VIEW REPRESENTATION IN LOGICAL DATABASE DESIGN

Shamkant B. Navathe

Graduate School of Business
New York University

Mario Schkolnick

IBM Research Laboratory
San Jose, California 95193

ABSTRACT: The process of logical database design consists of four phases: view modeling, view integration, schema optimization and schema mapping. View modeling is defined as the modeling of the usage and information structure perspectives of the real world from the point of view of different users and/or applications. The view integration phase combines these views into a single community view which is subjected to further optimization and mapping. As a result, instances of users' model may be altered and application programs transformed.

This paper proposes a scheme for view representation which will facilitate the process of view integration. This is done by enhancing the data abstraction framework proposed by Smith and Smith. It takes into account the instance-level interrelationships among data and the identification of instances via these interrelationships. The usage perspective is incorporated as rules and assertions about schema- and instance-level insertion and deletion.

The problem of view integration is briefly addressed. Valid transformations of views are indicated as a part of the integration process.

1 LOGICAL DATABASE DESIGN

1.1 Background

For any information system it is generally accepted that there are two levels of design, logical and physical. If an information system uses a database, the process of designing the database can be again divided into two stages: logical and physical database design.

i) Logical database design - It consists of integrating the requirements of a number of applications/users to arrive at a centrally controlled and maintained logical database structure. The central structure must support individual user views of the data and support their processing needs. In order to store the database in a particular database environment, the structure should be defined in terms of the facilities, features or constraints existing under that environment.

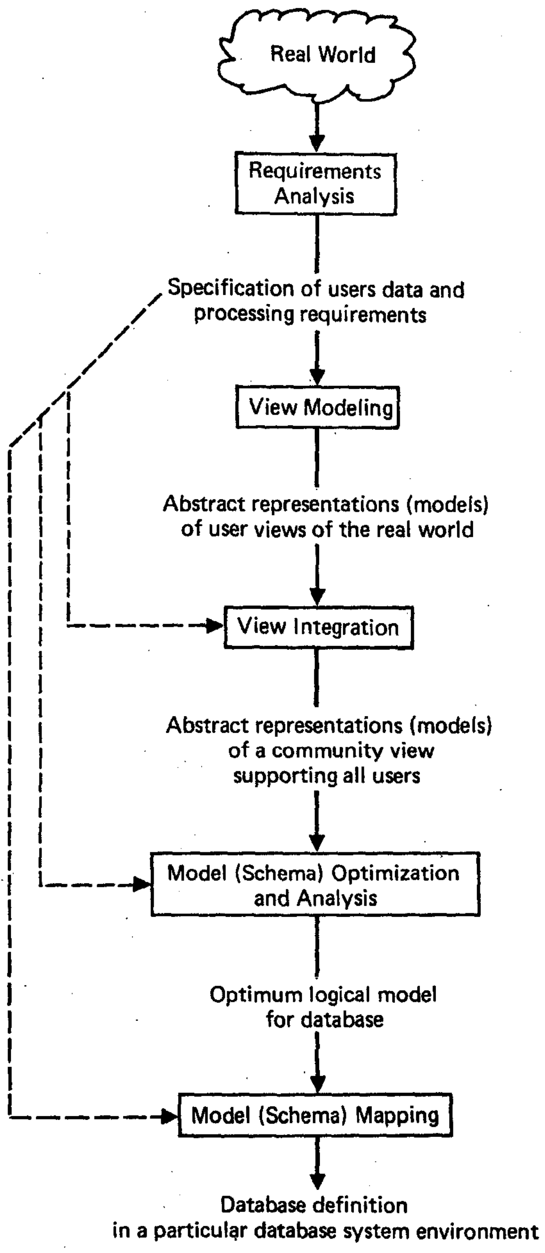
ii) Physical database design - Definition of a logical database still leaves a number of possibilities for its implementation in a certain database environment. Physical database design involves the evaluation of such alternatives and a choice of storage structure, placement strategies, and searching mechanisms, etc. This paper is concerned only with logical database design.

1.2 Current State of the Art

At the current state of the art the problem of logical database design is being tackled in practice by the database designers in an ad-hoc manner with little discipline. It leads to logical designs which do not fully meet the requirements and may either call for a redesign or require continual changes at a substantial cost. Much of the existing research in the area has been presented as independent data models (e.g., [1]), discussions of different perspectives of the problem (e.g., [10]), or individual analyses (e.g., [10,12]). An IBM program product [5] addresses the logical design problem for a particular database management system. Other syntheses leading to authomatic schema generation in particular systems based on a priori knowledge of access statistics and queries have also been proposed (e.g., [4,9]).

1.3 A Conceptual Framework

Since actual large databases in business, government and industry involve thousands of data elements and interrelationships, the solution space is not easy to deal with manually; hence it is imperative that comprehensive computer-aided approaches be developed to logical database design. It is necessary to integrate existing research in the area into a well-founded design discipline. The schematic below represents a proposed conceptual framework.[1]


Real World
Requirements Analysis
Specification of users data and processing requirements
View Modeling
Abstract representations (models) of user views of the real world
View Integration
Abstract representations (models) of a community view supporting all users
Model (Schema) Optimization and Analysis
Optimum logical model for database
Model (Schema) Mapping
Database definition in a particular database system environment

Conceptual Framework for Logical Database Design

NOTE

1. The framework has been proposed and is being investigated jointly with Bing Yao and Jay-Louise Weldon at New York University [13]. A similar framework was also formulated at an informal seminar on database design which took place at the IBM Research Laboratory, San Jose in late 1976. Participants in this seminar were Janis Bubenko, David Hsiao, Vincent Lum, Alan Merten, Mario Schkolnick, Arnie Solverg, Bob Taylor and Bing Yao.

Requirement analysis provides the input to all phases of the logical database design process. The input consists of a specification of the data requirements and the processing requirements of the potential users of a system. This activity has been investigated in great detail for information systems generally (e.g., [6]), and is also addressed by some researchers in the specific context of database systems (e.g., [3]). The main four phases of logical database design are the following:

i) View Modeling: Using requirement specifications as input, each user's view of the real world must be extracted and represented as explicitly as possible. In doing so, user's knowledge about the data and relationships at both schema and instance levels must be incorporated and the effects of processing of data upon these two levels must be considered.

ii) View Integration: The several and possibly conflicting user view must be integrated into one data model that represents a global or community view of the required data. The global model must support <u>all</u> user views. In case of conflicts, original users should be consulted to arrive at compromises. The integration therefore entails merging individual views as well as transformations.

iii) Model (schema) Optimization and Analysis: Since alternative global models can exist for the same data base, the model produced in step 2 must be analyzed and refined into an "optimal" structure.

iv) Model (Schema) Mapping: The model must finally be matched against the logical structures available in an existing database system environment and a database schema defined. Certain alternatives which are implementation-dependent are either analyzed in this phase or carried over into physical database design.

With respect to the above description of the logical database design process, the following observations are pertinent:

i) Each design step produces not a unique solution but a set of solutions associated with a vector of measures which represent various properties of the particular solution.

ii) The designer should interact with the design process to select an appropriate solution as the input to the next design step. A selection criterion must be defined and documented by the designer.

iii) Since the design requirements collected by the requirement analysis may be incomplete and inconsistent, the designer must be consulted to resolve ambiguities.

iv) The design is usually not a single-pass process. Various conditions discovered by the designer may force a re-design and iterate to an earlier design step.

The present paper addresses mainly the view modeling phase and to some extent the view integration phase of logical database design from the above framework. It proposes a technique for representing user and application views explicitly in terms of schema and instance level relationships. Particular attention is paid to the problem of identification of instances. The properties of different constructs with respect to instance insertion and deletion are discussed. Accurate and complete view modeling is extremely important to the logical design process described above. The introduction of one more data model per se should not be viewed as the final result of our research effort. Our long term goal is to develop techniques to facilitate the view integration process. The model described in this paper is just the first step in that direction.

When a number of views are subjected to view integration, the components of a view undergo merging, insertion, deletion, etc. at the schema level. These schema operations are briefly addressed in Section 3. During these phases of modeling and integration of views, additional inputs from designers/users to resolve conflicts are assumed.

2. VIEW REPRESENTATION

2.1 Background

The phase of logical database design called view modeling results in a representation of views of the real world which are pertinent to independent categories of users and/or applications. The process of view modeling involves at least the following two components (in addition to requirement specification):

i) Extracting from the user or from a person in charge of application development the relevant parts of real-world information;

ii) Abstracting this information into a form which completely represents the user view so that it can be subsequently used in the design.

In the rest of this paper the term <u>user view</u> will imply both application and user views.

The final outcome of the above two components is view representation individually by users. The present paper addresses view representation in the above context.

The problem of view representation has been addressed more as a by-product of a data model development. The most pertinent among these is the work of Chen [1] and Smith and Smith [12]. Chen's entity-relationship model is simple and easy to conceptualize, and allows a user view to be represented by means of entities, relationships and their respective attributues. Although the E-R diagrams are a good vehicle for view representation, the semantics of the E-R model fall short in the following areas:

i) defining relationships either among two other relationships or among an entity and a relationship;

ii) distinguishing among different types of relationships.

The work of Smith and Smith on data abstraction is very useful for view modeling. Their ideas on aggregation and generaliztion hierarchies are conceptually elegant and result in a homogeneous representation of a user view without any implementation constraints. As proposed, their approach is supposed to apply to the community view of data and as such it demands a lot of skill on the part of the designer using it if the scale of the problem is large. However, if suitably modified it can be used to advantage for view representation. We have done this, by redefining some of its constructs, adding new ones, and introducing semantic information along these contructs.

2.2 Objective

Kahn [7] pointed out that there are two aspects of user view which must be modeled in order to adequately represent it. These are: the information structure perspective (i.e., the non-process-oriented view), and the usage perspective (i.e., the process-oriented view). Both the E-R model of Chen and the data abstraction approach of Smith and Smith are biased toward the former.

Navathe [10] has proposed a technique for analyzing schema of existing databases. He distinguishes the associations among groups of data into two types: identifying associations and non-identifying associations. This concept is tied to the identifiability of individual <u>instances</u> of data and duplication of instances occurring in the database. Previously the data modeling approaches have tended to address relationships only at schema level, and have failed to incorporate the interrelationships among data at the instance level. Typically, a user has a lot of knowledge about instance relationships which does not surface in the conventional data models.

In the present paper we draw from the data abstraction model of Smith and Smith and propose a technique for view representation to achieve a better modeling of the usage perspective and to incorporate the relationships among data instances, especially those which are used for identification purposes. The objective is to obtain a vehicle which represents a user view as explicitly as possible in the following sense:

Information Structure Perspective

- distinction among different kinds of associations between entities;

- allowing associations in which entities or associations or a combination of the two can participate;

- incorporation of the dependence of entities on one another for the sake of identification.

Usage Perspective

- effect of insertion and deletion of entities and associations on one another at schema and instance levels;

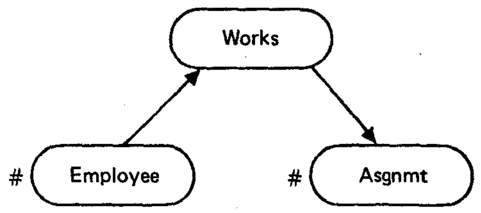
- incorporation of user-defined rules about instances of data.

2.3 Definitions and Notation

A user view is represented in terms of entities, associations, attributes and connectors in a view diagram.

An entity models anything that exists as a physical thing, e.g., an object, an event, a person, and whose existence is of interest to us. An association is an n-ary relationship among entities, among entities and associations, or among associations. It represents a fact, an idea or a specific aspect of a relationship between its components. Examples: "Advisor-of" is the name of an association between the entities Professor and Student. "Enrolled-as-teaching-assistant" is an association between two other associations, "Enrollment" and "Teaching-assignment." An association should not correspond to a physical thing which is of interest to some user; if it does, i.e., if a user is interested in the association as an object in itself, it must be represented as an entity. For example, "Has-supervisor" is allowed to be an association between Employee and Project so long as it stands for the concept of an employee being supervised on a project (by some person whose name is an attribute of "Has-supervisor" but whose existence is of no concern to us). However, if Supervisor per se is of interest to the user as a physical thing, Supervisor must be represented as an entity and "Has-supervisor" must be modeled as an association among three entities--Supervisor, Employee and Project.

An entity of an association is described by a descriptor set. Syntactically, a descriptor set can be described as a finite set of elements which are either elementary (or atomic) descriptors, called attributes or other descriptor sets. We intend that this definition not be a recursive one, i.e., a descriptor set will not contain itelf as an element. In fact, it appears that the deepest level of nesting that is needed to model real applications is two. Also, our use of the term set is an abbreviation for multiset [8], since duplicate attributes may exist. An example is given in 2.4.2.2. A descriptor which is an element of another is sometimes called a repeating group. Semantically, a descriptor set is a list of attribute values (or groups of attribute values in the case of repeating groups) that can be associated with an instance of an object (entity or association) in the database. Figure 1 shows two entities and one association with their respective descriptor sets. The entity Employee has two repeating groups, one being {Year, Grade}, the other {School, {Degree}}.



Employee = {Emp#, Dep#, {Year, Grade}, {Scool, {Degree}}}

Asgnmt = {Asgnmt#, Assignment-name}

Works = {Emp#, Asgnmt#, Supervisor-name, Start-date}

Figure 1

With the above definitions, a view is represented by a graph having entities and associations as the objects at its nodes. The edges linking the nodes are either directed or undirected and are termed connectors. A connector is a two-tuple which is ordered only if the connector is directed. In this case, an arrow is placed on the first member. For example, Figure 1 has two connectors: ($\overrightarrow{\text{Employee}}$, Works) and ($\overrightarrow{\text{Works}}$, Asgnmt). The meaning of the directed connectors will be explained later.

An association can also be described by an n-tuple containing the names of participating entities and associations. For example, Works in Figure 1 can be described as Works (Employee, Asgnmt). An instance of an entity or association is described as an n-tuple of component instances.

2.4 Rules for View Representation

The following subsections present a scheme for representing user views of data. A set of definitions and conventions are proposed. For each component of the view, the usage perspective is considered in terms of insertion and deletion at the instance level.

2.4.1 Entities

One or more elements of the descriptor set of an entity may be used to identify an instance of the entity. They constitute an internal identifier for the entity. There may be multiple candidates for the internal identifier of an entity. A user generally selects one of them to actually identify an instance of the entity. As per the discussion in Navathe [10], internal identifiers are of two types: total and partial.

A total internal identifier provides full identification so that for a given value(s) of the identifier, there is a unique instance of the entity. Contrary to this, a partial internal identifier implies that further external identification is necessary before a unique instance of the entity can be determined. Following Navathe's [10] notation, internal identifiers will be underlined in the descriptor notation of entities, e.g., Emp# for Employee and Asgnmt# for Asgnmt are underlined in Figure 1. If an entity has a total internal identifier present in it, it is called self-identified, and indicated be a # sign. Since there is a 1-1 correspondence between the values of a total identifier for a given entity and the set of instances of that entity, in the sequel we will not make a distinction between these two concepts.

Discussion on entities with respect to insertion and deletion is deferred to later subsections.

2.4.2 Associations

Associations are divided into two types, identifier associations and simple associations.

2.4.2.1 Identifier Associations

An identifier association arises when an entity is provided external identification from other entities and/or identifier associations. External identification has been defined by Navathe [10] as the process of obtaining a total identifier for an entity by augmenting its partial internal identifier with total identifiers of other entities and/or associations.

The need for external identification arises on two grounds:

i) A genuine need to distinguish between instances having the same value for an internal identifier which stand for different real-world objects. For example, two instances of Cities with the same City-name may stand for different cities and need external identification from State.

ii) A need to distinguish between duplicate instances of the same object used while realizing m:n relationships. For example, two instances of the Student which occur in relation to two different Course instances.

Let $X_1, X_2, \ldots, X_{n-1}$ be objects (entities or associations), X_n be an entity and $A(X_1, X_2, \ldots, X_n)$ an identifier association. Then there is an identifying function I_A which maps instances of $X_1, X_2, \ldots, X_{n-1}$ and a partial identifier for X_n into an instance of X_n having that partial identifier. Formally, let $\mathscr{I}_i$ denote a partial internal identifier for X_i, $\mathscr{P}(X_i)$ the set of all instances of X_i and $\mathscr{P}(\mathscr{I}_i)$ the set of all instances of the internal identifier $\mathscr{I}_i$.

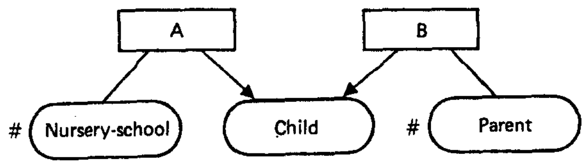
Then

$$I_A : \mathscr{P}(X_1) \times \mathscr{P}(X_2) \times \ldots \times \mathscr{P}(X_{n-1}) \times \mathscr{P}(\mathscr{I}_n) \to \mathscr{P}(X_n)$$

such that if $I_A(x_1, x_2, \ldots, x_{n-1}, y) = x_n$ for some value y of the internal identifier $\mathscr{I}_n$ then $\mathscr{I}_n(x_n) = y$. In the preceeding equation, $x_i \varepsilon \mathscr{P}(X_i)$, $i=1, \ldots, n$ and $\mathscr{I}_n(x_n) \varepsilon \mathscr{P}(\mathscr{I}_n)$ is the value of the internal identifier $\mathscr{I}_n$ for the instance x_n of entity X_n.

If $\tau_1, \tau_2, \ldots, \tau_{n-1}$ are total identifiers for $X_1, X_2, \ldots, X_{n-1}$, then $\tau_1, \tau_2, \ldots, \tau_{n-1}, \mathscr{I}_n$ is termed the augmented total identifier for X_n.

An example of an identifier association is shown in Figure 2. School-name is a total identifier for the entity Nursery-school while Childname is a partial identifier for the attribute Child. This means that there may be two or more instances of the entity child in the database with the same value of Childname. However, the existance of the identifier association A defines a function I_A such that given a value sn of School-name and a value cn of Childname, an instance of a Child is uniquely identified in the database by $I_A(sn,cn)$. Thus, {School-name, Childname} is an augmented total identifier for the entity Child.



Nursery-school = {School-name, Address}

Child = {Childname}

Parent = {Soc-sec-no, Last-name, First-name}

Figure 2

Every entity in a diagram must have either a total internal identifier or an augmented total identifier.

Insertion and Deletion of Entity and Identifier Association Instances

The existence of identifier associations permits establishing explicit rules for insertion and deletion of instance of entities and of identifier associations. This fact increases the semantics associated with our model. It also helps in the validation of changes in views one may wish to do during a view merging operation.

The rules for insertion and deletion of instances of entities and identifier association follow. These are natural rules so no further illustration is given.

An instance of an entity with a partial identifier can be inserted, provided

i) the instance is supplied external identification; and

ii) instances of entities and associations required in i) are supplied.

An instance of an entity with a total identifier can be freely inserted.

Insertion and deletion of an instance of an identifier association implies a corresponding insertion and deletion of instances of the identified entity. Insertion of an identifier association without a corresponding insertion of the identified entity is not possible except in the case of multiply-identified entities. (See 2.4.3.)

2.4.2.2 Simple Associations

Identifying associations are used to provide identification to instances of entities which are not self-identified. Sometimes we wish to define associations which do not have an identification purpose but rather to relate instances of objects. In general, if we have an association A(X,Y) between two objects X and Y and three instances $x \varepsilon X$, $y \varepsilon Y$, $(x,y) \varepsilon A$, i.e., x and y are related via A, the deletion of one of x or y clearly implies that the instance (x,y) of A has to be deleted from the database. In the previous section we discussed other deletion rules that might apply if A were an identifier association. In this section we discuss non-identifier associations and their insertion/deletion properties. An association B, defined for n objects Y_i, i=1,2,...,n, where each object is an entity or an association, is termed a simple association if B is not an identifier association. It is denoted as $B(Y_1,Y_2,\ldots,Y_n)$. A unary simple association has only one component.

Some associations have an owner/member characteristic. Member instances cannot exist without owners. We introduce this dependency in our model as follows: One of the components of B, say Y_j, j=1,2,...,m, may be defined as an <u>owner</u> of the association B, and the rest as <u>members</u> of B, provided the semantics of B implies the following. Whenever an instance y_{jk} of Y_j is deleted, all instances of B in which y_{jk} appears are deleted; furthermore, all instances of Y_ℓ, ℓ=1,2,...,j-1,j+1,...,m associated with y_{jk} via B with a directed connector $(\vec{B},Y_\ell)$ are also deleted. When Y_j is an owner of B, the corresponding connector is $(\vec{Y_j},B)$. Example: In Figure 3, association SUPPLIES (Supplier, Part, Project) is defined with Supplier as owner and Part as a member (nothing is specified for Project). It implies that whenever an instance of Supplier is deleted, corresponding Part instances are deleted owing to the directed nature of the connector (Supplies, Part); however, Project instances are not deleted since the connector (Supplies, Project) is undirected.

A number of simple associations may exist with the same components, e.g., Supplier and Can-Supply in Figure 3.

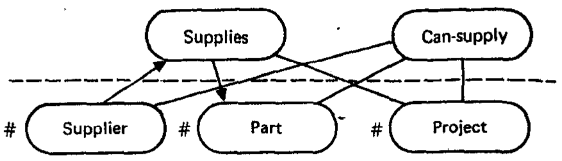


Figure 3

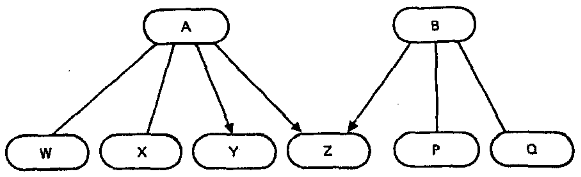


Figure 4

The owner/member description in an association can be extended. For example, for simple associations without an owner, connectors directed outward from the association can be given the following meaning. Consider the view diagram with associations A and B in Figure 4. Whenever an <u>instance</u> of W <u>or</u> of X is deleted, all instances of A in which the instance of W (or X) exist must be deleted. This is just the nornal deletion property described earlier. However, if in doing so there are no more instances of some Y (or Z) in A then those instances of Y (or Z) must be deleted provided they have no other directed connector incident from any other association. For those objects (entities or associations) having multiple incident directed connectors, the instance deletion occurs only if it is mandated by all connectors incident on it. For example, Z has two incident connectors; therefore the rule for <u>instance</u> deletion of Z is:

Delete an appropriate instance z of Z whenever

(w OR x) AND (p OR q)

are deleted, where w, x, p, and q are instances of corresponding objects such that

$(w,x,y,z) \varepsilon \mathcal{S}_A$, the set of instances of A, and $(z,p,q) \varepsilon \mathcal{S}_B$, the set of instances of B.

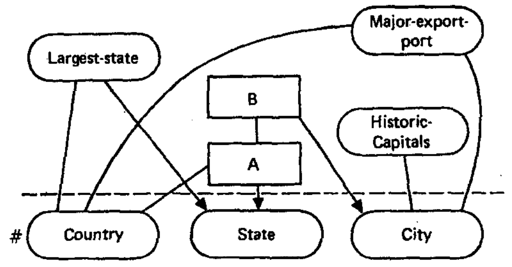
The above illustration indicates just one of the many possible ways in which a rule of instance-level insertion or deletion can be defined in the present model of view representation. Rules for which no graphic counterpart exists must be defined by means of separate assertions. (See 2.5.)

An <u>identifier of a simple association</u> is defined as the concatenation of the total identifiers of its components. In the descriptor notation a simple association $B(Y_1,Y_2,\ldots,Y_m)$ is described as

$$B = \{tid(Y_1),tid(Y_2),\ldots tid(Y_m), b_1,b_2,\ldots,b_n\}$$

where $tid(Y_j)$ is the total identifier of Y_j, and

b_i, i=1,2,...,n are attributes of B. The identifier of B is tid(Y_1), tid(Y_2),...,tid(Y_m). Duplicate descriptors are eliminated when they stand for external identifiers of the same instance. An example of the above concepts is given in Figure 5. It shows a database schema to keep information about countries, states and cities. Country is the only entity that has internal identification. State derives its identification from Country, i.e., its total augmented identifier is {Country-name, State-name.} Likewise, City derives its identification via the identifier association B. Its total augmented identifier should be {Country-name, Country-name, State-name, City-name}, the second and third attributes being the total identifier of State. However, both occurences of Country-name stand for the same instance of a country so it is not needed in the total augmented identifier. Thus, City has {Country-name, State-name, City-name} as its total identifier. A similar situation occurs for the identifier of the association Largest-state, which associates countries with the largest state in that country. The duplicate occurrence of Country-name is dropped from the descriptor set of Largest-state since it denotes the same instance of country. A different situation occurs with the association Major-export-port which associates every exporting country with that city (in the world) on which the port receiving the largest volume of traffic is located. Thus the association Major-export-port has two country names, one of the country which exports and the other of the importing country where the port is located.



Country = {Country-name, C-pop}

State = {State-name, S-pop}

City = {City-name, T-pop}

A = {Country-name, State-name}

B = {Country-name, State-name, City-name}

Largest-state = {Country-name, State-name}

Historic-capitals = {Country-name, State-name, City-name, From-year, To-year}

Major-export-port = {Country-name, Country-name, State-name, City-name, No-of-docks, Sea-name}

Figure 5

2.4.2.3 Placement of Entities and Associations in a View Diagram

As a general rule entities will be placed at a lower level with respect to all associations in a view diagram. To avoid confusion a dotted line is used to separate entities from associations whenever necessary (e.g., Figure 5).

2.4.3 Multiple Identification of Entities

In our model for view representation, every entity has actual and potential internal identifiers. The following cases can arise:

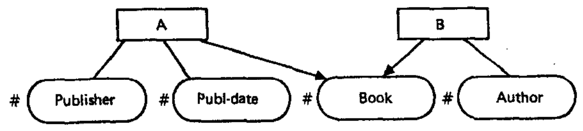
i) An entity has several potential internal identifiers, all of which are total. In this case the entity is not dependent on any identifier association and any one of its total identifiers suffices to identify its instances.

ii) An entity has several potential internal identifiers, all of which are partial. In this case external identification may be provided to the entity via one or more distinct identifier associations. As a result there will be different augmented total identifiers for the same entity. Example: In Figure 2 Child has an internal identifier Child-name. It is identified via two identifier relations A and B. The resultant augmented total identifiers are (School-name, Child-name) and (Soc-sec-no, Child-name). In such a case any one of the internal identifiers which gets augmented may be underlined in the descriptor notation for the entity.

iii) An entity has several potential internal identifiers, some of which are partial and some of which are total. If one of the total internal identifiers is actually used, then the entity can be declared as self-identified. Nevertheless a number of identifier associations can exist to provide external identification for it as in ii) above. Example: in Figure 6 the entity Book has a total internal identifier viz. Lib-congr-no; hence it is shown as a self-identified entity. Title is a partial internal identifier for it. Identifier associations A and B result in the augmented total identifiers: (Publ-name, City, Year, Title) and (Author-name, Title) for Book.

2.4.4 Special Types of Simple Associations

Three special types of simple associations called categorization, selection and subsetting are defined below. They are necessary to model explicitly a specific nature of the relationship among data instances. By distinguishing these types from the remaining simple associations it



Publisher = {Publ-name, City}

Publ-date = {Year}

Book = {Lib-congr-no, Title, Price}

Author = {Author-name}

Figure 6

is possible to define rules of schema and instance level insertion and deletion which are semantically appropriate for each individual type but which are not generally applicable to all simple associations.

2.4.4.1 Categorization

A categorization is an association between an owner entity and member entities called categories. Consider a categorization A of an entity X into entity types $X_1, X_2, \ldots, X_n$. It is denoted by the association $A(X, X_1, X_2, \ldots, X_n)$. It implies there exists a mapping f_A which maps every instance of X into a subset of categories $X_1, X_2, \ldots, X_n$.

Given $x \varepsilon X$, $f_A(x)$ identifies all those categories $X_{i1}, X_{i2}, \ldots, X_{im}$ such that $(x, x_{i1}, x_{i2}, \ldots, x_{im}) \varepsilon A$ with $x_{ij} \varepsilon X_{ij}$, $j=1,\ldots,m$. A number of categorization functions $f_A, f_B \ldots$ may be defined for a given X.

In the view diagram a categorization association is marked as "cat." The entity being categorized is connected to the categorization association by a directed connector (Figure 7).

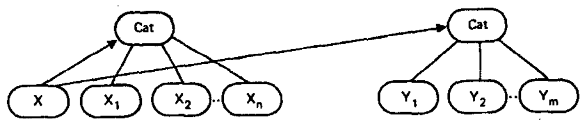


Figure 7

If the same target categories apply to two different entities, e.g., Students and Staff are both to be distinguished into Male and Female, two separate categorizations must be defined (Figure 8). Smith and Smith's [12] concept of generalization clusters is very similar to categorization; however, they allow for only mutually exclusive categories, which is a severe constraint in dealing with real-life data. For example, students in a university may be categorized into undergraduate, graduate, special, professional, resident, computer, aid-recipient, etc., so that one student could belong to multiple categories. It is unnatural to have to define a number of categorizations to satisfy mutual exclusiveness. Note that the associated mapping gives, for every student, the proper subset of categories.

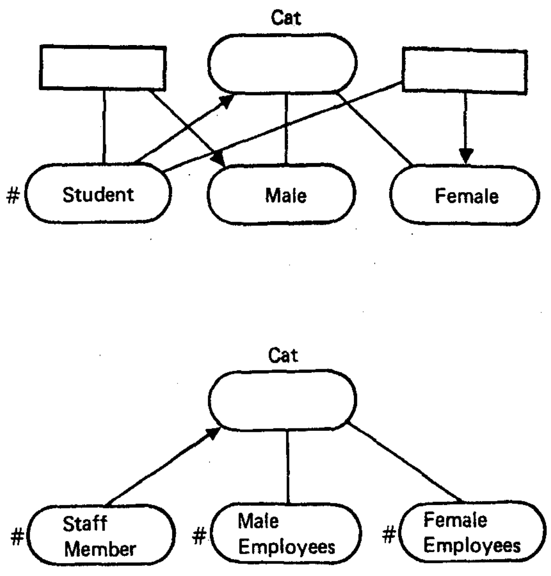


Figure 8

Whereas the owner of a categorization is externally identified or self-identified, each of the member entities (categories) may either depend for external identification on the owner or be self-identified. Transmission of external identifiers from the owner to the categories is modeled by a separate identifier association; e.g., note the (Student, Male) and (Student, Female) identifier associations in Figure 8.

Categorizations enforce an additional semantic notion related to insertion/deletion activities. Let C be a categorization of entity X with mapping f_C. Insertion of an instance x of X implies the insertion of instances of all categories included in $f_C(x)$. Conversely, insertion of an instance x_j of X_j implies a corresponding insertion of an X instance. For example, if Students are categorized into Medical, Engineering, Law, Science, Arts and Business Students, inserting a student with a joint-major in Law and Business implies inserting two new instances under those categories; alternatively, insertion in Arts implies an insertion in Student.

2.4.4.2 Selection

A selection association is a binary association among entities. Consider a selection association A(X,Y). It provides an association by which an instance of X is associated with a subset of the instances of Y. The subsets represent a partitioning of instances of Y which are non-overlapping such that a different instance of X is associated via A with each partition.

Let $\mathscr{I}_Y$ be the set of instances of Y. $2^{\mathscr{I}_Y}$ denotes the power set of $\mathscr{I}_Y$, i.e., the set of all its subsets.

Then there is a function G_A associated with A:

$$G_A : \mathscr{I}_X \rightarrow 2^{\mathscr{I}_Y}, \text{ such that}$$

$$\mathscr{I}_Y = \bigcup_{i=1}^{m} G_A(x_i)$$

and $G_A(x_i) \cap G_A(x_j) = \emptyset$ for $i \neq j$, where $x_1, x_2, \ldots, x_m$ are all the instances of X.

In the view diagram a selection association is marked as "Sel." X is represented as the owner of A(X,Y) by means of the directed connector (X,A) (e.g., see Figure 9).

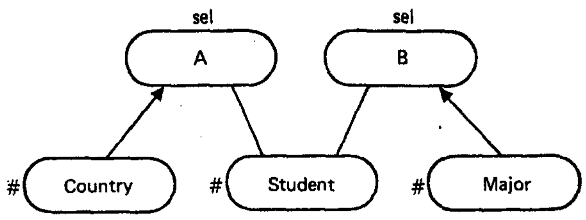


Figure 9

If the instances of a given entity are to be partitioned in various ways, a number of selections may be defined for it; e.g., in Figure 9, A and B select Student instances by Country and by Major.

The main difference between selection and categorization is at the instance level. For example, association A in Figure 9 associates countries (e.g., U.S., Canada, India and Taiwan) with the instances of students who are from the respective countries. If the same information were to be modeled using categorization, for each student instance an <u>additional</u> <u>instance</u> would be required under each of the categories such as U.S.-students, Canadian-students, etc. The mutual exclusivity constraint is easier to enforce on selection association than on categorization.

As was the case for categorizations, selections also enforce an additional semantic notion. Insertion of an owner instance x for selection A(X,Y) implies that the corresponding selection function G_A must be modified, instances of Y must be repartitioned, and new instances of A must be created if necessary. For example, if an instance of Country is inserted in Figure 9 for Nepal, the set of Student instances belonging to the Indian subcontinent previously may now be further partitioned.

2.4.4.3 Subsetting

A subsetting association is a unary association in which the component object is another association. Consider a subsetting association A(B). It provides a means of associating the name A with a subset of instances of B. Attributes of the subset as a whole are represented as attributes of A. For each subset of B a different subsetting association is required. There is no mutual exclusiveness contraint among the subsets so defined.

In the view diagram a subsetting association is marked "sub." B is marked as the owner of A(B) by means of the directed connector (B,A). In Figure 10 Advisor-of is an association between Professors and Students. Phd-advisors, Project-advisors, Curriculum-advisors are subsetting associations defined with Advisors-of as a component. The set of instances of Advisor-of which belong to these three subsets need not be mutually exclusive.

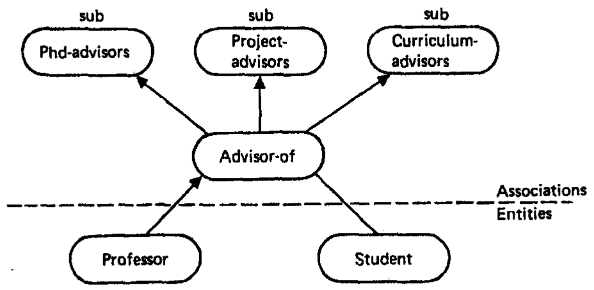


Figure 10

Selection and subsetting are parallel concepts which have been defined for entities and associations respectively. The former allows a subset of entity instances to be aggregated and associated with a foreign instance of an entity, whereas the latter allows a subset of association instances to be aggregated and named.

Insertion or deletion of an instance b of B implies a modification of the instance of each subsetting association for which B is an owner of the association and b is a member of the subset.

2.5 Assertions

When a user view is modeled according to the rules of Section 2.4, a part of the usage perspective associated with the view is inherent in terms of the rules of instance insertion and deletion which have been described for each object type. Some of the rules contribute to the meaning of a particular object type whereas some stand for a convention in view representation (e.g., see 2.4.2.2). These conventions can be modified and additional conventions may be defined to make the instance level interaction among data as explicit as possible. However, the farther the exercise is carried out the more cumbersome view representation becomes.

To state complicated interdependence of instances in a user view, it can be supplemented by a set of assertions; e.g., an assertion for the view in Figure 11 is the following:

Let $\mathscr{S}_J$ denote the set of instances of object J.

Let $\langle x,y\rangle$ denote an instance of an association (X,Y) which associates instances x and y.

Then if $a\varepsilon\mathscr{S}_A$, $s\varepsilon\mathscr{S}_S$ and $c\varepsilon\mathscr{S}_C$ are instances of Assistantship-slot, Student, and Class respectively and E=Enrollment, T=Teaching-assignment, W=Award, then $\langle a,c\rangle\varepsilon\,\mathscr{S}_T$ and $\langle a,s\rangle\varepsilon\mathscr{S}_W \Rightarrow \langle s,c\rangle\varepsilon\,\mathscr{S}_E$, i.e., a student who is given an assistantship slot for a particular class must be enrolled in it as a student. Another way of stating the same assertion would be:

$$\langle s,c\rangle\not\varepsilon\mathscr{S}_E \text{ and } \langle a,c\rangle\varepsilon\mathscr{S}_T \Rightarrow \langle a,s\rangle\not\varepsilon\ _W .$$

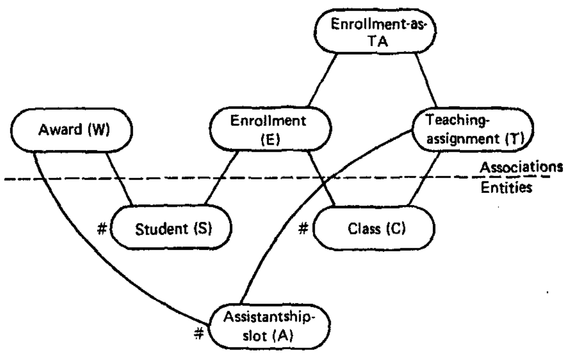


Figure 11

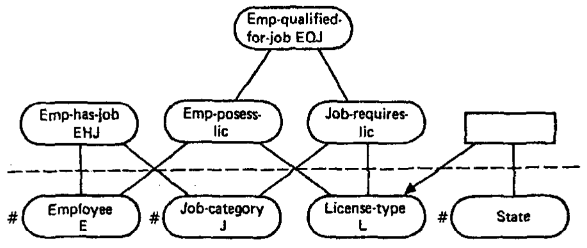


Figure 12

Consider another example view in Figure 12. It represents information on employees who have certain jobs and possess certain types of licenses. With the abbreviations for the objects as shown in the diagram, the assertion "an employee can be hired for a job provided he possesses the licenses required for the job" can be written as:

$$\langle e,j\rangle\varepsilon\mathscr{S}_{EHJ} \Rightarrow \langle\langle e,\ell\rangle,\ \langle j,\ell\rangle\rangle\varepsilon\mathscr{S}_{EQJ}$$

for at least one ℓ.

The assertions can be alternatively regarded as a statement of the validity constraints which a user makes to maintain consistency among the data expressed in his view (see [2]).

2.6 Summary

Subsections 2.3 through 2.5 have proposed a scheme for representation of user views. A view is represented in terms of entities, associations (together referred to as objects) and connectors connecting them. Entities are of two types--self-identified and externally identified. Associations are divided into identifier associations and simple associations. Three special types of simple associations termed categorization, selection and subsetting have been defined. Connectors are divided into directed and undirected connectors.

The overall model of view representation was described in terms of the definitions and notation of the various types of objects and connectors stated above and conventions pertaining to their representation in view diagrams. The rules of instance insertion and deletion for each object type express the semantics associated with their use in different contexts. Use of assertions to state additional rules was exhibited.

3 VIEW INTEGRATION

View integration is the second phase of logical database design where user views are merged to obtain a community view. In addition to merging, it also involves schema transformations necessary to arrive at "compromise views" possibly under consultation with users/designers. A detailed treatment of view integration using schema transformations proposed earlier for database restructuring [11] will be presented in a separate paper.

In performing any schema level operations on views, schema objects (viz. entities and associations) need to be inserted and deleted. The semantics of the view representation scheme presented in Section 2 will be further augmented with the schema insertion and deletion rules in Section 3.1 below. Some pointers to view merging are presented in Section 3.2.

3.1 Schema Insertion and Deletion

3.1.1 Entities

When an entity with a partial identifier is to be schema inserted in a view, one or more identifier associations must be inserted to provide total identification for that entity. An entity with a total identifier can, however, be freely inserted.

When an entity with a partial identifier which depends upon external identification is schema deleted from a view, identifier association(s) for which it was a receiving component must be deleted. For example, in Figure 5 if City is deleted, B must be deleted; if State is deleted, A must be deleted. For every identifier association so deleted, the identifier associations for which it is a sending component and those receiving entities which are not at the receiving end of any other identifier associations and which are not self-identified must be deleted. For example, in Figure 5 if A is deleted, B is deleted and City is deleted. Note, however, that in Figure 13 City has been alternatively identified from Mayor. Hence if State is deleted in Figure 13, it results in deletion of A' and B' but not of City.

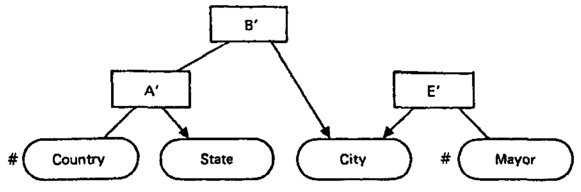


Figure 13

3.1.2 Identifier Associations

Schema insertion of an identifier association amounts to defining a new identifying function.

Schema deletion of an identifier association A has the following implications:

i) Upward propagation rule: associations in which A is a component must be schema deleted (e.g., B in Figure 5);

ii) AND rule: if <u>all</u> the identifier associations providing external identification to an entity are schema deleted <u>and</u> the entity has no total internal identifier then such an entity must be schema deleted. (For example, schema deletion of A results in schema deletion of State in Figure 5.)

3.1.3 Simple Associations

Schema insertion of simple associations is limited only by the fact that insertion of equivalent associations is redundant.

Schema deletion of a simple association also satisfies the upward propagation rule, namely, that all associations in which it is a component are schema deleted.

3.1.4 Categorization

By the definition of categorization, schema insertion of one or more additional owners for a given categorization is not allowed.

Schema insertion or deletion of a member entity (category) in a particular categorization C implies that the definition of associated function f_C be modified to include the new member entity. The effect of redefinition of f_C extends to instances of all categories, and may require some category instances to be inserted and/or deleted.

For a given entity X, if a categorization C associates it with categories $X_1, X_2, \ldots, X_m$, another categorization C' can be schema inserted which associates X with the same categories.

Schema deletion of a category from categorization C also results in a redefinition of f_C and a corresponding insertion and/or deletion of instances of other categories.

Schema deletion of a categorization C per se has no effect on its components. When the owner of a categorization is schema deleted, the categorization must also be schema deleted.

3.1.5 Selection

A given selection association A(X,Y) has only one owner and one member entity. Hence schema insertion of an entity either as owner or as member into an already defined selection is ruled out. However, any number of new selection associations may be schema inserted for a given pair of entities.

Schema deletion of a selection A by itself has no effect on its components. Since selections are binary, when the owner or member of a selection is schema deleted the selection must also be schema deleted.

3.1.6 Subsetting

Any number of subsetting associations can be schema inserted for a given component association. Similarly subsetting associations can be freely schema deleted without affecting their component.

3.2 View Merging

As was mentioned in the introduction, our goal in formulating the above model for view modeling was to facilitate the process of view integration. Such a study will be done in the near future. Some of the problems of view merging are illustrated in the following paragraphs. The operations discussed are all schema operations.

3.2.1 Merging of Simple Associations

In general, merging of simple associations would be feasible only for associations of the same type.

While operating on binary associations which are functional in nature, the following strategy can be employed. Suppose G(A,B) and H(B,C) are two simple associations for which there exist functions

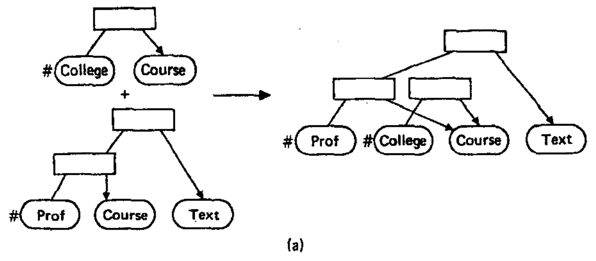
$$F_G:\mathscr{S}_A \rightarrow \mathscr{S}_B \quad \text{and}$$

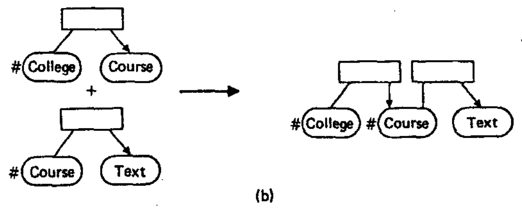
$$F_H:\mathscr{S}_B \rightarrow \mathscr{S}_C \ .$$

Then there exists a function $F_J:\mathscr{S}_A \rightarrow \mathscr{S}_C$ which is a composition of the two functions F_G and F_H, i.e., $F_J = F_H \cdot F_G$. During merging if all three of the above functions are encountered, F_J can be eliminated. This action should be approved by the user(s) since it may imply losing direct access from A to C.

3.2.2 Merging of Identifier Associations

Figure 14 shows examples of merging identifier associations in cases where the same entities play sending or receiving roles under different associations. In general the rule that the sending components in an identifier association must have total internal identifiers must be observed. Figure 15 shows possible valid and invalid structures in the context of this rule.





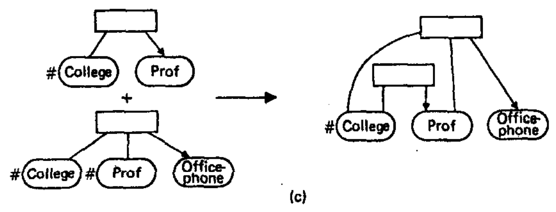


Figure 14

3.2.3 Merging of Entities

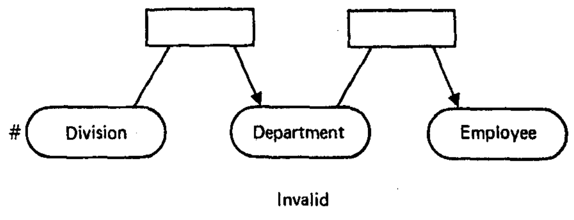
Identical entities can always be merged. Otherwise, suppose E and E' are two entities for which

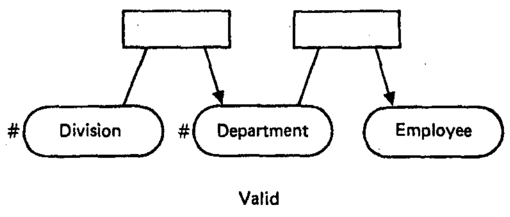
$E \subseteq E'$, (in terms of their descriptor sets)

$id(E) = id(E')$, for every internal id of E and E'

$tid(E) = tid(E')$, i.e., the total identifiers are the same

then the result of merging E and E' may be regarded as E'. It however implies that descriptors in the set E'-E receive null values in the instances of E after merging.





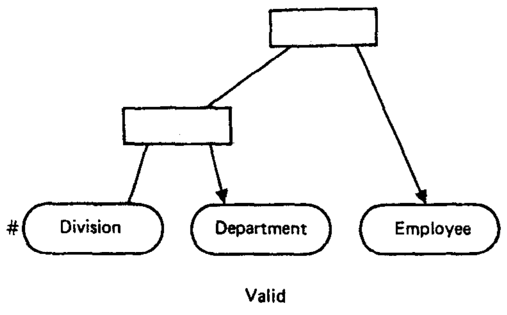


Figure 15

4 CONCLUSIONS

This paper has described a scheme for representing user/application views during the first phase of logical database design. The proposed scheme differs from conventional data models mainly in two ways. First, both schema-level and instance-level data relationships are incorporated in the view representation. Second, as a consequence, a part of the usage perspective in terms of insertion and deletion of instances is also implicit in the view representation. The resulting view representations are therefore more explicit than the representations in some of the indicated models.

The problem of view integration was only briefly addressed in this paper. It involves schema modifications in addition to the schema insertion and deletion and simple cases of merging discussed in the paper. The authors are currently working on this problem based on their earlier work on schema transformations for database restructuring.

REFERENCES

[1] Chen, P. P. S., "The Entity-Relationship Model - Toward a Unified View of Data," ACM Transactions on Database Systems, Vol. 1, No. 1 (March 1976).

[2] Eswaran, K. P., and Chamberlin, D. D., "Functional Specifications of a Subsystem for Data Base Integrity," Proc. VLDB 1975, pp. 48-68.

[3] Everest, G. C., Bray Olin, and Valters Indulis, "Developing a Data Perspective on the Specification of Information System Requirements," Report No. MIS C-TR-770-05, Management Information Systems Research Center, University of Minnesota.

[4] Gerritsen, R., "A Preliminary System for the Design of DBTG Data Structures," Comm. ACM, Vol. 18, No. 10 (October 1975).

[5] IBM, Database Design Aid General Information Manual and Designer's Guide, Publ. nos. GH20-1626-0 and GH20-1627-0, 1975.

[6] IEEE Transactions on Software Engineering, Vol. SE-3, No. 1 (January 1977).

[7] Kahn, B. K., "A Method for Describing Information Required by the Database Design Process," Proc. 1976 ACM-SIGMOD International Conf. on Management of Data, June 1976, ACM, New York.

[8] Knuth, D., "The Art of Computer Programming," Vol. 2., Addison-Wesley, 1969.

[9] Mitoma, M. F., "Optimal Data Base Schema Design," Ph.D. Dissertation, University of Michigan, 1975.

[10] Navathe, S. B., "Schema Analysis for Database Restructuring," presented at the Third VLDB Conference, Tokyo, Japan, October 1977; to appear in ACM Transactions on Database Systems.

[11] Navathe, S. B. and J. P. Fry, "Restructuring for Large Data Bases: Three Levels of Abstraction," ACM Transactions on Database Systems, Vol. 1, No. 2 (June 1976).

[12] Smith, J. M. and Smith, D. C. P., "Database Abstractions: Aggregation and Generalization," ACM Transactions on Database Systems, Vol. 2, No. 2 (June 1977).

[13] Yao, S. B., Navathe, S.B. and Weldon, J. L., "The Logical Database Design Process," Working Paper, New York University Graduate School of Business (in preparation).