Shamkant B. Navathe

Evolution of Data Modeling for Databases

The discipline of data modeling initially became established because it provided a way for specifying the structures of data* in actual file systems followed by database management systems (DBMSs). This led to the introduction of the network and the hierarchical models in the 1960s exemplified by the DBMSs called Integrated Data Store (IDS) of Honeywell (network model) and Information Management System (IMS) of IBM (hierarchical model). The relational model was then proposed as a mathematical basis for the analysis and modeling of data [16], providing data independence and addressing a variety of problems related to databases. It also provided a way for addressing redundancy as well as estimating the goodness of database structures in a formal way. The relational model made it possible to deal with integrity constraints, security schemes, distribution of data, and replication of data which could not be rigorously specified and analyzed previously.

The focus has subsequently shifted to modeling data as seen by the application and the user. Basic data abstraction concepts of classification, generalization, aggregation, and identification were applied in different combinations and different degrees to produce a plethora of "semantic" data models in the late 1970s and early 1980s. This article traces this evolution of data models and discusses the recent developments that have dominated the commercial practice of data modeling: the entity-relationship, the functional, and the object-oriented approaches. The article concludes with an overview of the current areas such as modeling of dynamic, active databases, and knowledge discovery from databases.

Data modeling benefited immensely from developments in knowledge representation and terminological reasoning, and new models such as CANDIDE [10] are springing up as a result of the marriage between these two areas. Certain developments have dominated the commercial practice of data modeling: the entity-relationship [14], and the binary approach called NIAM [37] are two examples. The functional approach was exemplified by the DAPLEX model [34] and is having an impact on object models coupled with functions such as the Iris model, now available commercially from Hewlett-Packard in their Open ODB system. A variant of the ER model called IDEF/1X (see [25]) gained a good degree of following in some government establishments. Recently, a major effort for standardizing the representation and modeling of parts data and designs under the name Product Data Exchange using STEP (PDES) [32] is under way. STEP is the ISO standard for the Exchange of Product model data. This has brought about the renaissance of data modeling which is being applied in diverse industries such as building and construction, electrical components, and architecture. Thus, the scope of data modeling now extends far beyond what it was in the early days of file-oriented systems; many organizations are embarking on corporate data modeling as a part of the strategic planning activity.

Since this issue of *Communications* is devoted to different aspects of modeling that include object-oriented analysis and modeling as well as the knowledge representation area, we will not dwell heavily on it. Our focus will be on the modeling of data as applied to the design of database structures. We will highlight the current trends in object-oriented modeling of data as well as modeling of active databases. Some of the outstanding problems for data analysis and schema design include bridging the gap between requirements specification and data modeling, integrating diverse modeling paradigms, reverse-engineering existing databases into a conceptual abstract representation, and combining the procedural and declarative aspects of data. We will not, however, be able to address these problems in this article.

*The word Data will be used in singular throughout this article in keeping with the convention in database literature.

Basic Definitions

In this section we define some basic terminology for the database area pertaining to data modeling. Interested readers may consult [8, 19] for detailed explanations of these terms.

A **data model** is a set of concepts that can be used to describe the structure of and operations on a database. By *structure of a database* we mean the data types, relationships, and constraints that define the "template" of that database. A data model should provide **operations** on the database that allow retrievals and updates including insertions, deletions, and modifications. Note that we will use the term "data model" to refer to the discipline for modeling data in a particular way—one that provides the building blocks or the **modeling constructs** with which the structure of a database can be described. We will use the term **application model** to refer to the description of data for a particular database. For example, the relational model is a data model. The definition of a particular database for the personnel application at company X will be called an application model of that database which uses the relational model. Application analysts often refer to the latter as a data model, causing confusion.

In any application, it is important to distinguish between the *description* of the database and the *database itself.* The description is called the **database schema.** A database schema is designed for a given set of applications and users by analyzing requirements. It is described by using a specific data model that provides the modeling constructs in the form of a language syntax or diagrammatic conventions. In some tools or DBMSs, the data model is not explicitly defined but is present implicitly in terms of the features present. Schemas usually remain relatively stable over the lifetime of a database for most applications. For dynamic environments, such as computer aided design (CAD), or computer-aided software engineering (CASE), the schema of the product being designed may itself change. This is referred to as **schema evolution.** Most DBMSs handle a very limited amount of schema evolution internally.

During the process of database design, the schema may undergo transformation from one model into another. For example, the schema of the personnel database may be initially described using the entity-relationship data model in the form of an ER diagram. It may then be mapped into the relational data model which uses structured query language (SQL)—an emerging standard, to define the schema. The entire activity of starting from requirements and producing the definition of the final implementable schema in a DBMS is called **schema design.** The DMBS is used to store a database conforming to that schema; it allows for database **transaction processing** in which a transaction is a unit of activity against the database that includes retrieval and update operations against the database. A transaction must be performed in its entirety, leaving a "permanent" change in the database or may be aborted, performing no change at all.

The actual database reflects the state of the real world pertaining to the application or the "miniworld." It must remain in conformity with the miniworld by reflecting the actual changes taking place. The data in the database at a particular time is called the "database instance" or the "database state." The actual **occurrences** or **instances** of data change very frequently as opposed to the schema, which remains static.

A data model in the database parlance is associated with a variety of languages: data definition language, query language, data manipulation language, to name the important ones. The **data definition language (DDL)** allows the database administrator or database designer to define the database schema. The DBMS has a compiler to process the schema definition in DDL and to convert it into a machine-processable form. This way, a centralized definition of the application model is created, against which a number of applications can be defined. **Data manipulation language (DML)** is a language used to specify the retrieval, insertion, deletion, and modification of data. DMLs may be divided broadly into two categories: declarative and procedural. The former allow the user to state the result (of a query) that he or she is interested in, whereas the latter require one to give a procedure for getting the result of the query. The nature of the language also depends on the data model. For example, while it is possible to have a declarative language for a model such as the relational model, the language for the network data model is "navigational," requiring the user to state how to navigate through the database, and thus is inherently procedural. Either type of language may be used in a stand-alone interactive fashion as a **query language.** Languages for data models can also be distinguished in terms of whether or not they are record-at-a-time or set-at-a-time. Record-at-a-time processing requires an elaborate control structure typically provided by a **host programming language** within which the DML commands or verbs are embedded. Set-oriented processing regards data as sets of elements (e.g., sets of tuples in the relational model) and provides for operators that apply to these sets, generating new sets. There is now a movement toward providing languages which seamlessly integrate the capability to provide general-purpose computation and special-purpose data manipulation against the database in a single language. These are called **database programming languages (DBPLs)** [6].

Scope of Data Models

In the traditional sense, data models used for database schema design have been limited in their scope. They have been used to model the *static* properties of data including the following:

Structure of data: The structure is expressed in terms of how atomic data types are aggregated into higher-order types. Furthermore, the models express relationships among these aggregates. The early models tended to be record-oriented, the basic aggregate structure being a record type consisting of data element types or field types. The database schema consists of a number of record types that are related in different ways. The hierarchical data model organizes the record types in a tree structure, whereas the network data model organizes a database schema with the record types as the nodes of a graph. The limitations of the record-based view of data modeling are discussed in [23]. The relational model introduced a set-oriented view of data modeling [16], and currently, the object-oriented view which structures a database in terms of objects and interobject interactions is gaining popularity [3]. We discuss these approaches in greater detail later in this article.

Constraints: Constraints are additional restrictions on the occurrences of data within a database that must hold *at all times.* Data model constraints serve two primary goals:

- Integrity: Integrity constraints are the rules that constrain the valid states of a database. They arise either as properties of data, or as user-defined rules that reflect the meaning of data.
- Security and protection: This applies to restrictions and authorization limitations that are applied to a database to protect it from misuse and unauthorized usage.

Constraints can be visualized at different levels:
a) **inherent constraints** pertain to the constraints that are built into the rules of the data model itself. For example, in the entity-relationship model a relationship must have at least two participating entity types (depending on the variant of the model used, the same entity type may be used twice).
b) **implicit constraints** are constraints that can be specified using the DDL of a data model to describe additional properties. They are expected to be automatically enforced. An example is a mandatory participation constraint in the entity-relationship model, which states that a specific entity must participate in a particular relationship.
c) **explicit constraints** are application-dependent constraints that specify the semantic constraints related to an application. These are the most general and difficult constraints to specify in full detail. There is a general trend to capture as much "application behavior" information within a database as possible in order to capture it in a central place. The 4GL movement is aimed at capturing this application semantics at a high level. Today's DBMSs are not equipped to handle such constraints easily, however.

Another dimension of constraint modeling is to capture state transition rather than just static state information. This gives rise to **dynamic constraints** which are stated in terms of what types of changes are valid on a database. An example is: "the salary of an employee can only increase." Both static and dynamic constraints allow us to define whether a database is consistent. Whereas static constraints can be evaluated on a "snapshot" of a database to determine its validity, the dynamic constraints are much more difficult to enforce, since they involve blocking/preventing a state transition at "run-time."

Other parameters of data models: A data model for a database may also specify some additional details relevant to the use of the data. One possible feature is **"distribution parameters."** These relate to the fragmentation of data in terms of how data is stored as fragments. In the relational model it is customary to refer to "horizontal" and "vertical" fragments. The former contain subsets of the data occurrences of a relation that meet some predicate condition, while the latter refer to a subset of the attributes of data for the whole relation. In a relation called ORDERS, each horizontal fragment may contain orders that are shipped from one warehouse, whereas ORDER may be vertically fragmented into shipping information and billing information. **Security** is another feature that may be built into a data model at different levels of granularity. Yet another feature is **redundancy,** which is hard to model explicitly; it may be captured in the form of specification of explicit copies or overlapping data. A model must allow for specification of features such as **keys** which uniquely identify data; it may also have a way to specify physical parameters such as **clustering** or **indexing** (e.g., B+ tree index on a certain field) as a part of the application model specification.

Views: In database modeling a **view** is a perceived application model as defined by a user or an application group. A view is another mechanism by which an application can record its specific requirements, in addition to the explicit constraints mentioned previously. Most data models provide a language to define views: it may be coincident with the query language, whereby the result of a query is defined as the view. Typically, a view is constructed when there is a request to retrieve from it, rather than "materializing" the view because the latter creates redundant data.

Toward Joint Data and Application Modeling

To give adequate support for the modeling of dynamic application environments, another school of thinking combines the functional analysis, which is typically the focus of software engineering during information system design, with conventional data modeling [11, 26], giving rise to a joint analysis and modeling of application and data requirements. We have also advocated this view during conceptual database modeling in [8]. In terms of the present issue of *Communications,* the preceding approach seems most relevant and significant. A top-down structured design methodology for both data and process modeling using extended ER and data flow diagrams respectively is proposed in [11]. It suggests building "mini-ER-process-schemas" based on the most refined process diagrams using some simple rules. These minischemas are then integrated into the overall model using view integration techniques [7] to give rise to the global data model. In [8] we propose that global schema design for a set of applications must proceed like a "rock-climbing activity," alternating by successively expanding the data model and the process model and cross-checking at each step to make sure the data model accounts for processing requirements and the process model refers to the data identified at the previous step. The cross-checking is facilitated by maintaining a data dictionary. As a methodology of joint design, top-down, bottom-up and mixed or inside-out methodologies are described. It has been shown by [26] that the ER model used normally for data modeling can itself be used with some additional constructs for such tasks as ordering, and sequencing, to adapt it to process modeling.

The area of transaction modeling as a part of data modeling is relatively new. A transaction specification language and a tool using an object model has been described by [31]. In [30] we define the language for specifying a transaction against a "neutral" data model at the conceptual level and show its mapping into SQL, assuming the transaction executes against an underlying relational database.

Data Model Classification

Data models for database management can be classified basically along two dimensions. The first dimension deals with the *steps of the overall database design activity* to which the model applies. The database design process consists of mapping requirements of data and applications successively through the following steps (see Figure 1).

a) Conceptual Design—here the model provides a way of capturing the users' perception of data. The **conceptual data model** provides the concepts essential for supporting the application environment at a very high nonsystem-specific level.

b) Logical Design — here the model represents the organization of data for some implementable data model called the **logical data model** or an **implementation model.** This model has modeling constructs that are easy for users to follow, avoid physical details of implementation, but typically result in a direct computer implementation in some DBMS.

c) Physical Design—typically, at this step, data modeling is *not* used to describe data. Physical design consists of a variety of choices for storage of data in terms of clustering, partitioning, indexing, providing additional access or directory structures, and so on. Some work has been done on developing concepts for describing physical implementations along the lines of a data model (e.g., see [9]).

In the three-level architecture of the ANSI/SPARC committee for database management [4], schemas corresponding to the three levels of a DBMS architecture have been clearly identified. They are called the Conceptual, External and Internal Schemas. The relationship of the ANSI/SPARC terminology with our steps of database design is shown in Table 1.

Note that in Table 1, we deliberately added a design step called view design to accommodate the notion of external schema from the ANSI/SPARC nomenclature. Typically, it is bundled into the logical design step.

In another dimension, data models can be classified into record-based models, semantic data models, and object-based models in terms of the "flexibility" or "expressiveness" scale. **Flexibility** refers to the ease with which a model can deal with complex application situations. **Expressiveness** similarly refers to being able to bring out the different abstractions and relationships in an involved application. Record-based models were the models adopted from files, which gave rise to hierarchical and network data models. They were hardly flexible, and expressive to a limited extent [23], but played an important role as implementation models for the entire DBMS development during the late 1960s and the 1970s. The relational model [16], an offshoot from the previous models, provided more **data independence** by "elevating" the model higher, away from the physical implementation details and also provided greater power in terms of set-at-a-time operations on the model. Next arrived the semantic data models, which were more appropriate for conceptual design. The entity-relationship model [14] exemplifies semantic data models and has been a precursor of much subsequent development. Semantic networks have a close resemblance to semantic data models, except that the semantic links in the latter are of a limited scope compared to the former, and the data modeled by a semantic data model is typically secondary storage-resident. The object-based models eliminate the distinction between entities and relationships that is common to the entity-relationship model and its extensions. This area was influenced by object-oriented programming languages such as Smalltalk [17] and has emerged as a viable data modeling approach, particularly in light of the emerging applications of database systems. Recently a group of prominent researchers even published a manifesto [3] to define what a true "object-oriented" (O-O) database management system is. The O-O data models apply to both the conceptual and logical design phases. In the remainder of this article we will highlight the concepts underlying the classes of models described earlier and their contribution to the general field of data modeling.

Implementation Data Models

In this category we place the models that had their origin in files but offered the possibility of organizing large collections of data. Bachman [5] proposed the idea of data structure diagrams to capture the structure of aggregate records and the one-to-many relationships called set-types among record types. This eventually led to the definition of the network data model which was defined by the CODASYL Database Task Group [15]. This hierarchical data model was developed as a part of the DBMS called Information Management System [IMS] by IBM. There are no documents or reports from any committees that defined the hierarchical data model. Today, because of the dominant use of IMS in the market, the hierarchical model has become synonymous with the IMS data model. The hierarchical data model allows for the database schema to be structured as a tree, with the nodes being record types and the links being "parent-child" relationships among the record types. It must be noted that the IMS data model is not strictly hierarchical, in that it allows a limited network by letting a record type have two "parent" record types. One, called the "physical parent," typically is in the same hierarchical structure of records, whereas the other, called a "logical parent," typically links a record to another from a different hierarchy. A general discussion of these models can be found in textbooks; in particular, the reader may refer to [19, chapts. 10–11] from which the schemas of a hierarchical and network database are reproduced in Figures 2 and 3.

The scope of these data models includes fixed rules for data-structuring. The systems implemented using these models therefore have a well-defined DDL to define the database and a DML to manipulate the data for retrievals as well as updates. Because of a large number of DBMSs that were implemented using these models, a very high percentage (probably exceeding 70%) of the databases currently used for day-to-day production in industry and government subscribe to the hierarchical and the network data models. The constraint-modeling features of these models are limited, especially in terms of implicit and explicit constraints described previously. The models, as well as the systems, are proving inadequate to model the emerging nontraditional applications in such areas as engineering, scientific and

Figure 1. The database schema design process

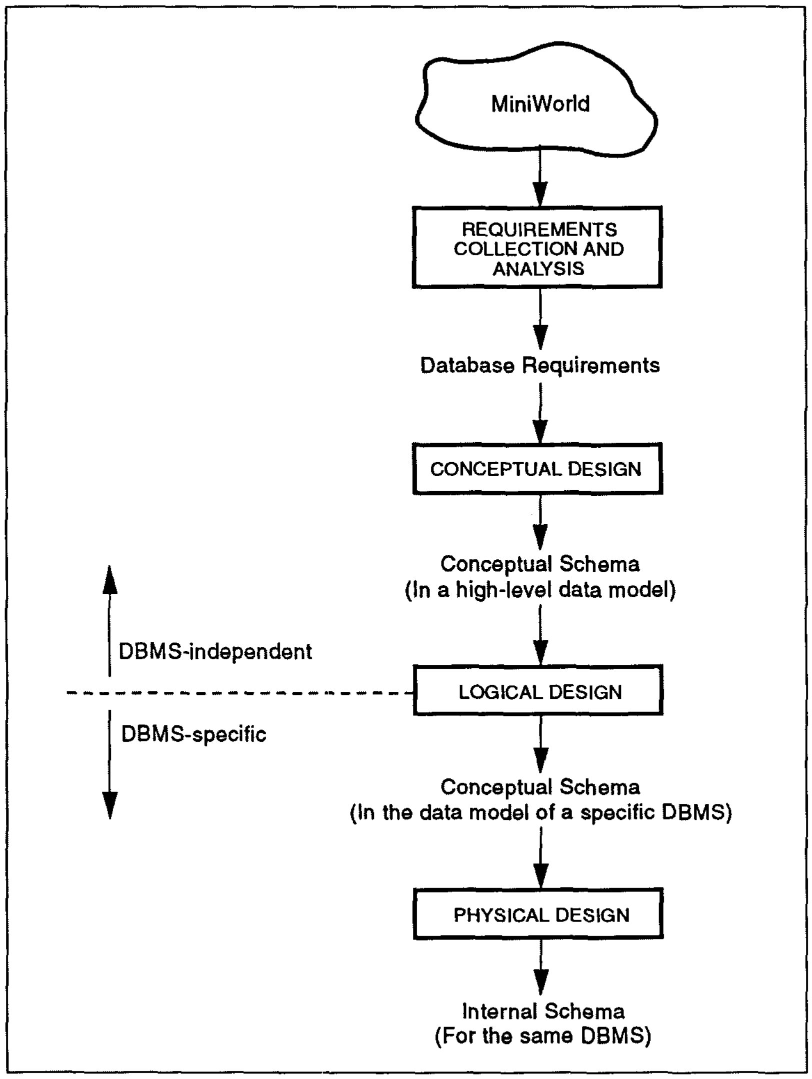
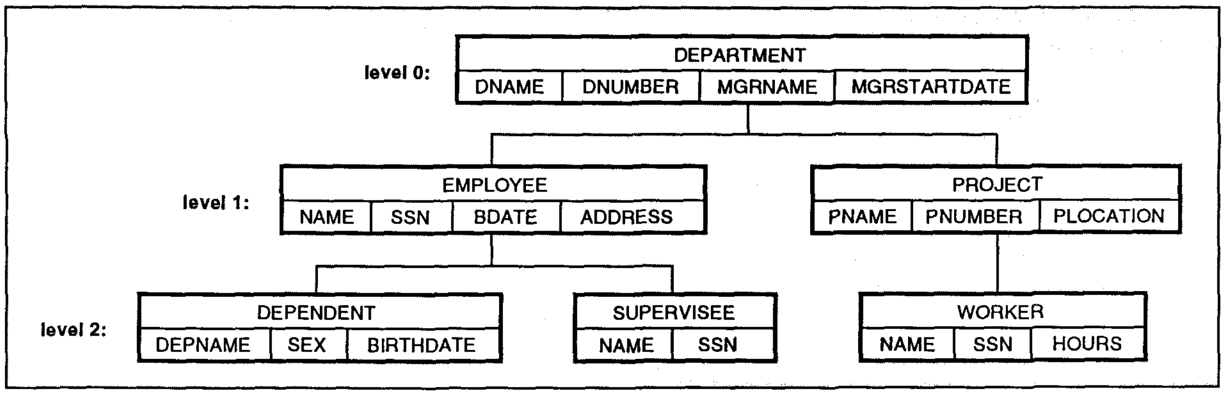

MiniWorld
REQUIREMENTS COLLECTION AND ANALYSIS
Database Requirements
CONCEPTUAL DESIGN
Conceptual Schema (In a high-level data model)
DBMS-independent
LOGICAL DESIGN
DBMS-specific
Conceptual Schema (In the data model of a specific DBMS)
PHYSICAL DESIGN
Internal Schema (For the same DBMS)

Table 1.
Relationship of ANSI/SPARC terminology to steps of database design

Step of design	End product using ANSI/SPARC terminology
Conceptual design using a conceptual model	—[†]
Logical design using an implementation model	Conceptual schema
View Design	External schema
Physical Design	Internal schema

[†]The result here is a conceptual schema in a high-level data model. It does *not* correspond to the conceptual model in the ANSI/SPARC sense.



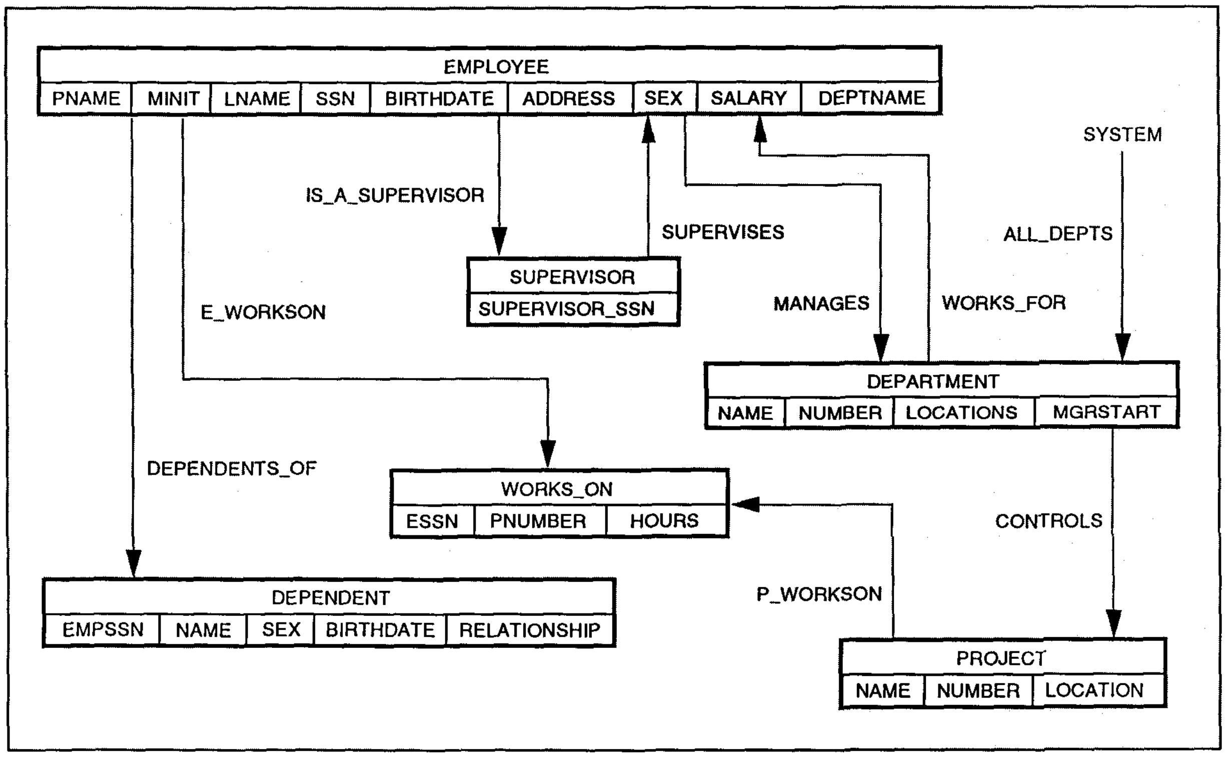


Figure 2. A hierarchical database schema

Figure 3. A network database schema

medical databases, geographic information systems, full-text and document-oriented systems, software engineering, and network management of large networks. The object-oriented data models are expected to play a major role in these applications.

The current state of the O-O DBMSs, however, is such that a large enough following has yet to be created for them. Whether they will capture a significant share of the traditional database application market is unknown at present. In the meantime, relational DBMSs are gaining increasing use because of the data independence inherent in the relational data model, ease of specifying and enforcing constraints, potential for optimization of query-processing strategies by the system, and a better foundation for distributed databases. A discussion of the relational model in some detail is therefore necessary.

The Relational Data Model

The relational model of data proposed by [16] was a landmark development because it provided a mathematical basis to the discipline of data-modeling based on the notions of sets and relations. The relational data model organizes data in the form of relations (tables). These tables consist of tuples (rows) of information defined over a set of attributes (columns). The attributes, in turn, are defined over a set of atomic domains of values. Because of its simplicity of modeling, it gained a wide popularity among business application developers. Today, a number of well-known DBMS products on the market

(DB2, RDb, INGRES, Oracle, INFORMIX, SyBase to name a few) are providing access to the relational model for a wide variety of users. The model has an associated algebra which includes operations of selection, projection, join as well as set operations of union, intersection, difference, Cartesian product and so on. The set-at-a-time nature of these operations makes them very powerful because entire tables become arguments of operators. The structured query language SQL based on relational calculus, which is a form of first-order predicate calculus is becoming a *de facto* standard for the data-processing industry. This declarative language coupled to the relational model gives plenty of scope to the system to make "intelligent decisions" regarding the processing of these set operations. In terms of constraints, there are two types of constraints that fall under the implicit constraint category for the relational data model. The first, called the "entity integrity" constraint, guarantees that no two tuples belonging to the same relation refer to the same real-world entity. In other words, it guarantees uniqueness of keys. The second constraint, called "the referential integrity constraint," makes sure that whenever a column in one table derives values from a key of another table, those values must be consistent. Unfortunately, all existing commercial relational DBMS products do not allow a proper specification and an automatic enforcement of these constraints. But the vendors are devising a variety of ways to deal with them. Figure 4 shows a set of relations connected by referential constraint arrows which equivalently represent the same data that was represented by the schemas in Figures 2 and 3.

Other features of the relational data model that have made it the popular choice for database applications today are the following:

- The basis of relational algebra and the simple and precise semantics make a good amount of query optimization feasible within the system, thus relieving the programmers' burden. This is not so in the network and hierarchical models in which the programmer manually optimizes the navigation of the model.
- The model allows a clean separation of the logical from the physical parameters making it easy for casual users to understand and deal with the database. At the physical level, typical DBMSs allow a variety of indexes and performance-enhancement features.
- The theory behind concurrent transaction-processing, as well as recovery, has been well developed for this model to give adequate performance for most commercial applications.
- For further development of distributed databases, the relational model provides a sound basis for addressing the problems related to fragmentation, redundancy, and distributed transaction processing in different modes in general.

The main argument against the relational data model is its "flatness" of structure, through which it loses the valuable information contained in the relationships or "links" among the data. It therefore clearly lacks the features for expressiveness and semantic richness for which the semantic data models are preferred.

Semantic Data Models

As shown in Figure 1, the Requirements Collection and Analysis activity produces a set of database requirements that must be captured by a data model in the next step, called Conceptual Design. This next step is best served by a high-level model called a conceptual or semantic data model. It is important to use a high-level model during this step because at this stage of analysis, the database is conceived in a very preliminary way by the potential set of users. It is not tied to a specific implementation model and certainly not to a specific DBMS. One of the shortcomings of the database design activity in organizations has been the lack of regard for the conceptual database design and a premature focus on some specific target DBMS. Designers are increasingly realizing the importance of the conceptual database design activity. The models that have been typically employed for this activity are referred to as semantic data models. A semantic data model used for purposes of conceptual design must possess the following characteristics [8]:

- Expressiveness: The model must be expressive enough to bring out the distinctions between different types of data, relationships, and constraints.
- Simplicity: The model must be simple enough for an end user to use and understand. Hence, it must always be accompanied by an easy diagrammatic notation.
- Minimality: The model must consist of a small number of basic concepts that are distinct and orthogonal in their meaning.
- Formality: The concepts of the model should be formally defined. It should be possible to state criteria for the validity of a schema in the model.
- Unique Interpretation: There should ideally be a single semantic interpretation of a given schema. In turn this implies that complete and unambiguous semantics be defined for each modeling construct.

An early candidate for a semantic data model was the entity-relationship model [14] commonly called the ER model. In terms of the preceding criteria, it is fairly simple to use, has only three basic constructs which are fairly, but not completely, orthogonal, has been formalized, and has a reasonably unique interpretation with an easy diagrammatic notation. We will not dwell on a description of the model here. Figure 5 shows the conceptual model of a database represented in the ER model for the same application shown earlier in Figures 2 and 3 for the hierarchical and the net-

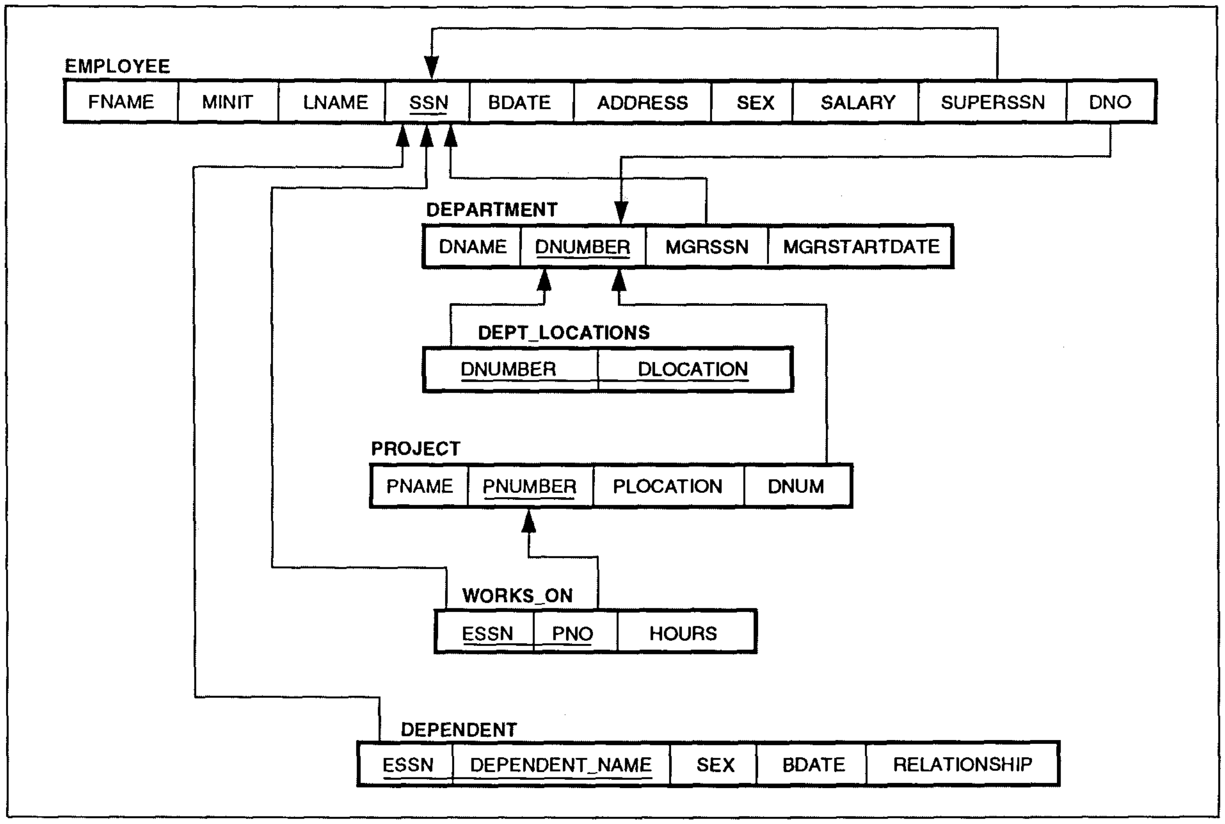


Figure 4. Relational schema with referential constraints

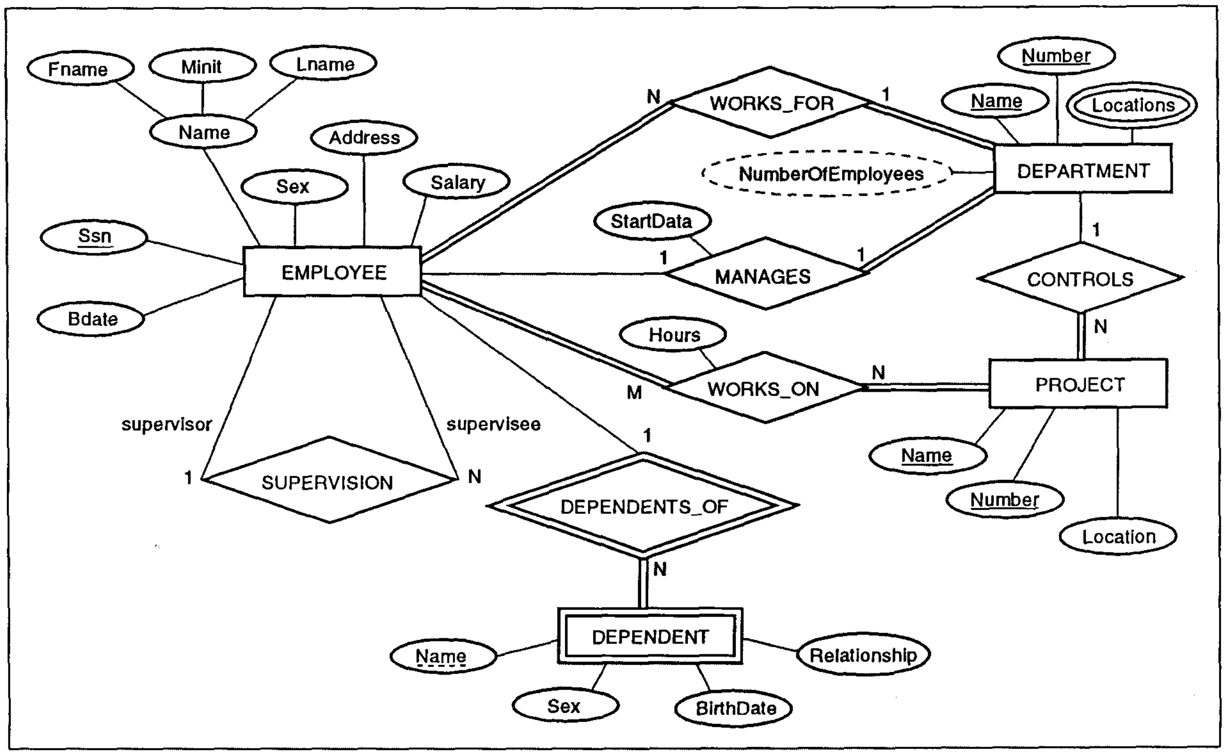


Figure 5. A conceptual database schema in the entity-relationship model

work schemas adapted from [19].

The basic constructs in the ER model are entity types, relationship types, and attribute types. In the diagram they are represented by rectangles, diamonds, and ovals respectively. The distinction between entities and relationships tends to be fuzzy; similarly, what one may call an attribute (e.g., Department is an attribute of the Employee entity type) may actually be worthy of being called an entity type (e.g., Department as an entity type with attributes Location, Manager_name, ...). Due to its lack of preciseness in modeling, the model did not become a basis for formal work areas such as distribution and transaction-processing, as did the relational data model. It has remained a favorite of the tool designers (see [7, chapt. 15]) in tools that produce and edit ER diagrams as a means for easy communication in the early stages of database design and then map the ER schemas into logical and physical schemas of target DBMSs semiautomatically.

The ER model has seen many extensions. Early work on extensions such as [33] defined invariant properties of the ER model. The model has been further enhanced with the concepts of classes and subclasses, and of inheritance hierarchies based on generalization and specialization. Recently, rules have been added to the model [36]. A detailed discussion of an enhanced ER model can be found in Chapter 15 of [19].

Fundamentally, the idea is to enrich the model to support all abstractions of data. If the distinction among entity and relationship is eliminated, it leads to a general data model with classes or object types. Along these lines, [35] proposed a semantic hierarchy model based on the notions of aggregation and generalization. A very general-purpose approach that provides a rich repertoire of facilities for grouping, relating, and constraining definitions of classes was presented by [21]. The fundamental abstractions that a semantic data model should support are given in the following list. These data abstractions apply equally to the object-oriented models to be discussed.

- Aggregation: This is the abstraction concept of building aggregate objects from component objects. The relationship between the lower-order object and the higher-order object is described as "IS_PART_OF" relationship. At another level, this abstraction is used to aggregate attributes into an object. In the ER model aggregation is used when two or more entity types are associated to define a relationship type.
- Identification: This refers to the process whereby abstract concepts as well as concrete objects are made unique by means of some identifier.
- Classification and Instantiation: Classification involves classifying similar objects into object classes. The relationship between the object and the class is "IS_MEMBER_OF" relationship. The opposite of classification is called "instantiation." Some models allow this abstraction to be used among classes where a **metaclass** generically stands for a number of classes that are its members.
- Subclass and Superclass concept: The instances of one entity type may be a subset of the instances of another entity type. For example, entity type STUDENT is a subclass of entity type PERSON. The relationship between the two is called an "IS_A" relationship.
- Attribute Inheritance: A subclass automatically inherits the attributes of its superclass.
- Generalization hierarchies: These are hierarchies of classes where a superclass is related to a number of its subclasses, based on some distinguishing feature. For example, the PERSON superclass may have subclasses STUDENT and EMPLOYEE. The STUDENT subclass is further divided into GRAD_STUDENT and UNDERGRAD_STUDENT, whereas EMPLOYEE is divided into FULLTIME and PARTTIME. The whole class hierarchy represents the abstraction of generalization moving up the hierarchy and specialization moving down the hierarchy. Constraints governing the superclass vs. its subclasses can be modeled as total vs. partial and mandatory vs. optional participation [7].

The binary model of data, e.g., NIAM [37] is also fairly popular. Compared to the ER approach, the binary approach may be considered as a bottom-up approach, since it builds up from atomic objects in terms of binary relationships. For capturing higher-level semantics, they resort to a variety of constraints.

Another class of semantic data models is functional data models. They use objects and functions over objects as the basic building blocks. Any request for information can be visualized in terms of a functional call with arguments. Functions at the primitive level have objects as arguments; but it is possible to build functions over functions. The DAPLEX model [34] was one of the first prominent models in this category. Recently, the IRIS data model was developed [20] and fully implemented based on the concepts of objects and functions. The open ODB system of Hewlett-Packard based on the IRIS implementation will soon be commercially available. The main advantage of the functional modeling approach is its formal mathematical basis of functions which can be arbitrarily composed to give rise to higher-order functions. Combined with the preceding abstractions and the principles of the object-oriented approach, it makes a powerful data model. See [22] for a comparative treatment of semantic data models.

Object-Based Data Models

In this section we will allude to a number of "other" data models. One major area is object-oriented (O-O) data models. They are similar to semantic models in terms of the following:

- Structural abstraction: Both pro-

vide the structural abstractions that have been outlined. While inheritance is a must in all object-oriented models, it is not so in all semantic data models.
• Value and Type Composition: Both provide for type and value constructors that allow application designers to build higher forms of values or "bulk-types" such as lists, sets, bags, and tuples.

The areas in which O-O data models differ from the semantic data models are:

• Treatment of identifiers: Semantic models make up identifiers or keys based on internal attributes or internal properties. Object-oriented models use identifiers external to the object, so that identifiers remain invariant while the objects may undergo change. These identifiers are called "surrogates" or system identifiers.
• Treatment of inheritance: Semantic models realize only attribute inheritance and it is limited among subtypes and supertypes. O-O models, on the other hand, provide both structural and behavioral inheritance. This refers to inheriting the structural properties and the procedures or methods. Inheritance among types that are *not related* by a type-subtype relationship may also be allowed, but this is not recommended.
• Information-hiding and encapsulation: The methods encapsulated within an object type allow its instance variables to be accesses. There is no way to access these instance variables (or attributes)

Additionally, the O-O data models are expected to provide the following features [3] as a must:

• Inheritance of properties and methods
• Support for complex objects
• Object identity
• Operator overloading: This refers to the use of operations with identical names having different meaning in the context of different object types.
• Separation of public and private portions of objects.

As mentioned earlier in this article, we will not discuss object-oriented modeling and analysis in detail, since this entire issue is devoted to the topic. Several O-O data models have been proposed in the literature and have been implemented in systems similar to GemStone [17], IRIS [20], and O_2. A number of other commercial O-O DBMSs have also been introduced, including Object Design, ObjectStore, and Versant. It is hoped that these models will come to the rescue of users who will benefit from the following advantages normally attributed to the O-O models: better modeling of the behavior, easier modeling of complex objects, extensibility of types, and localization of the effect of changes in data definition. The object models may be regarded mainly as implementation models such as the hierarchical and network models. On the other hand, they may be viewed as semantic data models also. For a recent set of references in the O-O DBMS area see [39].

Dynamic and Active Database Models

The main argument that can be leveled against traditional data-modeling in the database area is that it has not adequately addressed the modeling of behavior of data, and hence the "active" or dynamic aspects of a database. The **active database paradigm** was proposed by Morgenstern [27] and has been later incorporated into the data models of systems such as POSTGRES at UC-Berkeley and Starburst [38] at IBM Almaden Center. An active DBMS is a system with the additional capability, in addition to a standard DBMS, of monitoring the state of the database and executing some predefined actions when some corresponding predefined events occur. The dynamic aspect can be further subdivided into two parts:

a. Process-oriented view: This relates to the ever-changing environment of an application. The entire set of applications to which a database is subjected can be considered as a processing environment with known processes or activities. This view can then be modeled and supported in the DBMS.
b. Event-oriented view: We previously pointed out the nature of integrity constraints, including dynamic constraints in the section "Scope of Data Models." The event-oriented view identifies different types of events occurring at runtime during the processing of a database (i.e., as a result of such events as insertion or deletion) and considers the actions needed to enforce the integrity constraints.

Rules have been proposed as a means of capturing the dynamic aspect of data [13, 38] and may be combined with existing standard models, such as the ER model, to give the latter an active capability [36]. We have proposed an alternate way, called the **structure-function paradigm** of dealing with functions on a par with structure in data modeling [18, 29]. In this work, it is proposed that functions be treated as first-class objects generating two separate class lattices for function and structure. This gives rise to complex structural objects and complex functional objects [30] that are themselves related, with a third group of objects, called the interaction objects. Such a modeling layer on top of the traditional O-O model gives the modeler a great amount of flexibility to deal with complex physical systems typical of engineering data (ex., a complex machine, a nuclear plant, or biological domains, including the human body). We have extended this approach with **control** as an additional component for explicit modeling [12], which is especially helpful in simulation of complex systems. In general, the data-modeling area has not done much to support storing of data to drive controlled simulations and to capture data generated from simulations. Incorporating these ideas into data models brings data models closer to AI systems that model behavior explicitly.

Current Advances

A number of advances in data-modeling, other than the object-oriented data models, and the active data-modeling area outlined previously are evident. One such area is applying work in knowledge representation, such as terminological reasoning, to data models, providing them with better reasoning power, making them easier for users to deal with. We have developed a data model CANDIDE [10] on this principle and have developed a number of applications using it, including a federated heterogeneous database prototype, a horticultural information retrieval package, a full-text retrieval system, and a multimedia natural language interface for information retrieval.

Another area is the application of the principles of machine-learning, so that schemas can be automatically designed from instances by a qualitative conceptual clustering process [1]. **Knowledge discovery** or "knowledge mining" refers to the extraction of knowledge that is not explicitly a part of either the request or the data in the database. We have implemented an interface to the CANDIDE data model to perform knowledge-mining by a successive generalization of a query [2], so that the users can be given some meaningful answer to an imprecise query. We are likely to see new approaches for dealing with imprecision in data as well as in requests for data along these lines.

Conclusions

This article traced the evolution of the data-modeling area from the era of files to the present. We proposed a classification of data models and pointed out their essential features. There has been some work on schema analysis and design methodologies for designing database schemas (e.g., [8]), which we were not able to discuss in detail. In terms of methodologies, we pointed out the need for doing joint data and application (or functional) modeling.

The trend toward object-oriented models continues. Due to the robustness of the relational DBMS implementations however, they will continue to dominate for several years to come. We also touched on the behavioral aspect of modeling, which is likely to see more work in the future.

Within the confines of this article, we are not able to do justice to a number of data models in the area of nested relational data models, temporal and spatial data models. For new application areas such as CAD/CAM, Geographic Information Systems (GIS), and genetic mapping, a number of interesting developments in the data-modeling area are expected. Moreover, efficient use of these models requires appropriate methodologies and tools, which are discussed elsewhere in this issue.

Acknowledgments
The author is thankful to the referees for some excellent suggestions on making this article more readable and complete. Several suggestions from the students in the Spring 1992 graduate database class at Georgia Tech have also been incorporated. ◘

References

1. Anwar, T., Beck, H.W., Navathe, S.B. Conceptual clustering in database systems. In *Proceedings of the Workshop of the American Society for Information Science.* Special Interest Group on Classification Research (Washington D.C., Oct. 27, 1991).
2. Anwar, T., Beck, H.W., Navathe, S.B. Knowledge mining by imprecise querying: A classification based approach. In *Proceedings of the IEEE International Conference on Data Engineering* (Phoenix, Az., Feb. 1992).
3. Atkinson, M.P., Bancilhon, F., Dewitt, D., Dittrich, K., Maier, D., Zdonik, S. The object-oriented database system manifesto. In *Proceedings of ACM SIGMOD Conference* (Atlantic City, 1990).
4. ANSI Study Group on Database Management Systems. Interim Report, FDT, vol. 7, no. 2, ACM, 1975.
5. Bachman, C. The data structure diagrams. Data Base (Bulletin of ACM SIGFIDET, vol. 1, no. 2, Mar. 1969.
6. Bancilhon, F. and Buneman, P., Eds. *Advances in Database Programming Languages.* ACM Press, New York, N.Y.
7. Batini, C., Lenzerini, M. and Navathe, S.B. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv. 18,* 4 (Dec. 1986).
8. Batini, C., Ceri, S. and Navathe, S.B. *Conceptual Database Design: An Entity Relationship Approach.* Benjamin Cummings, Redwood City, Calif., 1992.
9. Batory, D.S. Concepts for a database system synthesizer. In *Proceedings of International Conference on Principles of Database Systems,* 1988. Also in *Domain Analysis and Software Systems Modeling.* R. Prieto-Diaz and G. Arango, Eds., IEEE Computer Society Press, 1991.
10. Beck, H.W., Gala, S., Navathe, S.B. Classification as a query processing technique in the CANDIDE semantic data model. In *Proceedings of International Conference on Data Engineering* (Los Angeles, Calif. Feb. 1989).
11. Carswell, J.L., Jr. and Navathe, S.B. SA-ER: A methodology that links structured analysis and entity-relationship modeling for database design. In *Entity-Relationship Approach: Ten Years of Experience in Information Modeling.* S. Spaccapietra, Ed., North Holland, 1986.
12. Caselli, S., Papaconstantinou, C., Doty, L.L. and Navathe, S.B. A structure-function-control paradigm for knowledge-based modeling and simulation of manufacturing workcells. *J. Intelligent Manuf.,* Special Issue on Manufacturing Controls, *3,* 1 (Jan. 1992).
13. Chakravarthy, S., Navathe, S.B., Karlapalem, A., Tanaka, A. The cooperative problem solving challenge: A database centered approach. In *Co-operative Knowledge Based Systems,* S.M. Deen, Ed., Springer Verlag, pp. 30–52.
14. Chen, P.P.S. The entity-relationship model: Towards a unified view of data. *ACM Trans. Database Sys., 1,* 1 (1976).
15. CODASYL. *Report of the CODASYL*

Database Task Group, ACM, Apr. 1971.

16. Codd, E.F. A relational model for large shared data banks. *Commun. ACM 13,* 6 (June 1970).
17. Copeland, G. and Maier, D. Making Smalltalk a database system. In *Proceedings ACM SIGMOD Conference* (Boston, Mass., June 1984).
18. Cornelio, A., Navathe, S.B. and Doty, K. Extending object-oriented concepts to support engineering applications. In *Proceedings IEEE International Conference on Data Engineering* (Los Angeles, Feb. 1989).
19. Elmasri, R. and Navathe, S.B. *Fundamentals of Database Systems.* Benjamin Cummings, Redwood City, Calif., 1989.
20. Fishman, D.H., Beech, D., Cate, H.P. et al. IRIS: An object-oriented database system. *ACM Trans. Off. Inf. Syst. 5,* 1 (1987).
21. Hammer, M., Mcleod, D. Database description with SDM: A semantic database model. *ACM Trans. Database Syst. 19,* 3 (Sept. 1987).
22. Hull, R. and King, R. Semantic database modeling: Survey, applications, and research issues. *ACM Comput. Surv. 16,* 2 (June 1984).
23. Kent, W. Limitations of record-based information models. *ACM Trans. Database Syst. 4,* 1 (1979).
24. Lecluse, C., Richard, P. and Velez, F. O_2, an object-oriented data model. In *Proceedings ACM SIGMOD* (Chicago, June 1988).
25. Loomis, Mary E.S. *The Database Book.* Macmillan, 1987.
26. Markowitz, V. Representing processes in an extended entity-relationship model. In *Proceedings IEEE International Conference on Data Engineering* (Los Angeles, Feb. 1990).
27. Morgenstern, M. Active databases as a paradigm for enhanced computing environments. In *Proceedings of the Ninth International Very Large Database Conference* (Florence, Italy, Oct. 1983).
28. Navathe, S.B. Schema analysis for database restructuring. *ACM Trans. Database Syst. 5,* 2 (June 1980).
29. Navathe, S.B. and Cornelio, A. Modeling engineering data by complex structural objects and complex functional objects. In *Proceedings of the International Conference on Extending Database Technology* (Venice, Italy, Mar. 1990), Springer Verlag Notes in Computer Science, no. 416.
30. Navathe, S.B. and Balaraman, A. A transaction architecture for a general purpose semantic data model. In *Proceedings of the Tenth International Conference on Entity-Relationship Approach* (San Mateo, Calif., Oct. 91), ER Institute.
31. Ngu, Anne H.H. Conceptual transaction modeling. *IEEE Trans. Knowledge and Data Engineering 1,* 4 (Dec. 1989).
32. NIST. A planning model for unifying information modeling languages for product data exchange specification (PDES). Joan Tyler. Rep. NISTIR 90-4234, Sept. 1990.
33. Scheuermann, P., Schiffner, G., Weber, H. Abstraction capabilities and invariant properties modeling within the entity-relationship approach. In *Entity-Relationship Approach to System Analysis and Design,* P.P.S. Chen, Ed., North Holland, 1980.
34. Shipman, D. The functional data model and the data language DAPLEX. *ACM Trans. Database Syst. 6,* 1 (Mar. 1981).
35. Smith, J.M. and Smith, D.C.P. Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst. 2,* 2 (Mar. 1981), 106–133.
36. Tanaka, A., Navathe, S.B., Chakravarthy, S., Karlapalem, A. ER-R, an enhanced ER model with situation-action rules to capture application semantics. In *Proceedings of Tenth International Confernce on Entity-Relationship Approach* (San Mateo, Calif., Oct. 91).
37. Verheijen, G., VanBekkum, J. NIAM: An information analysis method. In *Information System Design Methodology,* T. Olle, H. Sol, A. Verrijn-Stuart, Eds., North Holland, Amsterdam, 1982.
38. Widom, J. and Finkelstein, S.J. Set-oriented production rules in relational database systems. In *Proceedings ACM SIGMOD Conference* (Atlantic City, 1990).
39. Zdonik, S. and Maier, D., Eds., *Readings in Object-oriented Database Systems.* Morgan Kaufmann, 1990.

CR Categories and Subject Descriptors: H.1 [**Information Systems**]: Models and Principles; H.2 [**Information Systems**]: Database Management

General Terms: Design

Additional Key Words and Phrases: Database design, data model database, schema design

About the Author:
SHAMKANT B. NAVATHE is a professor in the College of Computing at the Georgia Institute of Technology. Current research interests include distributed database design, object-oriented databases, active databases, information management in heterogeneous and multimedia environments, and manufacturing applications. **Author's Present Address:** College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280; email: sham@cc.gatech.edu