Every tuplc in R_2 must be owned by an existing tuplc in R_1.

(2) Deletion of a tuplc from R_1 requires deletion of all owned tuplcs in R_2.

An *identity reference connection* from a subrelation R_1 to it6 base relation R specifics the constraints:

(1) Every tuplc in R_1 must reference an existing tuplc in R.

(2) Deletion of a tuplc from R requires dcletion of the referencing tuplc in R_1.

(3) If R_1 is a restriction subrelation, then every tuplc in R that belongs to the subrelation (specified by the value of the restricting attributes in R) must exist in R_1.

WC now give an informal listing of the update constraints associated with each relation type:

1. Primary relation:

(a) The tuplcs arc neither owned nor referenced by other tuplcs in the data model.

(b) Deletion of a tuplc requires the deletion of tuplcs owned by it in nest and association relations.

(c) Insertion of a tuplc requires the existence of referenced tuplcs in the relations referenced by attribute values in the new tuplc.

2. Referenced relation:

(a) The tuplcs arc referenced from other tuplcs in the data model.

(b) The ruling part defines the attributes through which the tuples are referenced by other tuplcs in the data model.

(c) Deletion of a tuplc is constrained by the existence of references to that tuplc. Also, a6 in 1(b)

(d) A6 in l(c)

3. Nest relation:

(a) The tuplcs may be referenced from other tuplcs in the data model.

(b) The ruling part defines a specific owner tuplc, and a specific tuplc within the nest of tuplcs that ha6 the same owncr tuplc.

(c) A6 in l(b). If the relation is referenced, deletion is constrained by existence of references to the tuplc.

(d) Insertion of a tuplc requires the existence of the owner tuplc in the owner relation, and the existence of referenced tuplcs in relations referenced by it.

4. Lexicon relation:

(a) A6 in 2.a.

(b) The ruling part is a set of attributes, through which the tuplc is referenced.

(c) Deletion of tuplcs is constrained by the existence of reference6 to that tuple.

(d) Inecrtion of a tuplc requires no checking.

5. Association relation of order i:

(a) A6 in 3.8.

(b) The ruling part defines i specific owner tuplcs, one from each of the i owner relations.

(c) A6 in 3.c.

(d) Insertion of a tuplc requires the existence of the *i* owner tuplcs in the *i* owner relation, and the existence of referenced tuplcs in relations referenced by it.

6. Subrelation:

(a) A6 in 3.8.

(b) The ruling part attributes arc used for referencing the base relation through an identity reference.

(c) A6 in 3.c.

(d) Insertion and deletion of tuplcs in a restriction subrelation arc totally controlled by existing tuplcs in the base relation.

As indicated earlier, a relation may have more than one connection with other relations in the data modal. A nest relation may for instance itself be referenced, and may also reference tuplcs of another referenced entity relation. In these cases, all connections impose constraint6 on the data model.

3.5.2. Data model update algorithms:

WC now give three simple algorithm6 for maintaining the structural integrity of the data model by observing the constraints given in the preceding section. The algorithm6 will be described in terms of the connection types defined in section 3.2.2.

3.5.2.1. Tuple insertion algorithm:

Upon receipt of a request to insert a new tuplc x in relation R:

a. Check the consistency of the new tuplc with the current tuplcs in the database:

a.1. For every relation R_1 referenced by R through a reference connection, verify that the tuplc y referenced by x exists in R_1.

a.2. For every relation R_1 that has an ownership connection to R, verify that the owner tuplc y of x exists in R_1.

b. If the new tuplc is consistent with the data model, insert it and for every relation R_2 owned by R through an ownership connection, send a message to the user reminding him to insert the tuplcs owned by x in R_2.

Thus insertion involves two actions: checking that tuplcs connected with the new tuplc exist in the data model, and insertion of other tuplcs connected with the new tuplc. The checking can be done automatically, but insertion of other new tuplcs will in most cases be done by the user. For example, the insertion of an employee tuplc involves insertion of his children in a nest relation CHILDREN owned by the EMPLOYEES relation, and of the tuplcs associating the employee with the department he work6 for in the EMPLOYEE-DEPARTMENT association relation, also owned by EMPLOYEES, However, any new tuplcs in both CHILDREN and EMPLOYEE-DEPARTMENT arc inserted by the user. The system only reminds the user that such data may exist, and if they do exist they should bc added to the data model.

In 6omc cases, a6 when a nest relation represents repeating properties of an entity class, an application program can be written to insert all properties of the entity simultaneously. Both a tuplc in the entity relation, and it6 nest of tuplcs that represent the repeating property arc inserted.

3.5.2.2. Tuple deletion algorithm:

Upon receipt of a request to delete tuplc x in relation R:

a. Check for direct references to x from other tuplcs in the data modal: If relation **R** is a referenced relation or a lexicon, check that x is not referenced by any tuplc from a relation with a direct reference to **R**. If x is referenced, send an error **message,** and do not complete the deletion,

b. Check if tuplcs owned by x may **be deleted**: For every relation R_1 owned by R, initiate deletion of the tuplcs in R_1 owned by x. For **every** subrelation R_2 of **R,** initiate deletion of the tuplc y in the subrelation that corresponds to x.

c. If all the owned and **subrelation** tuplcs can **be** deleted, complete deletion of x. Otherwise, do not complete deletion of x, and **send** a warning message that x could not be deleted,

Deletion also consists of two parts: checking that the tuplc being deleted is not referenced, and deleting tuplcs owned by the tuplc being deleted. The algorithm **is** rccureivcly applied.

3.5.2.3. A ttribute update algorithm:

Upon receipt of a request to update attribute A of tuplc x, which belongs to relation R:

a. If A is neither an attribute through which **R** references other rclatione, nor a member of the ruling part of R, perform the update.

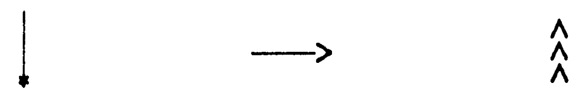
b. Update of connection attributes:

- **b.1.** Referencing attributes: If A is an attribute through which **R** references a relation R_1, check that the new value will reference an existing tuplc in R_1. If the new value references a non-existing tuplc in R_1, do not complete the update and **send** an error message.
- **b.2.** Ruling part attributes: If A is **a member** of the ruling part of **R,** initiate deletion of x using the deletion algorithm. If deletion is completed, insert the updated tuple x_1 with the new value for A using the insert algorithm. Otherwise, send an error **message**.

4. REPRESENTATION OF DATA MODELS

WC now present the guidelines that the structural model presents to a data model designer, and discuss how a choice is made between the different representation form6 provided by the structural model to represent a particular situation. We will see that the same data can be represented with different relationships, according to the situation, or the view of the data model designer. Eventually such differences can be accomodatcd in the integrated database model.

WC use the following notation to represent connections in our diagrams:



Ownership connection Direct reference Identity reference

4.1. Representation of relationships in the structural model:

One of the advantages of the structural model is that it guides the choice of representation for a particular situation. This is because the rules attached to each relation and connection type arc explicit, and will lead the data model designer to carefully consider the situation he is modclling. A model relevant to the real-world situation will be the result, and the situation will be clearly represented.

In the ensuing discussion, WC USC the term relationship to denote *a* relationship between two real-world entity classes, and the term connection to denote a connection between two relations in a data model.

Consider the relationship between two entity classes, FATHERS and CHILDREN. This is a 1:N relationship, and may be represented using several different constructs in the structural model (figure 5):

a. A6 an association between two entity relations representing fathers and children.

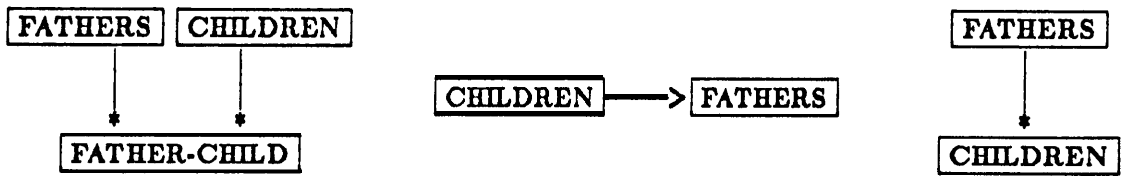
b. A6 a direct reference, from an entity relation representing children, to a referenced entity relation representing fathers.

c. A6 an ownership connection, from an entity relation representing fathers, to a nest entity relation representing children.

The choice among these alternatives depends upon the situation being modcllcd.

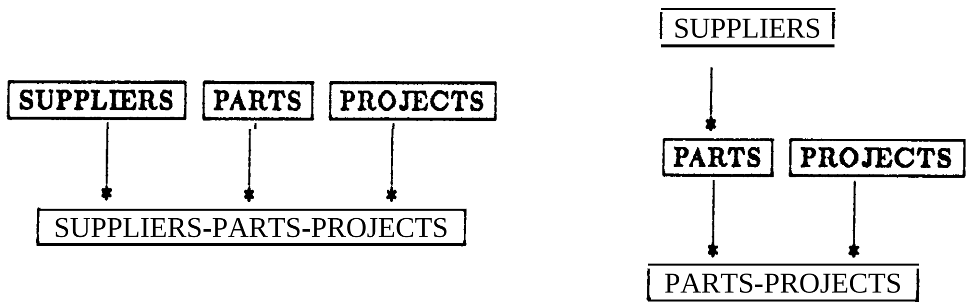
First, consider the case where the data model represents a community of people. Each person in the community ha6 an identity of hi6 own, and WC want to represent the father-child rclationehip between two persons in the community. In this case, the appropriate representation would be as an association between two persons, the FATHER-CHILD association relation (figure 5a). If either the father or hi6 offspring move from the community, there is *no* further need for a father-child connection between two persons in the community, This is well represented in the data model by the association, since deletion of a father (or child) tuplc causes the deletion of the associating tuplc, but leaves the tuplc representing the other person unaffected.

On the other hand, suppose the data model represents data from a school system. In this case, the father-child relationship is best represented by a reference connection from a CHILDREN relation to a FATHERS relation (figure 5b). This restricts the deletion of a father tuplc as long as it is being referenced by a child tuplc. Again, this is a faithful representation of the situation since WC want to keep information on the father as long a6 ha ha6 a child in the school. Also, every



(a) **Association** (b) Reference (c) Nest

Figure **5.** Some representations of the **FATHER:CHILD** relationship



(a) Association (b) **Nest** and association

Figure **6.** Some representations of the **SUPPLIERS:PARTS:PROJECTS** relationship

child in this school must have **some** information about hi6 father. (If the father is unknown, an "unknown father" tuplc could be **placed** within the FATHERS relation.)

Finally, if the data model represents data from a **company,** and a child is represented in the data model only because hi6 father works for **thc** company, then the relationship is best represented a6 a nest relation CHILDREN owned by **the** FATHERS relation (figure **5c).** (In this **case,** FATHERS could be a eubrclation of **the** EMPLOYEES relation.) Then, children arc automatically deleted from the data model once their father is deleted. Here, when an employee is fired (and the decision is made to remove hi6 representation from the active employees file), the company is not interested in any information about hi6 children.

Let is consider a second example, that of an inventory allocation. The situation being **rep**resented is the association **between** suppliers, parts and projects, **If each** of the three entity classes ha6 an independent existence of its own, **the** appropriate representation is an association among three entity relations SUPPLIERS, PARTS and PROJECTS (figure **6a).**

Alternatively, suppose that WC want to associate with **each** supplier the part6 that he supplies, 60 that a part **does** not have an independent existence, but depends on **the** supplier that supplies the part. Then, the situation is **best** represented by two entity relations, SUPPLIERS and PROJECTS, a nest entity relation PARTS owned by the SUPPLIERS relation, and an association relation PARTS-PROJECTS between PARTS and PROJECTS (figure 6b). Note that this represents the full association of **SUPPLIER:PART:PROJECT,** since by the definition of a nest relation, the ruling part of the nest relation PARTS includes the ruling part of the SUPPLIERS relation **(see** section 3.4.1).

These two examples show how the update rules **associated** with each relation type arc used for guidance when designing a data model. The update rules force the data model designer to carefully consider the characteristics of the situation that he is modclling, and thue the data model becomes a faithful representation of the situation.

4.2. Representation of a relationship between two entity classes:

In this section, we consider all possible ways in which the structural model can represent a relationship between two entity classes. This is important for identifying the constraints on relationships. It is also important when we discuss data model integration *in* section 5.

Consider two entity classes, A and B, related in some way, One characteristic of the relationship is its *cardinality.* The cardinality of the relationship restricts 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 bc related to N entities in B, $N \geq 0$, and an entity in B may be related to M entities in A, $M \geq 0$.

Cardinalities may be further constrained by specifying M and N as constant numbers. For example,, a 1:1 relationship is a constrained $1{:}N$ relationship with N set to 1.

The second characteristic of relationships is the *dependency.* The dependency specifics whether an entity of one class can exist independently, or whether it must be related to an existing entity of the other class. Dependencies can be classified into three types:

(a) A *total dependency* specifics that entities in both classes must be related to a specified number of entities of the other class at all times.

(b) A *partial dependency* specifics that entities from one class, entity class *A* say, must be related to a specified number of entities of the other class, B here, but that entities in B can exist independently.

(c) A no *dependency* specifics no dependency constraints.

A direct relationship between the two entity classes A and B may be represented in the structural model a6 one of five choices (figure 7):

(1) A reference connection: entity class A is represented a6 a relation R_a, referencing the relation R_b that represents entity class B (figure 7a). The cardinality of the relationship A:B is $N{:}1$, $N \geq 0$, and the dependency is partial of A on B (each entity in A must be related to exactly one entity in B).

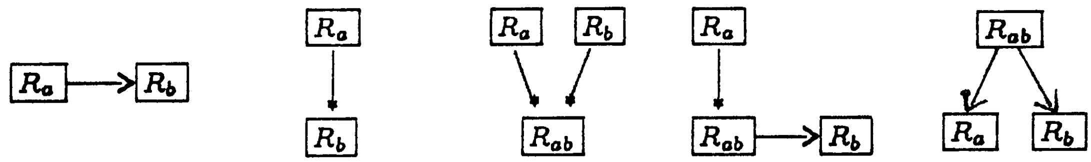
(2) An ownership connection: entity class A is represented by a relation R_a that owns a nest relation R_b representing entity class B (figure 7b). The cardinality of the relationship **A:B** is $1{:}N$, $N \geq 0$, and the dependency is partial of B on A (each entity in B must be related to exactly one entity in A).

(3) An association relation: relations R_a and R_b represent entity classes A and B, and an association relation R_{ab} represents the relationship (figure 7c). The cardinality of the relationship **A:B** is $M{:}N$, $M \geq 0$, $N \geq 0$, and there is no dependency.

(4) A nest of references: relations R_a and R_b represent the entity classes A and **B**. A nest relation R_{ab} owned by R,, and a reference connection from R_{ab} to R_b represent the relationship (figure 7d). The cardinality of **A:B** is $M{:}N$, $M \geq 0$, $N \geq 0$, and there is no dependency.

(5) A primary relation and two reference connections: relations R_a and R_b represent the entity classes, and the relationship is represented by a primary relation R_{ab} and two reference connections from R_{ab} to R_a and R_b (figure 7e). The cardinality of **A:B** is $M{:}N$, $M \geq 0$, $N \geq 0$, and there is no dependency.

Other relationships may exist indirectly. For example, if entity classes *A* and B, and entity classes B and C arc directly related, an indirect relationship exists between entity classes *A* and C. We will only further consider direct relationships in this report.



(a) Reference connection (b) Ownership connection (c) Association relation (d) **Nest** of references (c) Primary relation

Figure **7.** Representing two directly related entity **classes**

Data models **that** represent the same two related entity classes may usc different **repre-**scntations for the relationship according to the way they view the update constraints. Two **reasons** for choosing different representations can **bc** distinguished: difference in understanding and difference in representation. **Wc** illustrate **thc** differences with **an** example.

(1) The two data models differ in their understanding of the **same** real-world situation, Consider the two entity classes DEPARTMENTS and EMPLOYEES. It is possible that one **user** assumes that the relationship between DEPARTMENTS and EMPLOYEES is $1{:}N$ (each employee work6 in only one department). A second **user** is aware of exceptions and considers the relationship $M{:}N$ (an employee may work in more than one department). A disagreement exists **here** about **thc** actual situation being modcllcd, and one of the data models is in error, It may **be** that **the** first **user** knows only about employees that work in one department, If such a conflict occurs between the two data models, the real-world situation being modcllcd must **be** rc-examined to determine its actual characteristic& Wc will not consider this problem further.

(2) The two data models represent the real-world situation differently, each **user** choosing **the** representation which **best** suits his integrity control requirements. **Consider** the DEPARTMENTS and EMPLOYEES example, and suppose **the** relationship is of cardinality $1{:}N$. It may be represented in one of the following ways, among others:

(a) a reference connection from EMPLOYEES to DEPARTMENTS (figure **8a),** .

(b) **an** ownership connection from DEPARTMENTS to EMPLOYEES (figurer **8b,8c),**

(c) **an association** relation restricted to $1{:}N$ (figure **8d),**

(d) **a** nest of references from EMPLOYEES to DEPARTMENTS (figure **8e).**

The different representations reflect different integrity requirements:

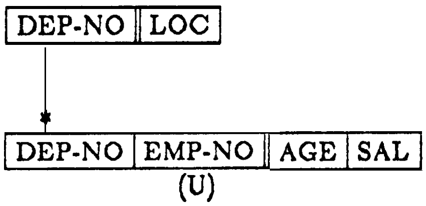
The reference representation requires **each** employee represented in the data model to belong to **a** department, and restricts deletion of a department from the data model while it is referenced by **some** employee,

The ownership connection representation also requires that each employee belongs to a department, but that deletion of a department tuplc from the data model results in the deletion of all the employee tuplcs who work in that department.

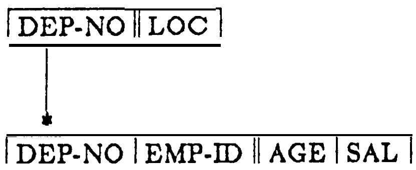
The association does not place any constraints on **the** existence of the actual entities rep resented, the employee and department tuplcs. However, an association can exist only between tuplcs represented in **the** data model.

Finally, the nest of references restricts the deletion of **a** department while referenced by **some** employee, but allows **employee** tuplcs to exist in the data model that arc not related to any department.

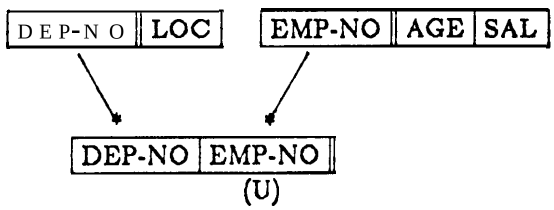
(a)Rcfcrcncc connection



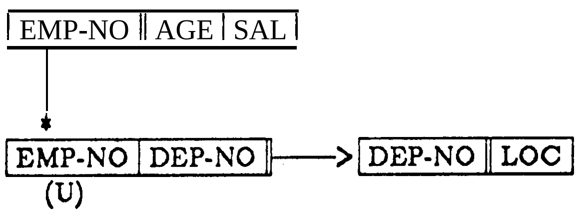
(b)Nest with unique employee identification



(c)Nest with non-unique employee identification



(d)Association



(e)Nest of references

Figure 8. Different representations of the DEP:EMP $1{:}N$ relationship

Since the association representation can be used to represent $M{:}N$ relationships, but here the DEP:EMP relationship is $1{:}N$, the EMP-NO attribute must have a unique value for each tuplc in the association relation, This is indicated in figure 8d by marking the attribute with a (U). Note that this does not violate Boyce-Codd normal form.

The nest of references may also represent an $M{:}N$ relationship, and to restrict it to $1{:}N$, wc also mark the *EMP-NO* attribute in the connecting nest relation by a (U) (figure 8e). Wc will usc this convention throughout the examples in section 5.

In the ownership connection representation, wc must consider two cases. The identifying attribute for each EMP tuplc in figure 8b is *EMP-NO,* and has unique values for each employee independent of his department. Hence, wc mark it (U). In figure 8c, the identifying attributes for an EMP tuplc arc the two attributes DEP-NO and *EMP-ID,* where *EMP-ID* serves to define the employee within his department, and hence is unique within a department but is not unique over all cmploycc6.

The different vicwo may all be equally valid, and hence more than one sat of views, and corresponding rcmantics, has to be retained in the integrated database model so that it can serve in a variety of situations.

WC now consider the problem of integrating different data models, defined by independent user groups and applications, into an integrated database model, to be used as the conceptual schema. WC assume a database system architecture similar to that described by the ANSI/X3/SPARC report.

5. INTEGRATION OF DATA MODELS

Wc now discuss integration of data **models.** First wc briefly define our terminology for logical database design.

A ***DATA MODEL*** is a representation of the requirements of a particular potential database **user** group or application. The definition of data models for individual user groups that expect to usc 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** arc bound to appear, The differences may be resolved by transformations of the original data models, **It** is possible that unresolvable conflicts will emerge among the original data models. Then managcmant **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. Hence, if no **conflicts** occurred between a **user** data model and the integrated database model, the database submodcl for that **user** will **be** the **same** as the data model. If some conflict had arisen, 6omc differences will exist **between** the data model and the database **submodel.**

In section 5.1 wc consider **some** general concepts of data model integration, and in section 5.2 wc consider the integration of **relations** *from* different data models that represent the same real-world entity class. In section 5.3 wc show how to integrate two different representations of a relationship between the same two real-world entity classes.

5.1. Concept8 of integration:

The data models wc integrate will represent real-world situations that partially overlap, otherwise there will **be** no need for integration. **Hence** wc expect to discover relations in separate data models that represent the **same** entity classes. The first phase of integration is to recognize **such** relations, This is not always a simple task, since different data models may usc different names for relation6 that represent the **same** entity class.

Recognition of relations that represent the **same** entity class in different data models is based on matching ruling parts, since **the** ruling part defines **the** correspondence to an entity class. **The** relation names and the ruling part attribute names can provide an initial hint to such **correspon-**dcnccs. **If** data exists, similar values within the ruling part attributes can further indicate candidates for entity matching, A match or overlap of the domain definition of ruling part attribute6 **can** establish **the** necessary equivalence.

Ruling parts may **be** translated via lexicons, **so** the search for similar ruling parts must also consider lexicons of ruling part6 in the data models. Since lexicons preserve the identity of ruling parts, wc will not specify throughout that lexicons can **be used** in the matching of ruling parts. **Some** examples of equivalence through lexicons wil! **be** given in section 5.2.

Wc assume in this report that rigorous definitions exist for the domains that the attributes cover. Definition of domains and attribute encoding can **be** a major effort, but is outside the scope of this report. This problem is also addressed by **people** working *on* the requirements analysis phase of database design.

The second phase of integration, following the recognition of relations that represent the same entity **classes,** is the recognition of differences in the representations. These differences arc of three types:

(1) Representation of different properties of the **same** entity class. **This is** reflected in different dependent part attributes in the relations that represent the **same** entity **class.**

(2) Rcprcscntation of different subsets of entities of the same entity class. **This** is reflected in different tuplcs in the relations that represent the same entity class,

(3) A combination of (1) and (2).

WC will cover integration of those **cases** in section 5.2.

The final phase is to integrate **the** representation of relationships between two entity classes. As shown in section 4.2, there arc five ways to rcprcscnt direct relationships in the structural model. Data models may choose to represent the relationship between the same two entity classes differently, according to their view of the situation. Hence, the final phase of integration is to create an integrated database model which will support different representations of relationships in the data models. WC **cover** this phase in section 5.3.

Many data models may have to **be** integrated into a single database model. To avoid excessive complexity WC will analyze the integration of only two data models in detail. Successive integration steps can merge another data model with the database model being built, creating a new database model. Since both data models and database models USC the same primitives, this should not pose a problem.

We hence have two data models, data model **1** (dml) and data model 2 (dm2). Both data models will include relations that represent some common entity classes, **as well** as other classes of data. WC only look at one entity class A in section 5.2, and two **entity** classes A and B with a relationship **between them** in section 5.3, WC will denote the relations that represent entity classes *A* and B in dml and dm2 by $\mathbf{R}_a$ and $\mathbf{R}_b$. If both representations arc the same, clearly there is no **need** for any transformation, and the integrated database modal (idbm) will USC the **same** representation. If representations differ, WC create an idbm to support both data models.

The idbm will then support database **submodel 1** (dbsml) and database submodcl 2 **(dbsm2)**, corresponding to dml and dm2 respectively. In moat **cases,** dml and dm2 will not **be** changed, so dbsml and **dbsm2** will **be** equivalent to dml and **dm2.** In **some cases,** where conflicts appear, **one** of the data models may have to be changed, and **the** corresponding database submodcl will reflect those changes. When the database model is established, it may also be desirable for pragmatic reasons to change a database submodcl to achcivc a better agreement with **the** database.

In some **cases,** only a subset of the tuplcs in relation $\mathbf{R}_a$ (or $\mathbf{R}_b$) in the idbm correspond to the $\mathbf{R}_a$ (or $\mathbf{R}_b$) relation included in dbsml or **dbsm2.** WC then USC a subrelation to represent the subset, and an identity connection will join it to $\mathbf{R}_a$ in **the** idbm. For example, if $\mathbf{R}_a$ in dbsml corresponds to a subrclation of $\mathbf{R}_a$ in **the** idbm, WC denote this subrelation by $\mathbf{R}_{a1}$ in the idbm, and $\mathbf{R}_{a1}$ will have an identity reference to $\mathbf{R}_a$. This subrelation $\mathbf{R}_{a1}$ of $\mathbf{R}_a$ contains only the ruling part attributes of $\mathbf{R}_a$, so that no duplication of information occurs in the idbm. All other attributes in $\mathbf{R}_a$ can **be** accessed through the identity reference to $\mathbf{R}_a$.

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

After integration, dbsml and dbsm2 arc both supported by the idbm. A mapping will exist from each submodcl to the **idbm.** This mapping includes additional integrity rules, derived from the integration process, which will apply to the idbm. **These** rules are enforced when a database submodcl performs an insertion, deletion, or update. WC will list thcsc additional rules with each **case** of integration.

5.2. Integration of different representations of entity classes:

5.2.1. Recognition of relations that represent the same entity claoo:

This phase of integration requires the recognition of relations included in different data models that represent the same entity class. Knowledge of the real-world situations being modcllcd is helpful to match relations that rcprcscnt the same entity class but have different names for relations and ruling part attributes. The domain definitions of ruling part attributes will then verify the equivalence of such relations by their partial overlap or total match.

Some models may include lexicons of ruling parts for some of the relations in the model. Examination of such lexicons is necessary when matching ruling parts, For example, dml may include a relation EMPLOYEES that contains the attributes (EMP-JVA ME, ADDRESS, *HOME-*PHONE, OFFICE, *OFFICE-PHONE, DEPT*), representing a directory of the employees. Data model 2, representing job information, includes a relation EMP that contains the attributes (*EMP-NUMBER, AGE, JOB,* SALARY, *DEPT),* and a lexicon relation *(EMP-NUMBER, EMP-NA ME)* (figure 9a). To recognize that both relations represent the same entity class of EMPLOYEES, the integrators must consider both the *EMP-NUMBER* and *EMP-NAME* attributes from the lexicon relation in dm2 when matching the ruling part of the EMP relation to the ruling part of the EMPLOYEES relation.

5.2.2. Integration of relations that contain different attributco:

WC first consider the case where one representation dominates the other. Here, dml includes a relation $\mathbf{R_1}$, and dm2 includes a relation $\mathbf{R_2}$ that represents the same entity class as $\mathbf{R_1}$, and contains all the attributes represented in $\mathbf{R_1}$, plus some additional dependent part attributes. The idbm will include a relation $\mathbf{R}$ that contains the set of attributes represented in $\mathbf{R_1}$, and a subrclation $\mathbf{R'}$ of $\mathbf{R}$ that contains the dependent part attributes represented in $\mathbf{R_2}$ but not in $\mathbf{R_1}$. The tuplcs in $\mathbf{R}$ correspond to the $\mathbf{R_1}$ tuplcs in dbsml, while the subset of tuplcs in $\mathbf{R'}$ will correspond to the $\mathbf{R_2}$ tuplcs in dbsm2. When dbsml inserts a tuplc, it is only inserted in $\mathbf{R}$, since it does not contain the dependent part attributes of R'. The tuplc is only visible to dbsml. When dbsm2 inserts a tuplc, it is inserted in both $\mathbf{R}$ and $\mathbf{R'}$, since it contains the dependent part attributes of both $\mathbf{R}$ and $\mathbf{R'}$. Hence, the tuplc is visible to dbsml also.

The general case is that neither relation $\mathbf{R_1}$ of dml nor relation $\mathbf{R_2}$ of dm2 contains the complete set of attributes, but each contains a set of attributes common to both models, and a ect of dependent part attributes unique to its model. In this case, we must create two subrelations. An example is shown in figure 9. Relation $\mathbf{R}$ represents the common attributes, and two subrclations $\mathbf{R_1}$ and $\mathbf{R_2}$ arc used to represent the tuplcs in dbsml and dbsm2 respectively. When dbsml inserts an employee tuplc, it is inserted in $\mathbf{R}$ and $\mathbf{R_1}$, but is invisible to dbsm2. When dbsm2 inserts the tuplc with the same ruling part value, the tuplc is also inserted in R2, and becomes visible to dbsm2. A check has to be performed to ensure that common attributes have the same values, Thus, the base relation $\mathbf{R}$ insures the integrity of data values that arc common to both data models.

The lexicon relation only references $\mathbf{R_2}$, since it is only represented in dbsm2.

If the two data models USC different ruling part attributes, and neither represents the ruling part attributes in the other data model (for example, if in figure lla dm2 did not include the lexicon), then two solutions exist. The first solution is to change one of the data models to include the ruling part attributes of the other data model. The second solution, which involves the database administrator, is to create a lexicon in the idbm in which every new tuplc is included before its insertion by either data model.

WC arc only dealing with the data model here. When actual databases arc to be integrated, inconsistencies may exist in the data, For example, the same employee may have his department relation $\mathbf{R}_1$ (EMPLOYEES)

EMP-NAME	ADDRESS-I	HOME-PHONE	OFFICE	OFFICE-PHONE	DEP-NO

DM1 (directory of employees)

relation $\mathbf{R}_2$ (EMP)

EMP-NO	AGE	JOB	SAL	DEP-NO

(lexicon)

EMP-NO	EMP-NAME

DM2 (job information)

(a)Lcxicon of a ruling part that must be considered

relation R

EMP-NAME	DEP-NO

<<<<

subrelation $\mathbf{R}_2$ (subset in DBSM2)

EMP-NAME	AGE	JOB	SAL

^
^
^ subrclation $\mathbf{R}_1$ (subset in DBSM1)

EMP-NAME	ADDRESS	HOME-PHONE	OFFICE	OFFICE-PHONE

(lexicon, visible to DBSM2 only)

EMP-NAME	EMP-NO

(b)Integrated database model

Figure 9. Integration of different sets of attributes (with lexicon)

listed as 'foundry' in one data model, and as 'management' in another. This problem is a post design issue, although WC note that the structural model would not allow this inconsistency if the different submodcls insert their tuplcs representing the same employee at different times.

WC also note that although many subrclations may exist for the same base relation in the integrated database model, this is only at the model level. At the implementation level, the base relation and all its subrelations may be placed in the same file, with a conditional field for each subrclation in each record to indicate whether the record is in the subrclation or not. It may also be worthwhile to change database submodcls by making them aware of a few additional attributes to simplify the database model.

5.2.3. Integration *of* relations that represent different sets of tuplcs:

WC know consider the case where there arc differences in the selection of entities to be represented. For example, if one data model, dml, includes a relation $\mathbf{R}_1$, and dm2 includes a relation $\mathbf{R}_2$ that represents a subset of the tuplcs in $\mathbf{R}_1$. The idbm will then include a relation $\mathbf{R}$ and a subrelation $\mathbf{R}_2$ of $\mathbf{R}$ to represent the tuplcs in $\mathbf{R}_2$ of dbsm2. The subrelation $\mathbf{R}_2$ may be a restriction subrelation if the subset of tuplcs in $\mathbf{R}_2$ is determined by attribute values in $\mathbf{R}$, or a non-restriction subrelation if the subset of tuplcs in $\mathbf{R}_2$ is determined externally, independent of the model.

For example, dml (for the payroll department) may represent all employees of a company in an EMPLOYEES relation, while dm2 (for the sales department of the company) includes the relation SALES FORCE, the cmployccs that work in the sales department. The idbm then includes a relation EMPLOYEES, and a subrelation SALES FORCE of EMPLOYEES. If the EMPLOYEES relation contains a DEPARTMENT attribute, the subrelation SALES **FORCE** is a restriction subrclation on the DEPARTMENT attribute, restricting the attribute to the value *sales*. If EMPLOYEES does not contain a DEPARTMENT attribute, SALES FORCE would be a non-restriction subrclation. In either case, after integration, dbsm2 is only allowed access to tuplcs in the SALES FORCE subrclation, but could still access their attribute values from the base relation EMPLOYEES, while dbsml would be allowed access to all employee tuplcs.

The general case is that the tuplce in the two relations partially overlap each other. Then dml incldcs relation R_1 and dm2 includes relation R_2 that represent the same entity class, such that the tuplcs in the two relations obey the constraints $R_1 \bigcap R_2 \neq \emptyset, R_1 - R_2 \neq \emptyset$, and $R_2 - R_1 \neq \emptyset$.

The idbm then includes a relation $R = R_1 \bigcup R_2$, and two subrelations of R, R_1 and R_2. Again, R_1 or R_2 could be either restriction *or* non-restriction subrclations, For example, rcfcring to a university database, dml (representing the computer science department of the university) includes a relation CSD PROFESSORS, and dm2 (representing information about permanent faculty) includes a relation TENURED PROFESSORS. The idbm then includes a relation PROFESSORS, and two subrclations of PROFESSORS, CSD PROFESSORS and TENURED PROFESSORS. Each database submodcl is allowed access to his subset, and the base relation assures the integrity of common data represented in both models,

In the last example, it is possible that the relation in each data model contains attributes common to both relations, and a set of it6 own attributes. Then, the base relation in the idbm will contain the common attributes, and each subrclation will contain its own additional set of attributes,

5.3. Integration of different representations of relationships:

In the following sections (5.3.1 - 5.3.4), wc assume that wc have two data models, dml and dm2, and that both data models represent two entity classes A and B, and a relationship between them. R_a and R_b will denote the relations that represent entity classes A and **B**. If the representation of the relationship between A and B involves an auxiliary relation (association, primary or nest relation) wc will designate it R_{ab}.

There arc five ways of representing a relationship between two entity classes in the structural model (section 4.2). Three of these representations arc not symmetric with respect to A and B (reference, nest, nest of references), and two arc symmetric (association, primary). If wc consider all possible combinations without looking at symmetries, the set of possible cases for combining different representations pairwisc is $2 \times (5 + 4 + 3 + 2 + 1) = 30$. Wc remove 5 cases where the representation is identical in both data models, and $(5 + 4)$ cases because the association and primary cases arc symmetric with rcspect to R_a and R_b. Then 18 cases remain to be considered. Wc consider all possible combinations with the association representation first (4 cases) in section 5.1.1. Wc then consider the cases that remain with nest of references (8 cases, section 5.1.2), with references (4 cases, section 5.1.3), and with nest (2cases, section 5.1.4).

5.3.1. Integration with an association:

In thia section, wc consider integration of an association with other representations of a relationship. In those cases, dml represents the relationship A:B as an association relation, and dm2 will usc a different representation, The association may represent a relationship of cardinality $M{:}N$. Our assumption (section 4.2) that both original data models accurately represent the same situation implies that the cardinality of both representations is the same. Hence, the cardinality of the relationship is restricted to the rcprcscntion in dm2.

In order to demonstrate how two different data models may be integrated, wc will present the intcgration of an association with the nest of references (figure 10a).

In this case, the only difference is that dml can freely delete tuplcs from R_b, while in dm2 deletion is restricted by referencing tuplcs from R_{ab}. Hence, wc create two subrclations, R_{b1} and R_{ab1}. Those subrclations represent the tuplcs in R_b (and R_{ab}) of dbsml. Tuplcs in R_b and R_{ab} in the idbm may include some tuplcs deleted from dbsml, but not deleted from R_b and R_{ab} in the idbm due to the deletion constraint of the rcfcrcncc in dbsm2. These tuples arc not visible to dbsml.

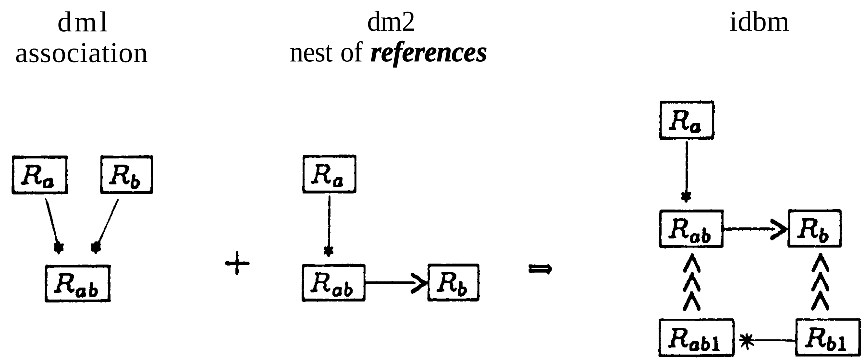


Figure **10a**. Integration of association and **nest** of references

The database submodcls now obey the following rules. Insertion and deletion in R_a from **either** dbsml or **dbsm2** is unrestricted, as is deletion of R_{ab} tuplcs, and unreferenced R_b tuplcs. If dbsml deletes a referenced R_b tuplc (**dbsm2** may not perform such a deletion), it is only **deleted** from R_{b1} (and **the** owned tuplcs arc deleted from R_{ab1}). **These** rules accurately ***reflect*** the constraints imposed by the views represented in the original data models.

For brevity, WC will USC the following format for **each** integration **case.** WC first list **the** differences between **the** two data models, **then** list **the** additional integrity constraints that **have** to exist in **the** mapping from **the** database submodcls to **the** integrated database model. **When** listing these additional constraints, ('relation name") will mean: do the insertion or deletion specified on **"relation"** if allowed by **the** integrity constraints of the idbm.

WC will now present the demonstration case again in brief notation.

(a) **ASSOCIATION AND NEST OF REFERENCES(figure 10a):**

Differences:

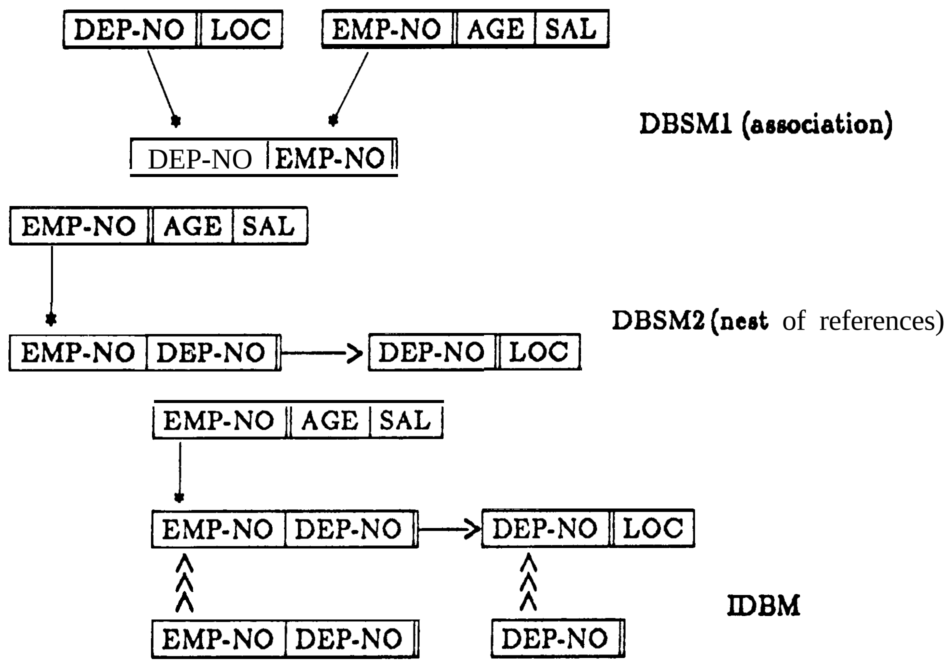
Dml may **freely delete** tuplcs from R_b, while in dm2, deletion of R_b tuplcs is restricted.

Additional constraints:
dbsml:
insert: (1) R_b - R_b,R_{b1}, (2) R_{ab}- (R_{ab},R_{ab1})
delete: (1) R_b - (R_b),R_{b1}

dbsm2:
insert: (1) R_b - R_b,R_{b1}, (2) R_{ab} - (R_{ab},(R_{ab1}))

The relation **name** to **the** left of the **"-"** refers to **the** database submodel, while those to the right **refer** to **the** database model. WC only consider **cases** which **need** additional control from **the** constraints. Insert in R_a of dbsml hence means insert in R_a of the idbm, since it is not listed. In dbsml, insert in R_b requires insertion of **the** tuplc in both R_b and R_{b1} of **the** idbm. Insert in R_{ab} requires insertion in (R_{ab},R_{ab1}) in the idbm, **the** ( ) brackets meaning if **the** integrity check of the idbm will allow it, **here** if both owner tuplcs exist. In **dbsm2,** insert in R_{ab} requires insertion in (R_{ab},(R_{ab1})), which means: insert the tuplc in R_{ab} if the integrity check of the ibdm holds (here both **the** owner tuplc in R_a and **the** referenced tuplc in R_b exist), than insert the same tuplc in R_{ab1} (if the other owner tuplc exists in R_{b1}).

Following each integration case, WC will give an example with attributes to illustrate the integration process. Example **1** illustrates the integration of association and **nest** of references.



Example 1

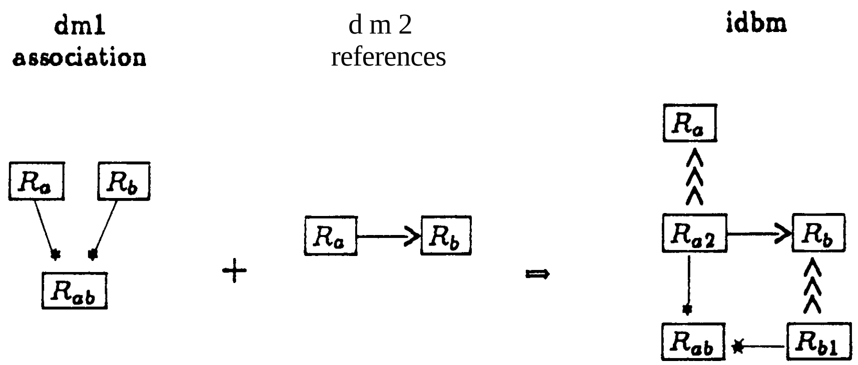


Figure **10b**. Integration of association and reference

(b) **ASSOCIATION AND REFERENCE(figure 10b):**

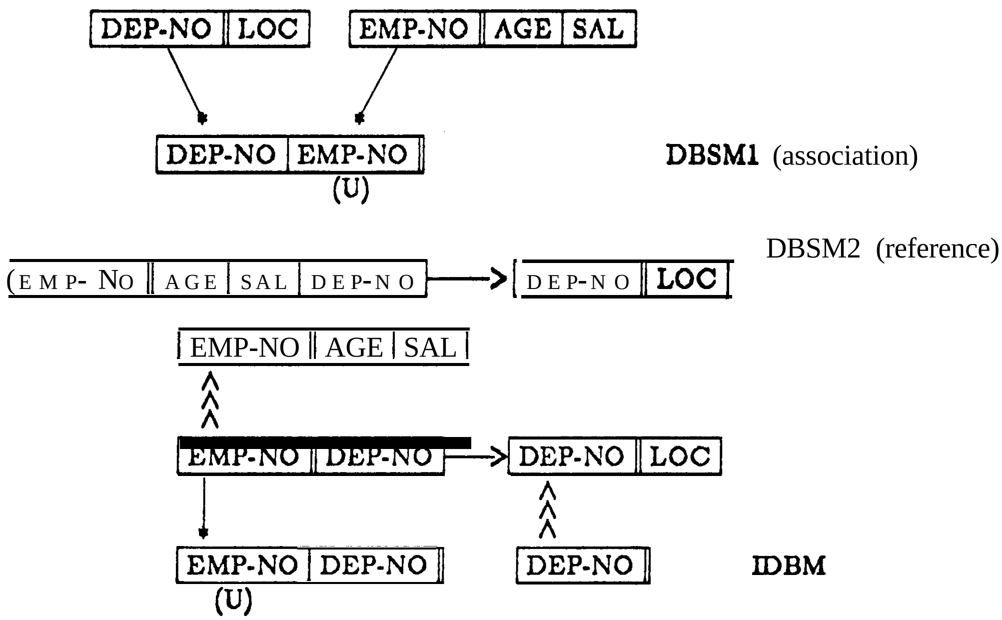
The cardinality **of** the relationship **A:B** is restricted **to** N**:1, since** the **reference cannot represent an** M**:**N relationship,

Differences:

(1) In dm2, every $\mathbf{R_a}$ tuplc must rcfcrcncc an $\mathbf{R_b}$ tuplc, while in dml not **all** $\mathbf{R_a}$ **tuples have** to be associated with $\mathbf{R_b}$ **tuples.**

(2) In dm2, deletion of $\mathbf{R_b}$ tuplcs is restricted by references.

Additional constraints:
dbsml:
insert: (1) $\mathbf{R_b - R_b, R_{b1}}$**, (2)** $\mathbf{R_{ab} - (R_{a2}, R_{ab})}$
delete: (1) $\mathbf{R_b - (R_b), R_{b1}}$**, (2)** $\mathbf{R_{ab} - R_{a2}, R_{ab}}$



Example 2

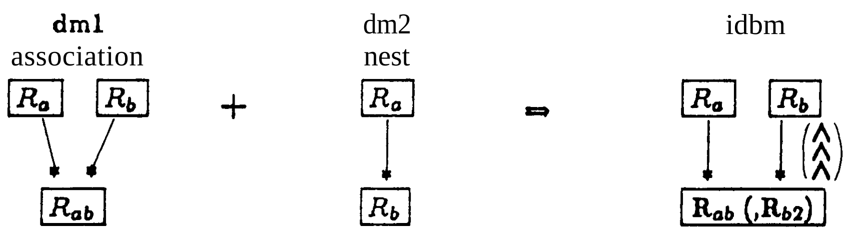


Figure 10c. Integration of association and nest

dbsm2:
insert: (1) R_a - (R_a,R_{a2},(R_{ab})), (2) R_b - R_b,R_{b1}

The requirement that every R_a tuplc must reference an R_b tuplc in dm2 leads to the creation of the subrelation R_{a2}, while the unrestricted deletion of R_b tuplcs in dml leads to the creation of R_{b1} (example 2).

(c) ASSOCIATION AND NEST(figurc 10c):

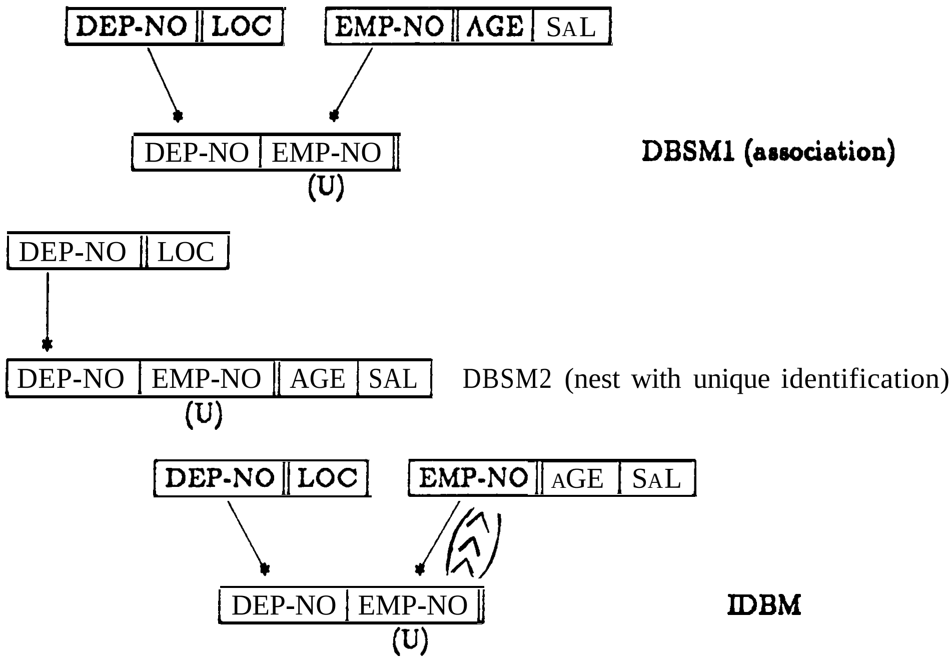
The cardinality of the relationship **A:B** is restricted to **1:*N***, since the ownership connection **can** only represent **1:*N*** relationships.

Diff **erences:**

(1) In dm2, existence of a tuple in R_b requires **the** existence of the owner tuplc in R_a, while in dml, R_b tuplcs can exist independently.

(2) In dm2, deletion of a tuplc from R_a **requires** the deletion of the owned tupler in R_b, while dml does not require these deletions.

Additional constraints:
dbsm2:
insert: R_b - (R_b,R_{b2})



Example 3

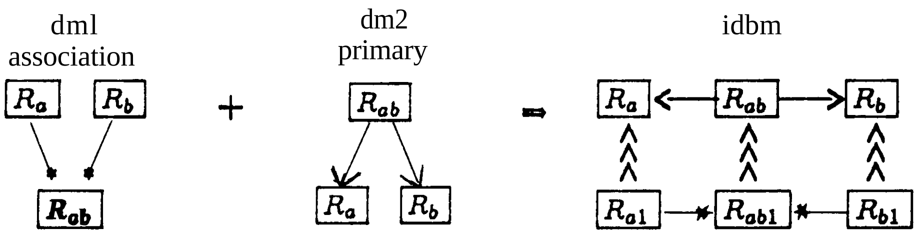


Figure **10d.** Integration of association **and** primary

The $\mathbf{R}_b$ tuplcs of **dbsm2** arc only those in $\mathbf{R}_{b2}$ in **the** idbm, since **they** require the existence of **the** owner tuplc. In **the** idbm, $\mathbf{R}_{ab}$ will also represent **the subset of** $\mathbf{R}_b$ tuplcs in $\mathbf{R}_{b2}$.

Here, wc must consider two examples, since **the nest relation** may represent different tuplc identification attributes than **the** association. First, wc consider **the case** where the identification is the same. In example 3, EMP-NO identifies the employee in both dbsml and dbsm2. Since the cardinality of DEPARTMENT:EMPLOYEE is $\mathbf{1}:N$, the EMP-NO attribute must have unique **values** in tuplcs of **the** relations marked (U). Note that this **does** not violate Boyce-Codd normal form. In this **case,** the integration is straightforward.

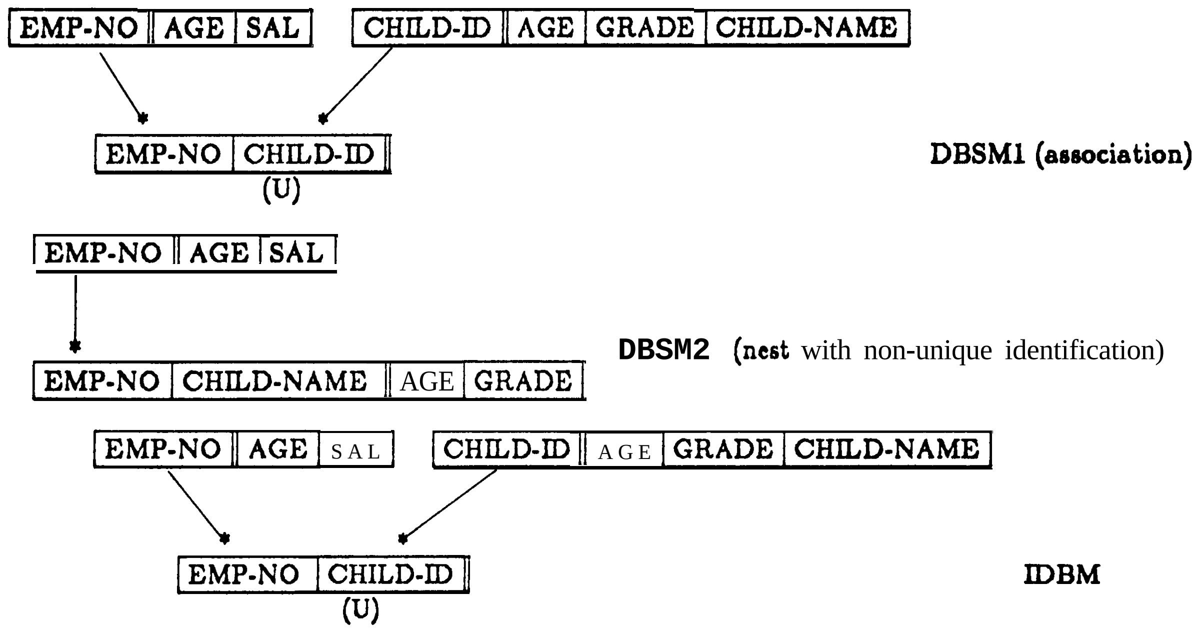
In **example** 4, the identifying information is different. Dbsm2 uses the two attributes (***EMP-NO, CHILD-NAME)*** as ruling part, **while** dbsml **uses** only ***CHILD-ID. CHILD-ID*** uniquely identifies every child tuplc, **but** CHILD-NAME **does** not. **Here,** if dbsm2 does not represent the attribute ***CHILD-ID,*** he has to be made aware of it to maintain the correct mapping between ***CHILD-ID*** and ***CHILD-NAME*** on **insertion** of child tuplcs.

(d) ASSOCIATION AND **PRIMARY(figure 10d):**

The cardinality of the relationship **A:B** is $M{:}N$.

Differences:

In dm2, deletion of $\mathbf{R}_a$ and $\mathbf{R}_b$ is restricted by references



Example 4

Additional constraints:
dbsm1:
insert: (1) R_a - R_a, R_{a1}, (2) R_b - R_b, R_{b1}, (3) R_{ab} - R_{ab}, R_{ab1}
delete: (1) R_a - (R_a), R_{a1}, (2) R_b - (R_b), R_{b1}
dbsm2:
insert: (1) R_a - R_a, R_{a1}, (2) R_b - R_b, R_{b1}, (3) R_{ab} - R_{ab}, R_{ab1}

5.3.2. Integration with *a* nest *of references:*

Now wc consider the cases that remain with nest of references. Dm1 represents the relationship **A:B** as a nest of references, and dm2 represent it differently. The cardinality of the nest of reference representation is $M{:}N$, but may again be restricted by the representation in dm2. The nest of reference representation is not symmetric with respect to entity classes A and B, and so wc must consider it twice with each non-symmetric rcprcscntation.

(a) NEST OF REFERENCES AND NEST OF REFERENCES(figure 11a):

Differences:
(1) Deletion of R_b (R,) is restricted in dm1 (dm2).
(2) Deletion of R_a (R_b) in dm1 (dm2) requires deletion of owned tuplce in R_{ab} (R_{ba}).

Additional constraints:
dbsm1:
insert: (1) R_a - R_a,R_{a1}, (2) R_b - R_b, R_{b2}, (3) R_{ab} - (R_{ab},R_{ab1},(R_{ba2}))
delete: (1) R_a - (R_a),R_{a1},(R_{ab}), (2) R_b - (R_b,R_{ab}).

dbsm2:
insert: (1) R_a - R_a,R_{a1}, (2) R_b - R_b, R_{b2}, (3) R_{ba} - (R_{ab},R_{ba2},(R_{ab1})).
delete: (1) R_a - (R_a,R_{ab}), (2) R_b - (R_b), R_{b2},(R_{ab}).

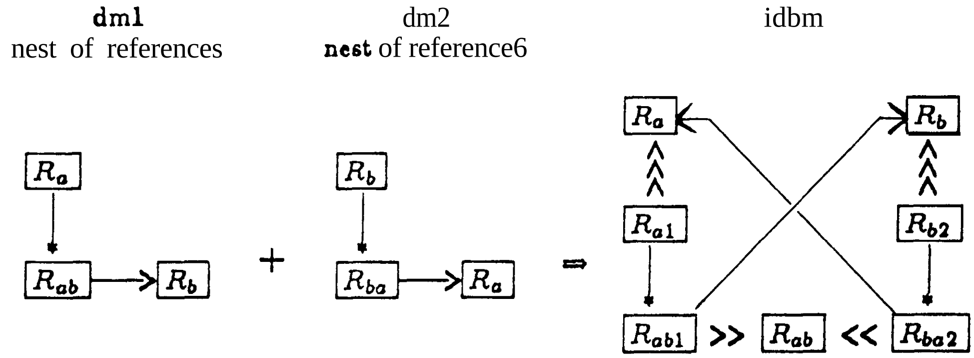
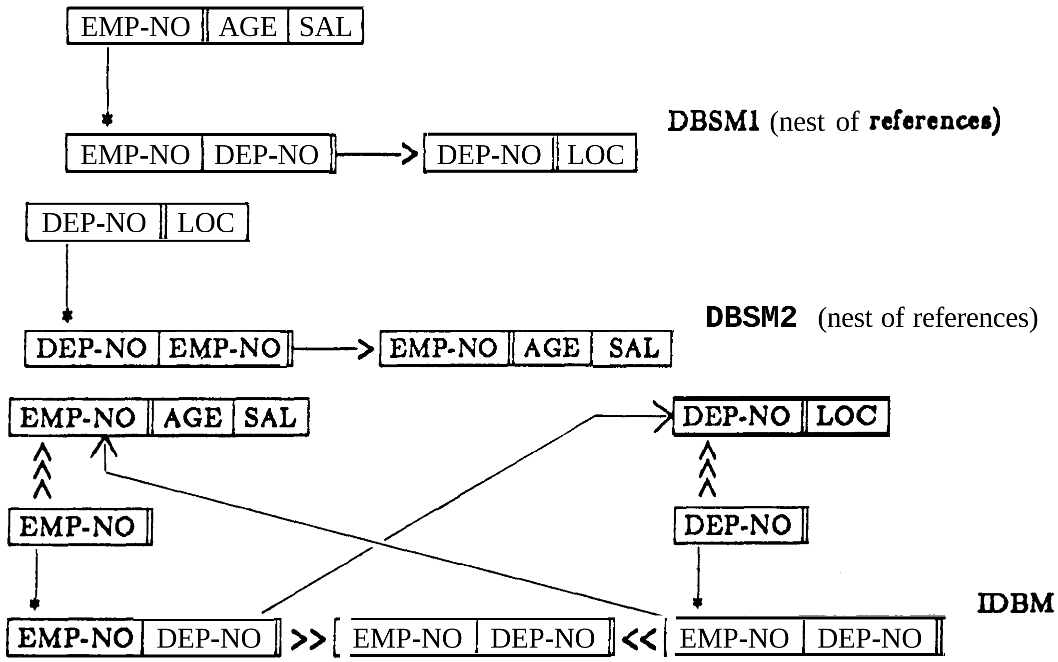


Figure 11a. Integration of nest of references and nest of references



Example 5

When dbsml tries to delete an R_a tuplc in the idbm that is referenced from R_{ba2}, it is only deleted from R_{a1}. If the tuplc is not referenced from R_{ba2}, the tuplcs in R_{ab} that correspond to those deleted from R_{ab1} (due to the deletion of R,) should also be deleted, since they no longer exist in either R_{ab1} or R_{ab2}. R_{ab} exists to ensure that the tuples associating tuples from R_a with tuplc from R_b arc consistent.

Example 5 illustrates this case.

(b) NEST OF REFERENCES AND REFERENCE(figure 11b, 11c):

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

Case 1 (figure 11b):

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

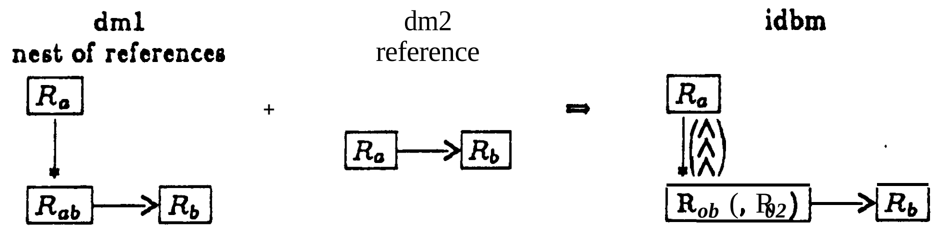
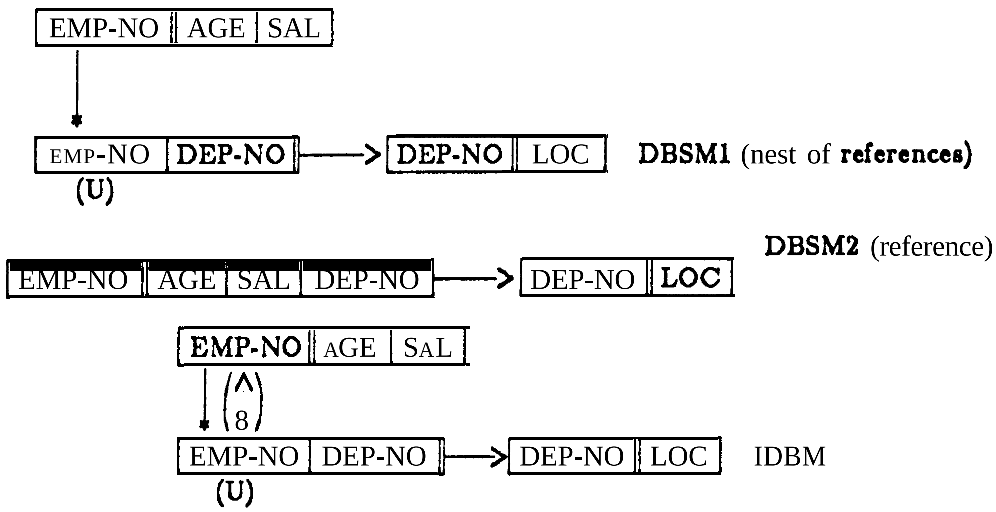


Figure 11 b. Integration of nest **of references** and reference (Case 1)



Example 6

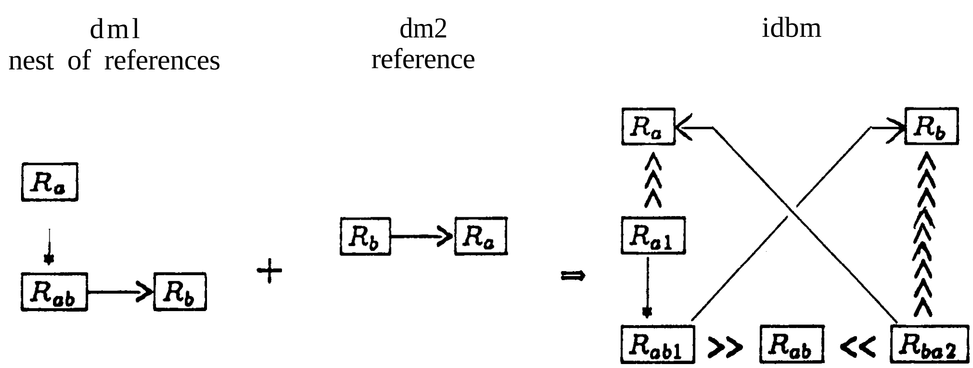


Figure **11c.** Integration **of** nest of references and reference **(Case 2)**

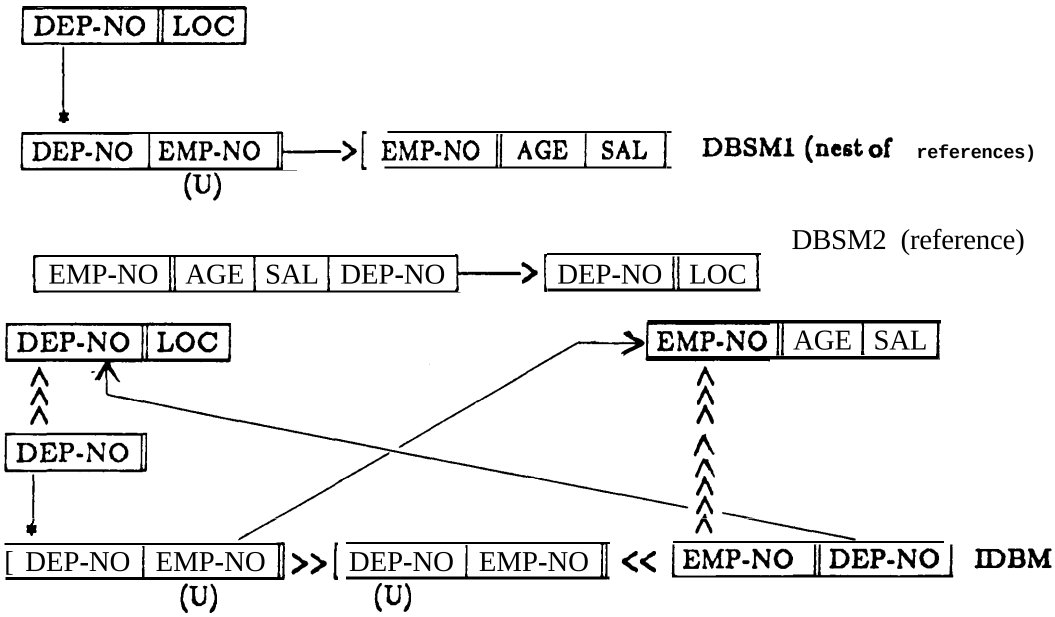
Differences:

A **tuple** in $\mathbf{R_a}$ in dm2 must be associated with an $\mathbf{R_b}$ tuplc.

Additional constraints:
dbsm2:
insert: $\mathbf{R_a}$ - ($\mathbf{R_a}$,$\mathbf{R_{a2}}$)

Case *2* (figure **11c):**

Again, **the** cardinality of **the** relationship **A:B** is restricted to **1:*N*.**



Example 7

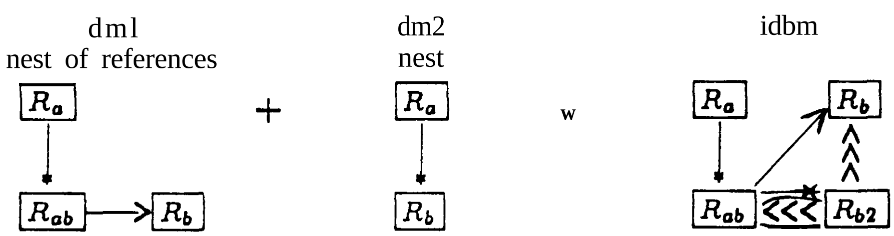


Figure **11d.** Integration of nest of rcfcrcnces and nest **(Case** 1)

Differences:

(1) Deletion of R_b (R_a) **tuples** is restricted in dml **(dm2).**

(2) Every R_b tuple in dm2 must be related to an R_a tuple.

Additional constraints:

dbsml:

insert: (1) R_a - R_a, R_{a1}, (2) R_{ab} - $(R_{ab1}, R_{ab}, R_{b2})$.

delete: (1) R_a - $(R_a), R_{a1}, (R_{ab})$, (2) R_b - (R_b, R_{ab}), (3) R_{ab} - $R_{ab1}, (R_{ab})$.

dbsm2:

insert: (1) R_a - R_a, R_{a1}, (2) R_b - $(R_b, R_{b2}, R_{ab}, R_{ab1})$.

delete: (1) R_a - (R_a, R_{ab}), (2) R_b - $(R_b, R_{ab}), Rb_2$.

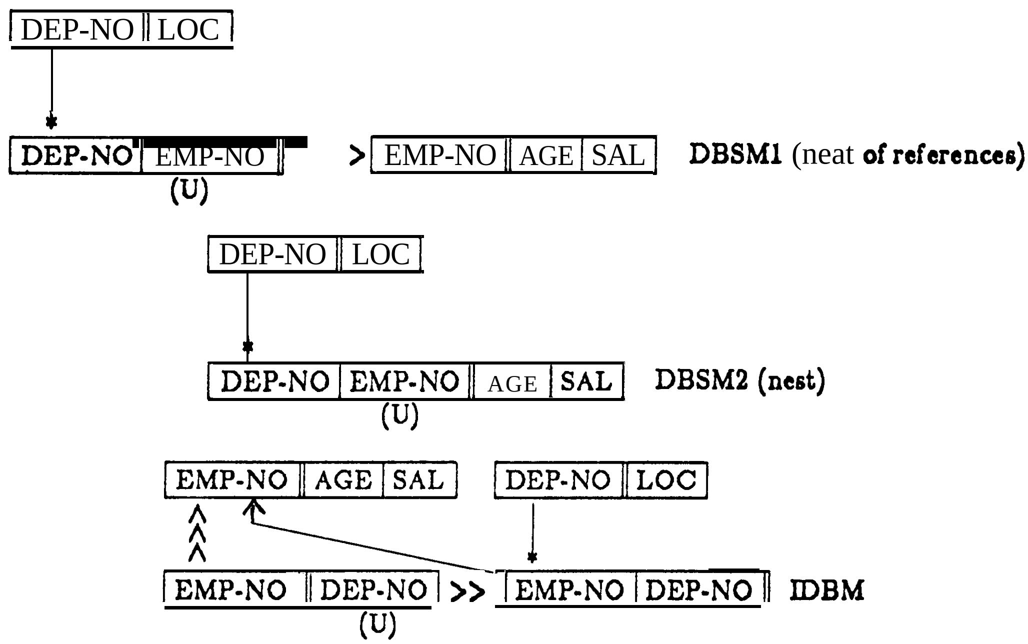
Example **7** illustrates this **case.**

(c) NEST OF REFERENCES AND NEST(figurc **11d, 11e):**

Again, both **nest** of **rcfcrcnces** and nest arc non-symmetric, **so** WC must examine two **cases.**

Case 1 (figure **11d):**

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



Example 8

Differences:

(1) $\mathbf{R}_b$ tuplcs may exist independently in dml.

(2) Deletion of $\mathbf{R}_b$ tuplcs is restricted in dml.

Additional constraints:
dbsml:
insert:. $\mathbf{R}_{ab}$ - $(\mathbf{R}_{ab},\mathbf{R}_{b2})$.

dbsm2:
inscrt: $\mathbf{R}_b$ - $(\mathbf{R}_b,\mathbf{R}_{ab},\mathbf{R}_{b2})$.
delete: $\mathbf{R}_b$ - $(\mathbf{R}_b),\mathbf{R}_{b2}$.

WC again consider two examples, because of the different ways the nest relation may represent the tuplc identifying information. In example 8, WC consider the case where the identifying information is the same.

In example 9, WC *now* consider the case where the identifying information is different. Here, WC must slightly change dm2 by introducing an additional attribute.

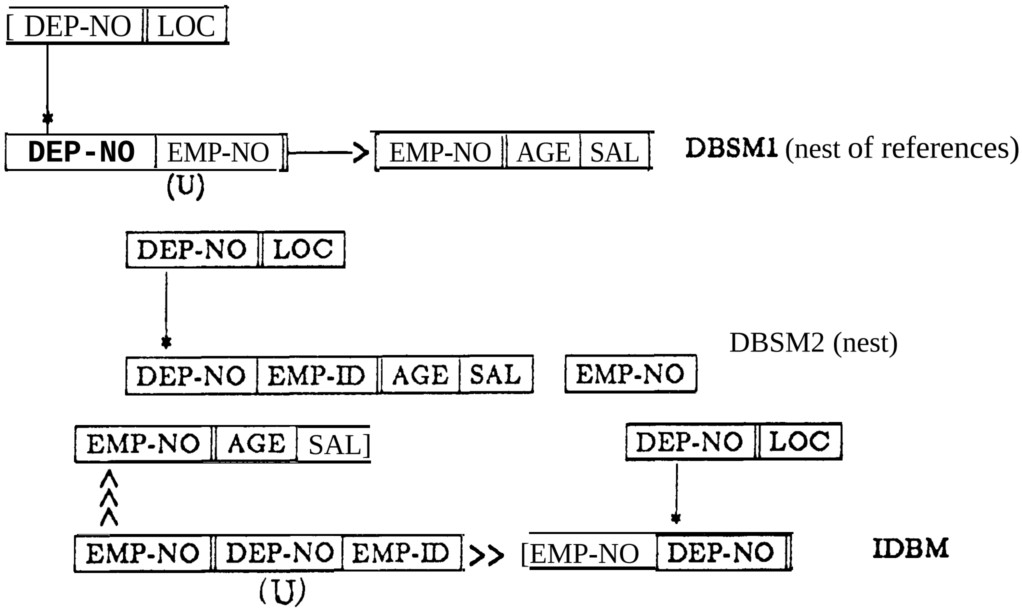
Case 2 (figure llc):

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

Differences:

(1) In dml, $\mathbf{R}_a$ tuples can **exist** independently, while in dm2 an owner tuplc $\mathbf{R}_b$ tuplc must exist.

(2) In dml, deletion of $\mathbf{R}_b$ tuplcs is restricted by **references,** while in dml, deletion of an $\mathbf{R}_b$ tuplc requires deletion of related $\mathbf{R}_a$ tuplcs.



Example 9

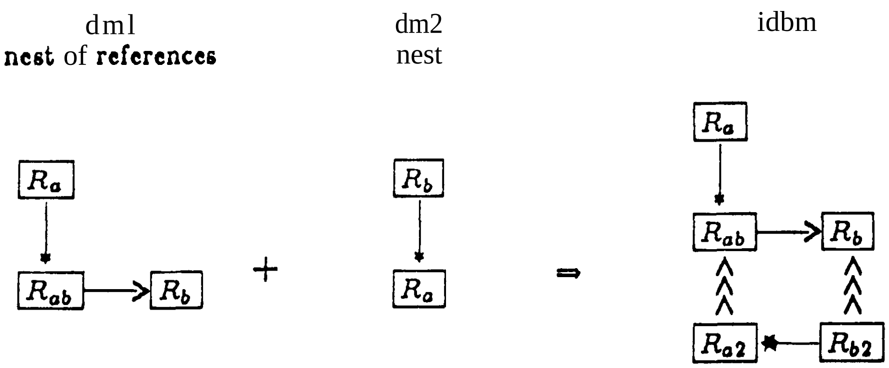


Figure **11e. Integration** of **nest** of rcfcrcnccs and nest (Case 2)

Additional constraints:

dbsml:

insert: (1) R_b - R_b, R_{b2}, (2) R_{ab} - (R_{ab}, (R_{a2})).

dbsm2:

insert: (1) R_a - (R_a, R_{a2}, R_{ab}), (2) R_b - R_b, R_{b2}.

delete R_b - (R_b), R_{b2}.

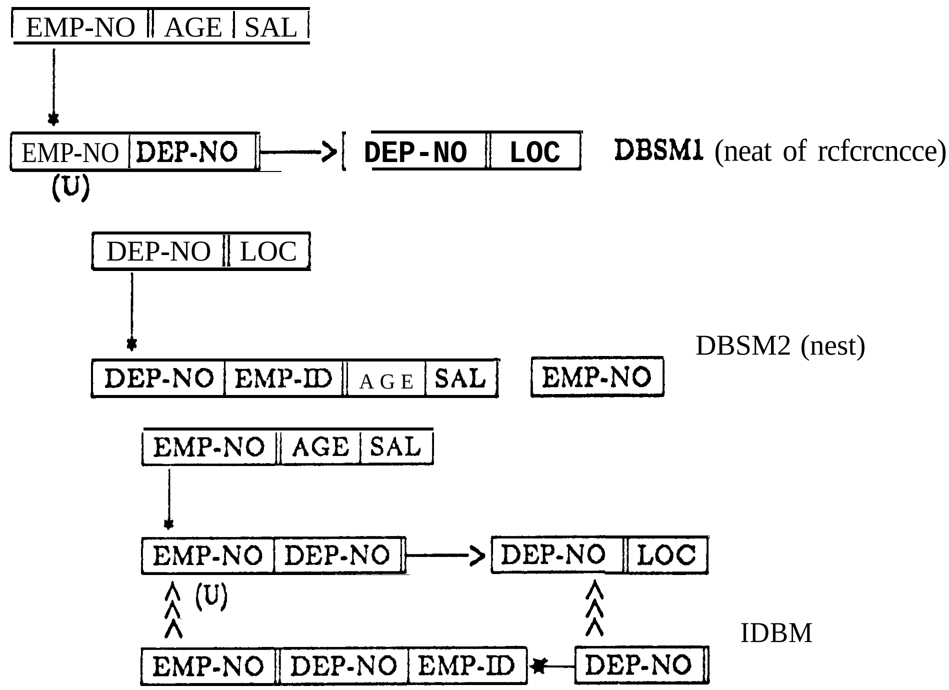
WC will only consider **one example** for this **case,** example 10, with different identification.

(d) NEST OF REFERENCES AND PRIMARY(figure 11f):

The **cardinality** of **the** relationship A:B is $M{:}N$.

Differences:

In dm2, deletion of R_a is restricted by rcfcrcnccs



Example 10

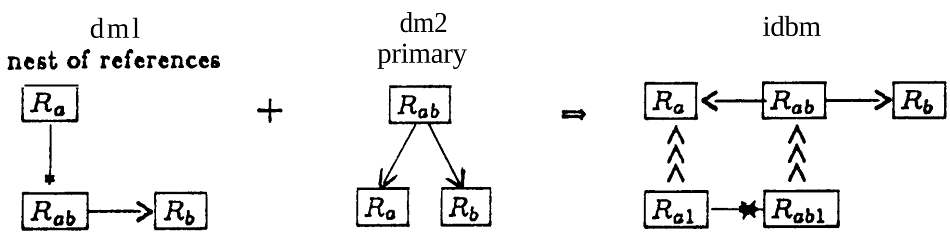


Figure 11f. Integration of nest of rcfcrcnccs and primary

Additional constraints:
dbsml:
insert: (1) R_a - R_a, R_{a1}, (2) R_{ab} - R_{ab}, R_{ab1}
delete: (1) R_a - (R,), R_{a1}
dbsm2:
insert: (1) R_a - R_a, R_{a1}, (2) R_{ab} - R_{ab}, R_{ab1}

5.3.3. **Integration with a reference:**

Dml represents **the relationship A:B as a** rcfcrcncc connection from R_a to R_b, **and** dm2 represents it using a different structure. **The** cardinality of the relationship **A:B** is N:1, **possibly** restricted by the dm2 representation.

(a) REFERENCE AND REFERENCE(figurc **12a**):

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

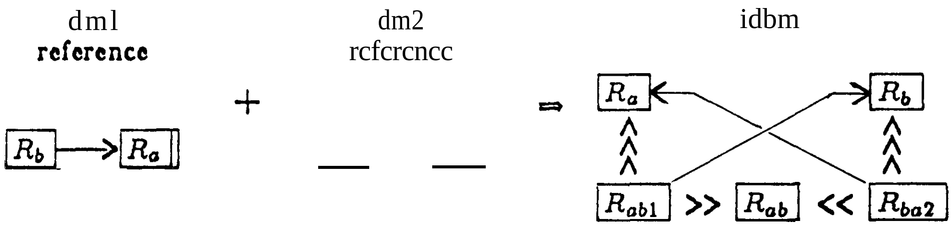
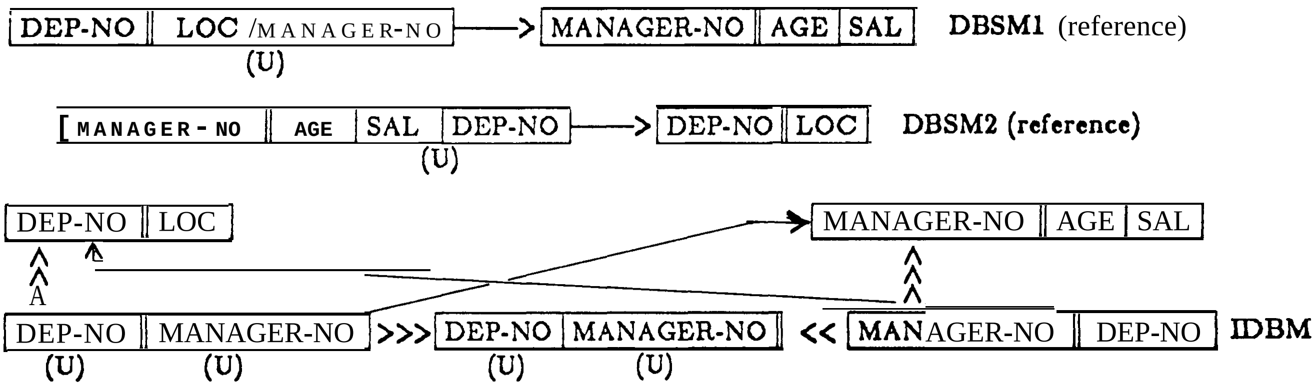


Figure 12a. Integration of rcfcrcncc and reference



Example 11

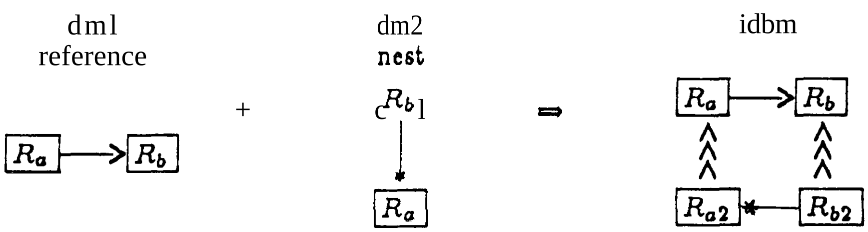


Figure 12b. Integration rcfcrcncc and nest (Case 1)

Differences:

(1) In dml (dm2), every R_a (R_b) tuplc must rcfcrcnce an R_b (R,) tuple.

(2) Deletion of R_b (R,) tuplcs is restricted in dml (dm2).

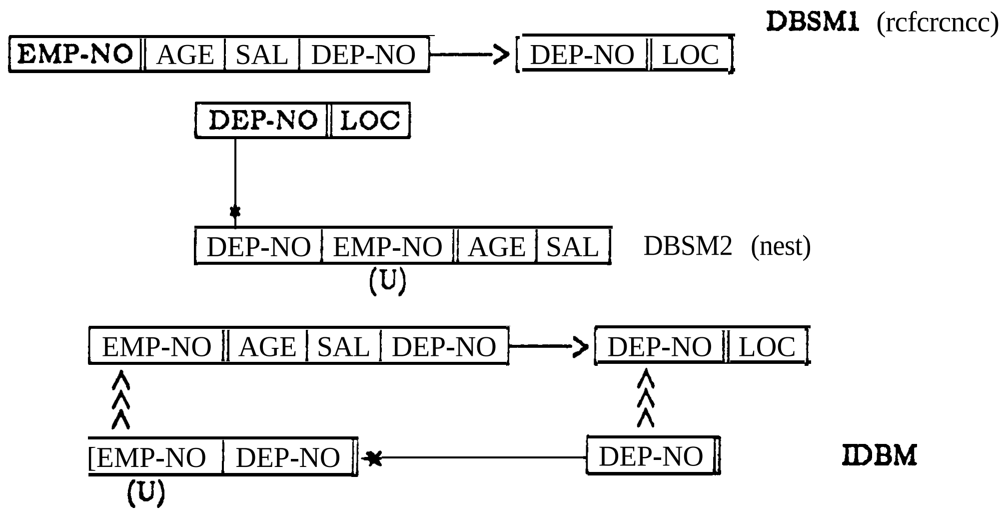
Additional constraints:
dbsml:
insert: R_a - (R_a,R_{a1},R_{ab},R_{b2}).
delete: R_a - (R_a),R_{a1},(R_{ab}).

dbsm2:
insert: R_b - (R_b,R_{b2},R_{ab},R_{b2}).
delete: R_b - (R_b),R_{b2},(R_{ab}).

(b) REFERENCE AND NEST(figure 12b,12c):

Case 1 (figure 12b):

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



Example 12

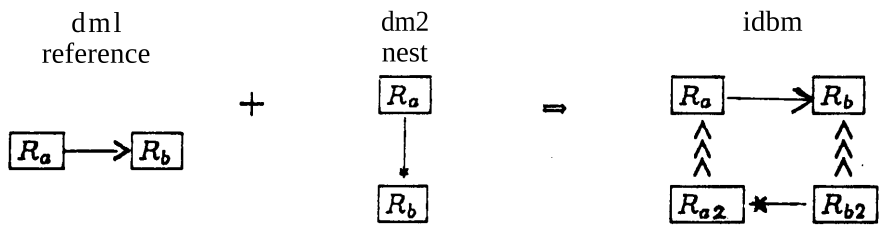


Figure 12c. Integration of reference and nest (Case 2)

Differences:

(1) In dbsml, deletion of R_b tuplcs is restricted by referencing.

(2) In **dbsm2,** deletion of an R_b tuple requires deletion of related tuplco in R_a.

Additional constraints:
d bsml:
insert: (1) R_a - $(R_a,(R_{a2}))$, R_b - R_b, R_{b2}.

dbsm2:
insert: R_a - (R_a,R_{a2}), R_b - R_b, R_{b2}.
delete: R_b - (R_b),R_{b2}.

Example 12 illustrates this case by a $1{:}N$ relationship between **DEPARTMENTS:EMPLOYEES.**

Case 2 (figure 12c):

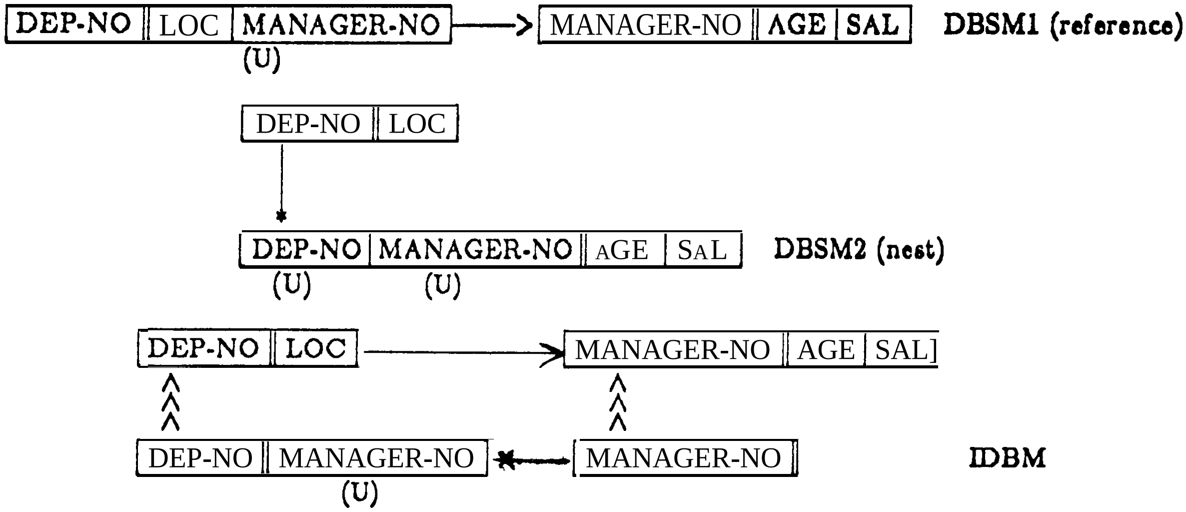
The cardinality of **the** relationship **A:B** is restricted to **1:1,** since in dml it is $N{:}1$, and in dm2 it is $1{:}N$.

Differences:

(1) Every R_a tuplc in dml must reference an R_b tuplc, while in dm2 every R_b tuplc must be owned by an R_a tuple.

(2) Deletion of R_b tuples is restricted in dml.

(3) Deletion of an R_a tuplc in dm2 requires deletion of owned R_b **tuples.**



Example 13

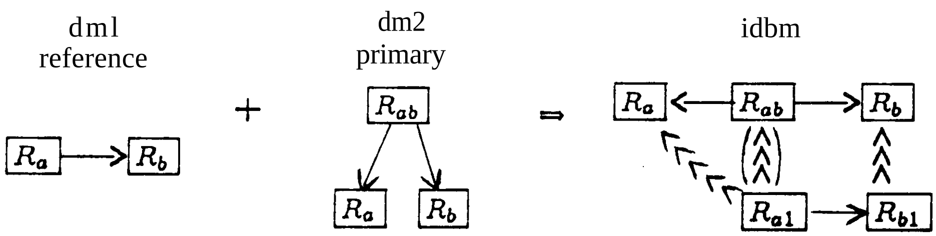


Figure 12d. Integration of reference and primary

Additional constraints:
dbsml:
insert: R_a - (R_a, R_{a1}, R_{b2})

dbsm2:
insert: R_b - (R_b, R_{a1}, R_{b2})
delete: R_b - $(R_b), R_{b2}$

Example 13 illustrates this case,

(c) REFERENCE AND PRIMARY(figure 12d):

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

Differences:

In dm2, deletion of R_a is restricted by references

Additional constraints:
dbsml:
insert: R_b - R_b, $(R_{b1}$
delete: (1) R_b - $(R_{b1}, (R_b))$
dbsm2:
insert: (1) R_b - R_b, R_{b1}, (2) R_{ab} - R_{ab}, R_{a1}

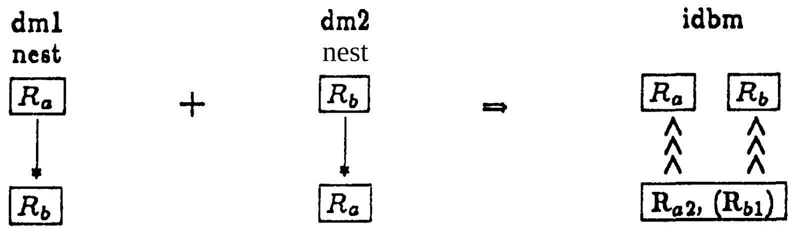
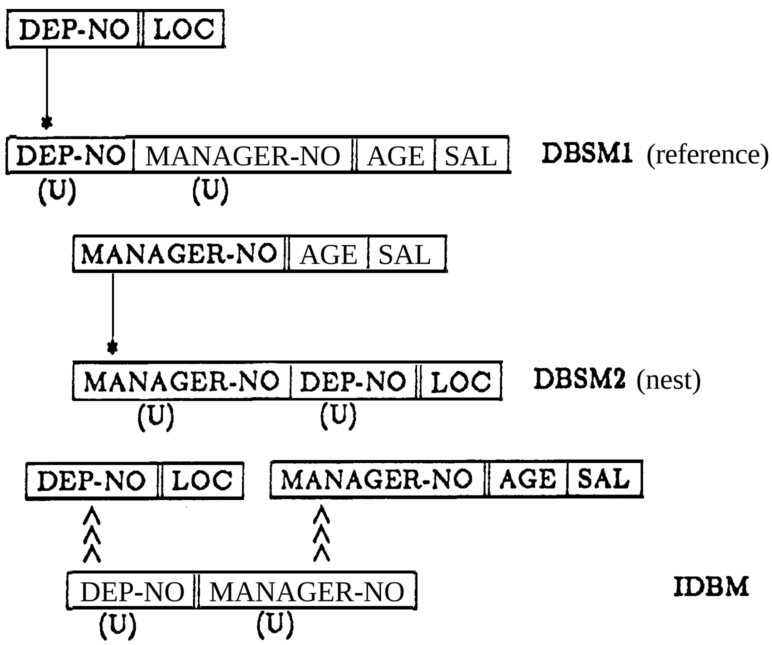


Figure 13a. Integration of nest and nest



Example 14

5.3.4. Integration with a nest:

(a) NEST AND NEST(figure 13a):

The cardinality of **A**:**B** is restricted to **1:1.**

Diff **erences:**

(1) In dml **(dm2),** every $\mathbf{R_b}$ $(\mathbf{R_a})$ tuplc must be owned by an $\mathbf{R_a}$ $(\mathbf{R_b})$ tuplc.

(2) Deletion of an $\mathbf{R_a}$ $(\mathbf{R_b})$ tuplc in dml (dm2) requires deletion of **the** owned $\mathbf{R_b}$ $(\mathbf{R_a})$ tuplc.

Additional constraints:
dbsml:
insert: $\mathbf{R_b - (R_b, R_{b1})}$

dbsm2:
insert: $\mathbf{R_a - (R_a, R_{a2})}$

Example 14 illustrates this **case.**

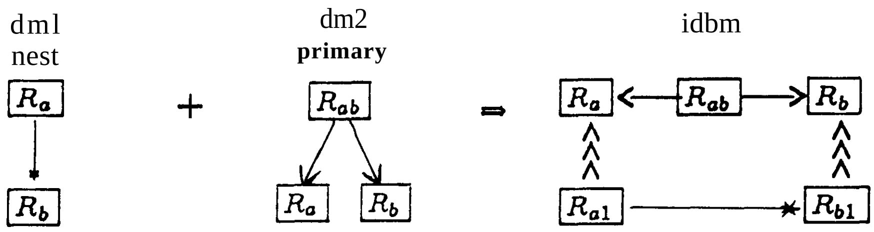


Figure 13b. Integration of nest and primary

(b) NEST AND PRIMARY(figure 13b):

The **cardinality** of the relationship **A:B** is $1{:}N$.

Differences:

(1) In dm2, deletion of $\mathbf{R_a}$ is restricted by references

(2) In dm2, dclction of an $\mathbf{R_a}$ tuplc results in deletion of owned $\mathbf{R_b}$ **tuples,** while in dm2 deletion of $\mathbf{R_b}$ is restricted by references

Additional constraints:
dbsml:
insert: (1) $\mathbf{R_a}$ - $\mathbf{R_a}$, $\mathbf{R_{a1}}$, (2) $\mathbf{R_b}$ - ($\mathbf{R_b}$, $\mathbf{R_{b1}}$)
delete: (1) $\mathbf{R_b}$ - $\mathbf{R_{b1}}$, ($\mathbf{R_b}$)
dbsm2:
insert: (1) $\mathbf{R_a}$ - $\mathbf{R_a}$, $\mathbf{R_{a1}}$, (2) $\mathbf{R_{ab}}$ - $\mathbf{R_{ab}}$, $\mathbf{R_{b1}}$

6. RELATIONSHIP TO OTHER MODELS

In this section, we examine some of the similarities between the structural model and other data models.

6.1. The relational model:

In relational model theory, the concepts of functional [Codd72] and multivalued [Fagin77] dependency among attributes arc important for normalization of relations and data model design. A functional dependency between two attributes A_1 and A_2, denoted by $A_1 \Rightarrow A_2$, means that for each value of $DOM(A_1)$, a unique corresponding value of $DOM(A_2)$ can be determined. Functional dependency between two sets of attributes is defined correspondingly. Attributes of a relation R in Boyce-Codd normal form obey the constraint: if any attribute A in R is functionally dependent on a set of attributes X in R, and A is not in the set X, then all attributes *in* R arc functionally dependent on X.

All relations in the structural model arc in Boyce-Codd normal form, and hence obey the above constraint. A functional dependency will also exist between each attribute in a referenced relation, and the ruling part of the referencing relation. Hence, a reference connection from a relation R to another relation $R'[A_1, \ldots A_i]$ defines i functional dependencies $K(R) \Rightarrow A_j; j = 1, \ldots i$. This is so because a functional dependency $K(R) \Rightarrow X_r$ will exist in relation R, where X_r is the set of referencing attributes in R. WC will also have the functional dependencies $X_r \Rightarrow K(R')$ and $K(R') \Rightarrow A_j; j = 1, \ldots i$. From the transitivity rule for functional dependencies, it follows that $K(R) \Rightarrow A_j; j = 1, \ldots i$.

Since the structural model is constructed from relations, a relational query system based on the relational algebra or the relational calculus can be used on the structural model. However, additional capabilities exist in the structural representation to simplify expression of queries by making use of represented connections. For example, consider the structural schema in figure 8a. A query such as 'FIND THE WORK LOCATION OF EMPLOYEE NUMBER 5" does not have to be expressed a6 a join between two relations, since the reference connection specifics the department tuplc that corresponds to the tuple representing employee number 5.

In the relational model, one has to specify integrity constraints to maintain tuples in different relations in a consistent state. In the structural models, such constraints may bc specified implicitly via connections.

6.2. The hierarchical model:

A hierarchical model can. be expressed using relations as record types and ownership connections as hierarchical arcs. Hence, if a structural model is restricted such that only ownership connections are used, and such that all relations arc connected together in a tree structure, a hierarchical definition tree would result. The difference in representation is the redundancy created by repetition of the ruling part attribute of the owner relations in the owned relations. However, such redundancy need not be implemented in a hierarchical implementation.

6.3. The network model:

The link set concept of the network model can be represented in the structural model. An automatic set can be defined using an ownership connection. Again, the only difference is the redundant representation of the ruling part attributes from the *owner* relation in the owned relation.

However, the existence of the connecting attributes implicitly specifies the set occuranca when a new member tuplc is inserted in the data model without requiring an additional procedure to specify the correct owner.

A manual set can be represented by a 1:N association between two relations, as in the example of figure 8d. Here, the DEPARTMENTS relation corresponds to the owner type, and the EMPLOYEES relation to the member type. Employee tuplcs can exist without belonging to any department tuple, and the set of members of each department owner tuplc is specified via the association.

The implementation oriented features of the hierarchical and network models arc implementation dependent, and hence arc best left to the implementation phase. Wc note that the structural model may represent structures that are not part of any of the three other models, as shown in section 4.2.

7. THE DATABASE DESIGN PROCESS

This section summarizes the process of designing the database with the aid of the structural model, and provides a brief discussion of considerations for model implementation. The approach, which wc only outline here, provides much of the motivation for concepts presented in the structural model. A detailed description and analysis of the remaining steps of the database design process will bc the eubjcct of a later report.

An overview of the entire design process for an integrated database system is given in figure 14. Wc define three group6 of people that partake in the design process: the potential users, the integrators, and the implementors. These group6 will interact during the database design process. The vertical axis in figure 14 defines the activities of each group relative to a time frame.

A potential user is a group of people or application program6 that expect to use the database system. Many such potential users will exist since wc arc designing a large, integrated database. Each potential user must analyse hi6 requirements, and define a data model with expected load estimates. Since a database typically serves many diverse but potentially related interests, many such data models can be established.

In section 4.1, wc showed how the structural model guide6 the design of data models. Additional information is solicited from the potential user about hi6 expected use of hi6 data model. This information is not part of the data model, but is attached to the relations and connections of hi6 data model. This includes additional integrity constraints, and expected retrieval and update characteristics *for* the data model. Load estimates will be classified into several update and retrieval components on the relations and connection6 of the data model.

The database integrators then undertake to combine these data models into an integrated database model. In the process of combining the data models, conflict6 may arise which have to be resolved by changing 6omc of the data models. There may be data models which turn out to be unrelated, or weakly related, to the core of the integrated database model so that they arc not included. The result of the data model integration is to define preliminary database submodels for the user groups. This process will need consultation with the users if their data model ha6 to be changed.

The integrators then combine the load estimates from the individual user data model6 and produce load estimates for the database model. When the transformation from data models to database model is simple, the load estimates can simply be added together. In complex situations, load data will have to be transformed to correspond to the transformation from the data models to the database model.

The implemetors then use the cummulative load estimates on relations and connections to design the file structures and accc66 methods. They take into account the expected update and retrieval load6 for the database model. Connections that arc expected to be used frequently should be explicitly represented in the implementation. A methodology for designing the file structures and access methods based on the expected update and retrieval characteristics of relations and connections will be described in a later report.

When usage patterns change, it is reasonable to change implemented file structures and accc66 methods without affecting the structural database model. Only the performance of retrieval and updates along model relations and connections whose implementation is changed will be affected. Provisions should be made in the implementation for such a restructuring.

Provisions must also be made for changing user data models, or for deletion of existing submodels and addition of new data models. This may cause a change in the database model. Structural model changes which only affect rarely used connections will be easier to accomodatc than changes which affect very critical and tightly bound connections.

	USERS	XNTEGRATORS	IMPLEMENTORS
Timc	Requirements analysis		
	Data models construction	Data models integration into database model	
	Load estimates on data models	Load integration onto database model	Alternative file selections analysis
	Database submodels de-fined on database model	Alternative performance estimates	File design decision
		Performance prediction	

Figure 14. The integrated database design process

Wc will address the issue of database design and implementation using the structural model in a separate report. Wc will give a quantitative approach to database design, and discuss possible implementation choices for the structural model constructs.

8. Conclusions:

The model wc have presented provides a bridge between the simplicity of the relational model and the explicitness of the network model. On the one hand, all structures in the model arc relations in Boyce-Codd normal form 60 that the uniformity of the relational model is maintained. Query techniques devised for relational models can be easily incorporated into the structural model. On the other hand, important structural information about the real-world situation is incorporated in the data model, and provides important knowledge both for potential users and for database system implementors.

Wc then showed how the different representations in two data models can be integrated, leading to the construction of an integrated database model which correctly supports the different data models of the users. The integrated database model then supports the user submodels.

Our point of view of the implementation process is that connections between relations have to be carefully considered. Binding of important connections will cause reasonable levels of performancc to be achieved, At the same time, unbound connections remain recognized, and may be employed when restructuring due to changing demands becomes necessary. The decision of which connections to bind is best supported by inclusion of connections which arc candidates for binding in the database model.

References:

[Abrial74] Abrial, J.R., 'Data Semantics", in J.W.Klimbie and K.L.Koffeman (cds.), "Data Base Management" (Proc. IFIP Conf. on Data Base Management), North-Holland, 1974, pp.1-60

[Chang78] Chang, S.-K. and W.-H.Cheng, "Database Skeleton and its Application to Logical Database Synthesis", IEEE Trans. on Software Engineering, Vol.SE-4, No.1, January 1978, pp.18-30

[Chen76] 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

[Codd70] Codd, E.F., "A Relational Model for Large Shared Data Banks", Comm. ACM, Vo1.13, No.6, June 1970, pp.377-387

[Codd72] Codd, E.F., "Further Normalization of the Data Base Relational Model", in R.Rustin (cd.), "Data Base Systems", Courant Comp. Sci. Symp., Volume 6, Prentice-Hall, 1972, pp.33-64

[CODASYL74] CODASYL Data Description Language, Journal of Development (June 1973), National Bureau of Standards Handbook 113, Gov. Printing Office, Wash.D.C.,Jan. 1974, 155 pp.

[DeBlasis77] DeBlasis, J.P. and T&Johnson, "Data Base Administration - Classical Pattern, Some Experiences and Trends", Proc. NCC, 1977, AFXPS Vol.46, pp.1-7

[Eswaran75] Eswaran, K.P. and D.D.Chamberlin, 'Functional Specifications of a Subsystem for Database Integrity", in D.S.Kerr (cd.), 'Very Large Data Bases", (Proc. Intl. Conf. on VLDB), ACM, 1975

[Fagin77] Fagin, Ronald, 'Multivalued Dependencies and a New Normal Form for Relational Databases", ACM Trans. on Database Systems, Vol.2, No.3, Sept. 1977, pp.262-278

[Fry76] Fry, J.P. and E.H.Siblcy, "Evolutionof Data-Base Management Systems", ACM Comp. Surveys, Vol.8, No.1, March 1976, pp.7042

[Hammer78] Hammer, M. and D.McLeod, "The Semantic Data Model: A Modclling 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

[Kent77] Kent, W., "New Criteria for the Conceptual Model", in P.C.Lockemann and E.J.Neuhold (cde.), "Systems for Large Data Bases" (Proc. 2nd Intl. Conf. on VLDB), North-Holland, 1977, pp.1-12

[Manacher75] Manachcr, S., "On the Feasibility of Implementing a Large Relational Data Base with Optimal Performance on a Mini-Computer", in D.S.Kerr (cd.), 'Very Large Data Bases" (Proc. Intl. Conf. on VLDB), ACM, 1975, pp.175-201

[Mylopoulos75] Mylopoulos, J. and N.Roussopoulos, 'Using Semantic Networks for Data Base Management", in D.S.Kerr (cd.), "Very Large Data Bases", (Proc. Intl. Conf. on VLDB), ACM, 1975, pp.

[Navathe78] Navathc, 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

[Schmid75] Schmid, H.A. and J.R.Swenson, 'On the semantics of the relational model", in W.F.King (cd.), ACM SIGMOD Intl. Conf. on Management of Data, San Jose, California, 1975, pp.211-223

[Sibley76] Sibley, E.H., "The Devclopmcnt of Database Technology", ACM Computing Surveys, Vol.8, No.1, March 1976, pp.1-7

[Smith77] 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

[Steel75] Steel, T.B.,Jr., "ANSI/X3/SPARC Study Group on Data Base Management Systems Interim Report", FDT (pub. ACM SIGMOD), Vo1.7, No.2, 1975

[Stonebraker74] Stoncbrakcr, M., "High level integrity assurance in relational data base management systems", Electronic Research Lab. report ERL-M473, University of California, Berkeley, California, May 1974

[Taylor76] Taylor, R.W. and R.L.Frank, "CODASYL Data-Base Management Systems", ACM Computing Surveys, Vol.8, No.1, March 1976, pp.670104

[Tsichritzis76] Tsiclnizis, D.C. and F.H.Lochovsky, 'Hierarchical Data-Base Management", ACM Comp. Surveys, Vol.8, No.1, March 1976, pp.105-124

[Tsichritzis77] Tsichritzis, D. and F.Lochovsky, "Views on Data", in D.Jardine (cd.), "The ANSI/SPARC DBMS Model", North-Holland, 1977, pp.51-65

[Wiederhold77] Wicdcrhold, G., 'Database Design", McGraw-Hill, 1977, Chapter 7, pp.329-367

[Wiederhold78] Wicdcrhold, G., 'Management of Semantic Information for Databases", Proc. 3rd USA-Japan Comp. Conf., AFIPS & IPSJ, San Fransisco, California, 1978, pp.192-197