

Chapter 2

LOGICAL DATABASE MODELS

The evolution of database systems was initially driven by the requirements of traditional data processing. Hierarchical and network data models were adopted by database management systems (DBMS) as database models in the 1960s and 1970s. The hierarchical and network data models have the drawbacks that the data models couple with the need for a formally based database model, which clearly separate the physical and logical model. Relational database model, put forward by E. F. Codd in 1970s (Codd, 1970), has a simple structure and a solid mathematical foundation. It rapidly replaced the hierarchical and network database models and became the dominant database model for commercial database systems.

With the breadth and depth of database uses in many emerging areas as diverse as biology and genetics, artificial intelligence, computer aided design, and geographical information systems, it was realized that the relational database model as defined by Codd, had semantic and structured drawbacks when it came to modeling of such specialized applications. The next evolution of database models took the form of rich data models such as the object-oriented data model (Abiteboul, Hull and Vianu, 1995; Elmasri and Navathe, 1994; Kim and Lochovsky, 1989) and the semantic data models (Abiteboul and Hull, 1987; Elmasri and Navathe, 1994; Hammer and McLeod, 1981).

Relational database model and object-oriented database model are typical the representatives of the logical database models. Based on these two basic database models, there exists a kind of hybrid database model called object-relational database model. In addition, new developments in artificial intelligence and procedure control have resulted in the appearances of deductive databases, active databases, temporal databases, and spatial

databases. These databases generally adopt either one of the above-mentioned two basic database models or a hybrid database model.

2.1 The Relational Database Model

Relational database model introduced first by Codd (1970) is the most successful one and relational databases have been extensively applied in most information systems in spite of the increasing populations of object-oriented databases. A relational database is a collection of relations.

2.1.1 Attributes and Domains

The representations for some features are usually extracted from real-world things. The features of a thing are called *attributes*. For each attribute, there exists a range that the attribute takes values, called *domain* of the attribute. A domain is a finite set of values and every value is an atomic data, the minimum data unit with meanings.

2.1.2 Relations and Tuples

Let A_1, A_2, \dots, A_n be attribute names and the corresponding attribute domains be D_1, D_2, \dots, D_n (or $\text{Dom}(A_i)$, $1 \leq i \leq n$), respectively. Then relational schema R is represented as

$$R = (D_1/A_1, D_2/A_2, \dots, D_n/A_n)$$

or

$$R = (A_1, A_2, \dots, A_n),$$

where n is the number of attributes and is called the degree of relation.

The instances of R , expressed as r or $r(R)$, are a set of n -tuples and can be represented as $r = \{t_1, t_2, \dots, t_m\}$. A tuple t can be expressed as $t = \langle v_1, v_2, \dots, v_n \rangle$, where $v_i \in D_i$ ($1 \leq i \leq n$), i.e., $t \in D_1 \times D_2 \times \dots \times D_n$. The quantity r is therefore a subset of Cartesian product of attribute domains, i.e., $r \subseteq D_1 \times D_2 \times \dots \times D_n$. Viewed from the content of a relation, a relation is a simple table, where tuples are its rows and attributes are its columns. Note that there is no complex data in relational table. The value of a tuple t on attribute set S is generally written $t[S]$, where $S \subseteq R$.

2.1.3 Keys

If an attribute value or the values of an attribute group in a relation can solely identify a tuple from other tuples, the attribute or attribute group is called a *super key* of the relation. If any proper subsets of a super key are not a super key, such super key is called a *candidate key* or shortly *key*.

For a relation, there may be several candidate keys. One chooses one candidate as the *primary key*, and other candidates are called *alternate key*. It is clear that the values of primary key of all tuples in a relation are different and are not null. The attributes included in a candidate key are called *prime attributes* and not included in any candidate key called *non-prime attributes*. If an attribute or an attribute group is not a key of relation r but it is a key of relation s , such attribute (group) is called *foreign key* of relation r .

2.1.4 Constraints

There are various constraints in the relational databases. We identify these constraints as follows.

- (a) *Domain integrity constraints*. The basic contents of domain integrity constraints are that attribute values should be the values in the domains. In addition, domain integrity constraints are also prescribed if an attribute value could be null.
- (b) *Entity integrity constraints*. Every relation should have a primary key and the value of the primary key in each tuple should be sole and cannot be null.
- (c) *Referential integrity constraints*. Let a relation r have a foreign key FK and the foreign key value of a tuple t in r be $t[FK]$. Let FK quote the primary key PK of relation r' and t' be a tuple in r' . Referential integrity constraint demands that $t[FK]$ comply with the following constraint: $t[FK] = t'[PK]/\text{null}$.
- (d) *General integrity constraints*. In addition to the above-mentioned three kinds of integrity constraints that are most fundamental in relational database model, there are other integrity constraints related to data contents directly, called *general integrity constraints*. Because numbers of them are very large, only a few of them are considered in current relational DBMSs. Among these constraints, *functional dependencies (FD)* and *multivalued dependencies (MVD)* are more important in relational database design theory and widely investigated.

The functional dependencies (FD) in relational databases represent the dependency relationships among attribute values in relation. In the relational databases, functional dependencies can be defined as follows.

Definition: For a relation $r(R)$, in which R denotes the schema, its attribute set is denoted by U , and $X, Y \subseteq U$, we say r satisfies the functional dependency $FD: X \rightarrow Y$, if

$$(\forall t \in r) (\forall s \in r) (t[X] = s[X] \Rightarrow t[Y] = s[Y]).$$

Based on the concept of functional dependency, the partial/full functional dependencies and the transitive functional dependency can be defined as follows.

Definition: For a relation $r(R)$, in which R denotes the schema, its attribute set is denoted by U , and $X, Y \subseteq U$, we say Y is fully functionally dependent on X , denoted by $X \rightarrow_f Y$, if and only if $X \rightarrow Y$ and there does not exist $X' \subset X$ ($X' \neq \Phi$) such that $X' \rightarrow Y$. If such X' exists, then Y is partially functionally dependent on X , denoted by $X \rightarrow_p Y$.

The notion of keys can consequently be defined in terms of FD s.

Definition: For a relation $r(R)$, in which R denotes the schema, its attribute set is denoted by U , and $K \subseteq U$, we say K is a candidate key of R if and only if $K \rightarrow_f U$.

Multivalued dependencies (MVD) originated by Fagin (1977) are another important data dependencies that are imposed on the tuples of relational databases, relating an attribute value or a set of attribute values to a set of attribute values, independent of the other attributes in the relation. In classical relational databases, multivalued dependencies can be defined as follows.

Definition: For a relation $r(R)$, in which R denotes the schema, its attribute set is denoted by U , $X, Y \subseteq U$, and $Z = U - XY$, we say r satisfies the multivalued dependency $MVD: X \twoheadrightarrow Y$, if

$$(\forall t \in r) (\forall s \in r) (t[X] = s[X] \Rightarrow (\exists u \in r) (u[X] = t[X] \wedge u[Y] = t[Y] \wedge u[Z] = s[Z])).$$

In the relational databases, the functional and multivalued dependencies satisfy the inference rules, namely, the *axiom systems* (Armstrong, 1974; Beeri, Fagin and Howard, 1977). For the functional dependency, for example, the *Armstrong axioms* (1974) can be used to derive all possible FD s implied by a given set of dependencies. Let $r(R)$ be a relation on schema R , its attribute set be denoted by U , and $X, Y, Z \subseteq U$. Then the following is a set of Armstrong axioms.

- (a) *Inclusion rule:* if $X \supseteq Y$, then $X \rightarrow Y$.
- (b) *Transitivity rule:* if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.
- (c) *Augmentation rule:* If $X \rightarrow Y$, then $X \cup Z \rightarrow Y \cup Z$.

2.1.5 The Relational Algebra

Relational database model provides some operations, called the *relational algebra operations*. These operations can be subdivided into two classes:

- (a) the operations for relations only (*select, project, join, and division*) and
- (b) the set operations (*union, difference, intersection, and Cartesian product*).

In addition, some new operations such as *outerjoin, outerunion* and *aggregate* operations are developed for database integration or statistics and decision support. By using these operations, one can query or update relations.

Union (\cup)

Union is a binary set operation on two relations that are union-compatible. That means they have the same number of attributes with the same domains pairwise. Formally let r and s be two union-compatible relations on the scheme $R(A_1, A_2, \dots, A_n)$. Then

$$r \cup s = \{t \mid t \in r \vee t \in s\}$$

It is clear that the result of $r \cup s$ is a relation on the schema R that includes all tuples which are either in r or in s or in both r and s . Of course, duplicate tuples, if any, must be eliminated.

Difference ($-$)

Difference is a binary set operation on two relations that are union-compatible. Formally let r and s be two union-compatible relations on the scheme $R(A_1, A_2, \dots, A_n)$. Then

$$r - s = \{t \mid t \in r \wedge t \notin s\}$$

It can be seen that the result of $r - s$ is a relation on the schema R that includes the tuples which are only in r but not in s .

Cartesian product (\times)

Cartesian product is a binary set operation on two relations. Formally let r and s be two fuzzy relations on schema R and S , respectively. Then

$$r \times s = \{t(R \cup S) \mid t[R] \in r \wedge t[S] \in s\}$$

That is, the result of $r \times s$ is a relation on the schema $R \cup S$, in which a tuple is a combination of a tuple from r and a tuple from s . So $|r \times s| = |r| \times |s|$. Here $|r|$ denotes the number of tuples in r .

Projection (Π)

Projection is a unary operation on a relation. Formally let r be relation on the scheme $R (A_1, A_2, \dots, A_n)$. Then the projection of r over attribute subset $S (S \subset R)$ is defined as follows.

$$\Pi_S (r) = \{t (S) \mid (\forall x) (x \in r (S) \wedge t = x [S])\}$$

In other words, the result of $\Pi_S (r)$ is a relation on the schema S that only includes the columns of relational table r which are given in S . It should be noticed that, if the attributes in S are all non-key attributes of $r (R)$, duplicate tuples may appear in $\Pi_S (r)$ and must be eliminated.

Selection (σ)

Selection is a unary operation on a relation. Formally let r be relation on the scheme $R (A_1, A_2, \dots, A_n)$. Then the selection of r based on a selection condition P specified by a Boolean expression in forms of a single or composite predicate is defined as follows.

$$\sigma_P (r) = \{t \mid t \in r \wedge P (t)\}$$

Clearly, the result of $\sigma_P (r)$ is a relation on the schema R that only includes the tuples in r which satisfy the given selection condition P .

The five relational operations given above are called the primitive operations in the relational databases. In addition, there are three additional relational operations, namely, intersection, join (and natural join), and division. But the three operations can be defined by the primitive operations.

Intersection (\cap)

Intersection is a binary set operation on two relations that are union-compatible. Formally let r and s be two union-compatible relations on the scheme $R (A_1, A_2, \dots, A_n)$. Then

$$r \cap s = \{t \mid t \in r \wedge t \in s\} \text{ or } r \cap s = r - (r - s)$$

The result of $r \cap s$ is a relation on the schema R that includes the tuples which are both in r and in s .

Join (\bowtie)

Join is a binary set operation on two relations. Formally let $r (R)$ and $s (S)$ be any two relations. Let P be a conditional predicate in the form of $A \theta B$, where $\theta \in \{>, <, \geq, \leq, =, \neq\}$, where $A \in R$, and $B \in S$. Then

$$r \bowtie_P s = \{t (R \cup S) \mid t [R] \in r \wedge t [S] \in s \wedge P (t [R], t [S])\} \text{ or}$$

$$r \bowtie_p s = \sigma_p (r \times s)$$

The result of $r \bowtie_p s$ is a relation on the schema $R \cup S$, in which a tuple is a combination of a related tuple from r and a related tuple from s . Not being the same as the Cartesian product operation, the two combined tuples respectively from r and s must satisfy the given condition.

When attributes A and B are identical and “ θ ” takes $=$, the join operation becomes the natural join operation, denoted $r \bowtie s$. Let $Q = R \cap S$. Then

$$r \bowtie s = \{t (R \cup (S - Q)) \mid (\exists x) (\exists y) (x \in r(R) \wedge y \in s(S) \wedge x[Q] = y[Q] \wedge t[R] = x[R] \wedge t[S - Q] = y[S - Q])\}$$

Division (\div)

Division, referred to quotient operation sometimes, is used to find out the sub-relation $r \div s$ of a relation r , containing sub-tuples of r which have for complements in r all the tuples of a relation s . Then the division operation is defined by

$$r \div s = \{t \mid (\forall u) (u \in s \wedge (t, u) \in r)\},$$

where u is a tuple of s and t is a sub-tuple of r such that (t, u) is a tuple of r .

Alternatively, let $r(R)$ and $s(S)$ be two relations, where $S \subset R$. Let $Q = R - S$. Then the division of r and s can be defined as follows.

$$r \div s = \Pi_Q (r) - \Pi_Q (r(Q) \times s - r)$$

2.1.6 Relational Database Design

Overall Design of Databases

The objective of database design is to capture the essential aspects of some real-world enterprise for which one wishes to construct a database (Petry, 1996). Figure 2-1 shows a simplified description of the database design process (Elmasri and Navathe, 1994). Then four major steps are applied for the database design process, which are the *requirements collection & analysis*, *conceptual data modeling*, *logical database model*, and *physical database model*, respectively.

In the first step, the database designers collect and analyze the data requirements from prospective database users. As a result of this step, a concisely written set of users' requirements is formed.

In the second step, the conceptual data models (e.g., ER/EER and UML) are used to create a conceptual schema for the database. Here, the conceptual

schema is a concise descriptions of the data requirements of the users and includes detailed descriptions of the data types, relationships, constraints, and etc. But there are no any implementation details in the conceptual schema. So it should be easy to share the conceptual schema with non-technical users. It is worth mentioning that a complex database is generally designed cooperatively by a design group and each member of the group may have different background. So using multiple conceptual data models to create the conceptual schema can facilitate the database designers with different background to design their conceptual data schemas easily by using one of the conceptual data models they are familiar with. But finally all these conceptual schemas designed by different members should be converted into a union conceptual schema. There are already some efforts for converting different conceptual schemas (Cherfi, Akoka and Comyn-Wattiau, 2002).

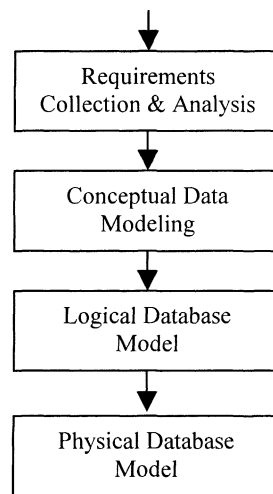


Figure 2-1. Database Design Process

In the third step, the logical database model is designed through mapping the conceptual schema represented by the conceptual data model. The result of this step is perhaps a relational or object-oriented database model. In (Teorey, Yang and Fry, 1986), for example, relational databases were logically designed using the ER model.

Finally, in the fourth step, the physical database model is design. Of course, this step is mostly already formulated with a commercial DBMS.

Relational Database Design Theory

In the context of relational databases, the relational database model should be designed in terms of a set of *good* schemas such that update

anomalies and data redundancy are minimized. Here update anomalies mean that the undesired consequences occur when updating the data in the relational databases (e.g., inserting, deleting or modifying the tuples). The reason is that there exist certain undesired dependency relationships between the attributes of a relation.

The relational database design theory has been developed for minimizing update anomalies and data redundancy, which core is the normalization theory. The process of normalization is the process of relation schema decomposition so that certain undesired dependency relationships are removed to lead to certain *normal forms* (NFs).

Let $r(R)$ be a relation on schema R , U be the attribute set of R , and $X, K, A \subseteq U$. Here K is the candidate key of R . Then we have the following major NFs in the relational databases.

- (a) *The first normal form* (1NF): R is in 1NF, denoted by $R \in 1NF$, if and only if every attribute value in $r(R)$ is atomic.
- (b) *The second normal form* (2NF): R is in 2NF, denoted by $R \in 2NF$, if and only if $R \in 1NF$ and for any non-prime attribute A , $K \rightarrow_f A$.
- (c) *The third normal form* (3NF): R is in 3NF, denoted by $R \in 3NF$, if and only if $R \in 1NF$ and for any $X \rightarrow A$ ($A \not\subseteq X$), either X is a superkey of R or A is a set of prime attributes.
- (d) *The Boyce-Codd normal form* (BCNF): R is in BCNF, denoted by $R \in BCNF$, if and only if $R \in 1NF$ and for any $X \rightarrow A$ ($A \not\subseteq X$), either X is a superkey of R .

A lower NF can be normalized into a higher NF through relation schema decomposition (via projection). Figure 2-2 shows the details (Chen 1999).

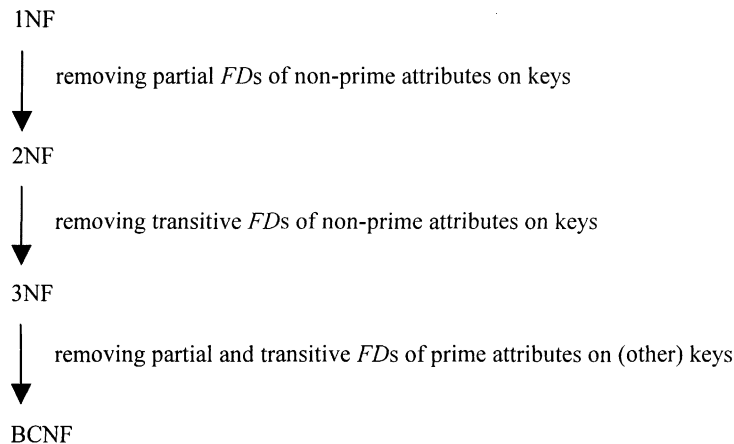


Figure 2-2. Normal Forms Based on Functional Dependencies

It should be noticed that the schema decomposition should satisfy the following properties:

- (a) *lossless-join*. It means that the relation reconstructed from the resultant relations of the decomposition will be the same as the original relation with respect to information contents.
- (b) *dependency-preservation*. It means that the functional dependencies in the original relation are preserved by the resultant relations of the decomposition.

The four NFs discussed above are based on the functional dependency. In addition, there are other kinds of normal forms such as the *fourth normal form* (4NF) and the *fifth normal form* (5NF), which are related with multivalued dependency and join dependency, respectively.

2.2 The Nested Relational Database Model

The normalization, being one kind of constraints, is proposed in traditional relational databases. Among various normalized forms, first normal form (1NF) is the most fundamental one, which assumes that each attribute value in a relational instance must be atomic. As we know, the real-world applications are complex, and data types and their relationships are rich as well as complicated. The 1NF assumption limits the expressive power of traditional relational database model. Therefore, some attempts to relax 1NF limitation are made and one kind of data model, called non-first normal (or nested) relational database model have been introduced,

The first attempt to relax first-normal formal limitation is made by Makinouchi (1977), where, attribute values in the relation may be atomic or set-valued. Such relation is thereby called non-first normal form (NF^2) one. After Makinouchi's proposal, NF^2 database model is further extended (Ozsoyoglu, Ozsoyoglu and Matos, 1987; Schek and Scholl, 1986). The NF^2 database model in common sense now means that attribute values in the relational instances are either atomic or set-valued and even relations themselves. So NF^2 databases are called nested relational databases also. In this paper, we do not differentiate these two notions. A formal definition of NF^2 relational schema is given as follows.

Definition. An attribute A_j is a structured attribute if its schema appears on the left-hand side of a rule; otherwise it is simple.

Definition. An NF^2 relational schema may contain any combination of simple or structured attributes on the right-hand side of the rules. Formally,

Schema:: Simple_attribute | Simple_attribute, Structured_attributes

Structured_attributes:: Simple_attribute | Simple_attribute,
Structured_attributes

A schema is called flat if and only if all of its attributes are simple. It is clear that a classical schema, namely, a flat relational schema, is a special case of a nested relational schema. Two nested schemas are called union-compatible, meaning the ordered attributes have the same nesting structure, if and only if the corresponding simple attributes and structured attributes are union-compatible.

Let a relation r have schema $R = (A_1, A_2, \dots, A_n)$ and let D_1, D_2, \dots, D_n be corresponding domains from which values for attributes (A_1, A_2, \dots, A_n) are selected. A tuple of an NF^2 relation is an element in r and denoted as $\langle a_1, a_2, \dots, a_n \rangle$ consisting of n components. Each component a_j ($1 \leq j \leq n$) may be an atomic or null value or another tuple. If A_j is a structured attribute, then the value a_j need not be a single value, but an element of the subset of the Cartesian product of associated domains $D_{j1}, D_{j2}, \dots, D_{jm}$.

Based on the NF^2 database model, the ordinary relational algebra has been extended. In addition, two new restructuring operators, called the *Nest* and *Unnest* (Ozsoyoglu, Ozsoyoglu and Matos, 1987; Roth, Korth and Batory, 1987) (as well as *Pack* and *Unpack* (Ozsoyoglu, Ozsoyoglu and Matos, 1987)), have been introduced. The Nest operator can gain the nested relation including complex-valued attributes. The Unnest operator is used to flatten the nested relation. That is, it takes a relation nested on a set of attributes and desegregates it, creating a “flatter” structure. The formal definitions and the properties of these operators as well as the ordinary relational algebra for the NF^2 data model have been given (Colby, 1990; Venkatramen and Sen, 1993).

2.3 The Object-Oriented Database Model

Although there has been great success in using the relational databases for transaction processing, the relational databases have some limitations in some non-transaction applications such as computer-aided design and manufacturing (CAD/CAM), knowledge-based systems, multimedia, and GIS. Such limitations include the following.

- (a) The data type is very restricted.
- (b) The data structure based on the record notion may not match the real-world entity.
- (c) Data semantics is not rich, and the relationships between two entities cannot be represented in a natural way.

Therefore, some non-traditional data models were developed in succession to enlarge the application area of databases since the end of the 1970s. Since

these non-traditional data models appeared after the relational data model, they are called post-relational database models. Object-oriented database model is one of the post-relational database models.

Object-oriented (OO) data model is developed by adopting some concepts of semantic data models and knowledge expressing models, some ideas of object-oriented program language and abstract data type in data structure/programming.

2.3.1 Objects and Identifiers

All real-world entities can be simulated as *objects*, which have no unified and standard definition. Viewing from the structure, an object consists of *attributes*, *methods* and *constraints*. The attributes of an object can be simple data and other objects. The procedure that some objects constitute a new object is called *aggregation*. A method in an object contains two parts: signature of the method that illustrates the name of the method, parameter type, and result type; implementation of the method.

In general, attributes, methods and constraints in an object are encapsulated as one unit. The state of an object is changed only by passing message between objects. *Encapsulation* is one of the major features in OO data models.

In OO data models, each object has a sole and constant identifier, which is called *object identifier* (OID). For two objects with same attributes, methods and constraints, they are different objects if they have a different OID. The OID of an object is generated by system and cannot be changed by the user.

The OID generated by system can be divided into two kinds, i.e., *logical object identifier* and *physical object identifier*. Logical object identifier is mapped into physical one when an object is used because only physical object identifier concerns the storage address of the object.

2.3.2 Classes and Instances

In OO data models, objects with the same attributes, methods and constraints can be incorporated into a *class*, where objects are called *instances*. In a class, attributes, methods and constraints should be declared. Note that the attributes in a class can be classified into two kinds: instance variables and class variables. Instance variables are the attributes for which values are different in different objects of the class, while class variables are the attributes for which values are the same in different objects of the class.

In fact, classes can also be regarded as objects. Then, classes can be incorporated into another new class, called *meta class*. The instances of a

meta class are classes. Therefore, objects are distinguished into *instance objects* and *class objects*.

2.3.3 Class Hierarchical Structure and Inheritance

A subset of a class, say A, can be defined as a class, say B. Class B is called a *subclass* and class A is called *superclass*. A subclass can further be divided into new subclasses. A *class hierarchical structure* is hereby formed, where it is permitted that a subclass has several direct or indirect superclasses. The relationship between superclass and subclass is called *IS-A relationship*, which represents a *specialization* from top to bottom and a *generalization* from bottom to top. Because one subclass can have several direct superclasses, a class hierarchical structure is not a tree but a class *lattice*.

Because a subclass is a subset of its superclass, the subclass inherits the attributes and methods in its all superclasses. Besides inheritance, a subclass can define new attributes and methods or can modify the attributes and methods in the superclasses. If a subclass has several direct superclasses, the subclass inherits the attributes and methods from these direct superclasses. This is called *multiple inheritance*.

When inheriting, the naming conflict may occur, which should be resolved.

- (a) Conflict among superclasses. If several direct superclasses of a subclass have the same name of attributes or methods, the conflict among superclasses appear. The solution is to declare the superclass order inherited, or to be illustrated by user.
- (b) Conflict between a superclass and a subclass. When there are conflicts between a subclass and a superclass, the definition of attributes and methods in subclass would override the same definition in the superclass.

Note that a naming method may have a different meaning in different classes. The feature that a name has a multiple meaning is called *polymorphism*. The method with polymorphism is called *overloading*. Because the method in an object is polymorphism, the procedure corresponding to the method name cannot be determined while compiling program and do while running program. The later combination of the method name and implementing procedure of a method is called *late binding*.

References

Abiteboul, S. and Hull, R., 1987, IFO: A formal semantic database model, *ACM Transactions on Database Systems*, 12 (4): 525-565.

- Abiteboul, S., Hull, R. and Vianu, V., 1995, *Foundations of Databases*, Addison Wesley.
- Armstrong, W. W., 1974, Dependency structures of data base relationships, *Proceedings of the IFIP Congress*, 580- 583.
- Beeri, C., Fagin, R. and Howard, J. H., 1977, A complete axiomatization for functional and multivalued dependencies in database relations, *ACM SIGMOD Conference*, 47-61.
- Chen, G. Q., 1999, *Fuzzy Logic in Data Modeling; Semantics, Constraints, and Database Design*, Kluwer Academic Publisher.
- Cherfi, S. S., Akoka, J. and Comyn-Wattiau, I., 2002, Conceptual modeling quality: from EER to UML schemas evaluation, *Lecture Notes in Computer Science*, 2503: 414-428.
- Codd, E.F., 1970, A relational model of data for large shared data banks, *Communications of the ACM*, 13 (6): 377-387.
- Colby, L. S., 1990, A recursive algebra for nested relations, *Information Systems*, 15 (5): 567-662.
- Elmasri, R. and Navathe, S. B., 1994, *Fundamentals of Database Systems*, Second Edition, Benjamin/Cummings.
- Fagin, R., 1977, Multivalued dependencies and a new normal form for relational databases, *ACM Transactions on Database Systems*, 2 (3): 262-278.
- Hammer, M. and McLeod, D., 1981, Database description with SDM: a semantic database model, *ACM Transactions on Database Systems*, 6 (3): 351-386.
- Kim, W. and Lochovsky, F. H., 1989, *Object-Oriented Concepts, Databases and Applications*, Addison Wesley.
- Makinouchi, A., 1977, A consideration on normal form of not-necessarily normalized relations in the relational data model, *Proceedings of Third International Conference on Very Large Databases*, Tokyo, Japan, October, 447-453.
- Ozsoyoglu, G., Ozsoyoglu, Z. M. and Matos, V., 1987, Extending relational algebra and relational calculus with set-valued attributes and aggregate functions, *ACM Transactions on Database Systems*, 12 (4): 566-592.
- Petry, F. E., 1996, *Fuzzy Databases: Principles and Applications*, Kluwer Academic Publisher.
- Roth, M. A., Korth, H. F. and Batory, D. S., 1987, SQL/NF: A query language for non-1NF relational databases, *Information Systems*, 12: 99-114.
- Schek, H. J. and Scholl, M. H., 1986, The relational model with relational-valued attributes, *Information Systems*, 11 (2): 137-147.
- Teorey, T. J., Yang, D. Q. and Fry, J. P., 1986, A logical design methodology for relational databases using the extended entity-relationship model, *ACM Computing Surveys*, 18 (2): 197-222.
- Venkatraman, S. S. and Sen, A., 1993, Formalization of an IS-A based extended nested relation data model, *Information Systems*, 20 (1): 53-57.



<http://www.springer.com/978-0-387-24249-1>

Fuzzy Database Modeling with XML

Ma, Z.

2005, XXIV, 216 p. 74 illus.,

ISBN: 978-0-387-24249-1