CHAPTER 1.2

Introduction to Databases

Wen-Hsiang Kevin Liao and Dennis McLeod
Computer Science Department, University of Southern California, Los Angeles, California

Abstract

Databases based on relational, object-oriented, and object-relational models represent significant advances in database technologies. In the context of general-purpose database management systems, the fundamentals of database models are examined. A historical perspective on the evolution of major database models is provided. The principal concepts underlying relational, object-oriented, and object-relational database models are presented with examples. Finally, a brief view of database federation issues is introduced that serves as the foundation for discussion later in the book.

Computerized databases are essential and inseparable components of most of today's information systems. Database systems are used at all levels of management, research, and production to provide uniform access to and control of consistent information. In the past few decades, we have witnessed enormous advances in database technologies. The goal of this chapter is to provide a brief and informal overview of state-of-the-art database systems and database federation issues. The emphasis is on important concepts of relational, object-based, and object-relational databases and the relationships among them.

In the first section, the components of databases and their functional capabilities are introduced, followed by a brief historical perspective on the developments of relational, object-based, and object-relational databases. The structures, constraints, and operations of the relational database model are then introduced, and an informal presentation of major SQL features is given. A discussion of the concepts of the object-based database model follows, and the principal concepts of the object-relational database model are presented and its relationships with relational and object-based databases are discussed. Finally, a brief view of database federation issues is introduced.

1.2.1 An Overview of Database Management

A *database* (DB) is a collection of structured (organized), interrelated information units (objects). An *information unit* is a package of information at various levels of granularity. An information unit could be as simple as a string of letters forming the name of an experimenter or a data value collected from an experiment. It could also be as complex as the protocol of an experiment, a published paper, the image of a rat brain, the audio clip of a speech, or the video clip of an experiment in a laboratory. Every database is a model of some real world system. At all times, the contents of a database are intended to represent a snapshot of the state of an application environment, and each change to the database should reflect an event (or sequence of events) occurring in the environment. A database can be of any size and of varying complexity. For example, a database can contain the information of only a few hundred people working on the same project, or it could contain the information of a bank, an airline company, or data collected from scientific experiments.

A general-purpose database management system (DBMS) can be viewed as a generalized collection of integrated mechanisms and tools to support the definition, manipulation, and control of databases for a variety of application environments. In particular, a general-purpose DBMS is intended to provide the following functional capabilities:

1. Support the independent existence of a database, apart from the application programs and systems that manipulate it.
2. Provide a conceptual/logical level of data abstraction.
3. Support the query and modification of databases.
4. Accommodate the evolvability of both the conceptual structure and internal (physical) organization of a database, in response to changing information, usage, and performance requirements.
5. Control a database, which involves the four aspects of semantic *integrity* (making sure the database is an accurate model of its application environment), *security* (authorization), *concurrency* (handling multiple simultaneous users), and *recovery* (restoring the database in the event of a failure of some type).

At the core of any database system is a *database model* (data model), which is a mechanism for specifying the structure of a database and operations that can be performed on the data in that database. As such, a database model should:

1. Allow databases to be viewed in a manner that is based on the meaning of data as seen by their users.
2. Accommodate various levels of abstraction and detail.
3. Support both anticipated and unanticipated database uses.
4. Accommodate multiple viewpoints.
5. Be free of implementation and physical optimization detail (physical data independence).

Abstractly speaking, a database model is a collection of generic structures, semantic integrity constraints, and primitive operations. The structures of a database model must support the specification of objects, object classifications, and inter-object relationships. The semantic integrity constraints of the database model specify restrictions on states of a database or transitions between such states so that the database accurately reflects its application environment. Some constraints are embedded within the structural component of a database model, while others may be expressed separately and enforced externally to the DBMS. The specification of a particular database constructed using these general-purpose structures and constraints can be referred to as a (conceptual) *schema*.

The operational component of a database model consists of a general-purpose collection of primitives that support the query and modification of a database; namely, given a database with an associated conceptual schema, the operations facilitate the manipulation of that database in terms of the schema. Such primitives may be embodied in a stand-alone end-user interface or a specialized language or embedded within a general-purpose programming language. Database-specific operations can be constructed using the primitives of the database model as building blocks.

1.2.2 Historical View of Key Database Developments

To provide a historical perspective on relational, object-based, and object-relational databases models and systems, Fig. 1 shows a 30-year timeline of their developments (McLeod, 1991). Note that there are many database models and systems being developed during this period of time. For the purpose of our discussion, only those relevant database models and systems are included in the figure.

The relational database model was first introduced in 1970 (Codd, 1970). It is based on a simple and uniform data structure, the relation, and has a solid mathematical foundation. Many commercial products of relational DBMSs were implemented in the 1980s. The descendants of these relational DBMSs are still dominating the database market around the world. The semantic/object-based database models adopt objects and classes as their basic structures and add many semantic primitives such as specialization/generalization, data abstraction, and encapsulation. Several semantic/object-based database systems were developed in the 1990s. The object-relational database model extends the relational database model with abstract data types and primitives from semantic/object-based data models. Its basic structure is still a relation, but the value contained in a cell of the relation could be of any abstract data type in addition to atomic data types like strings and numbers. The specialization/generalization of both relations and data

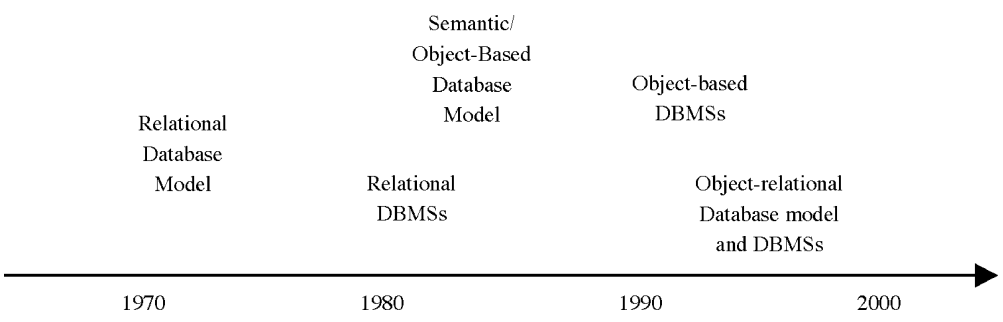

Figure 1 Historical view of key DBMS developments.

types are also incorporated into the object-relational model.

1.2.3 Relational Database Model

Simplicity and uniformity are the main characteristics of the relational database model. To introduce the relational database model, the basic structures and constraints of the relational model can be considered (Codd, 1970; Chen, 1976). The operational component of the relational model will be illustrated later in the section on the SQL query language. A relation can be crudely viewed as a table containing a fixed number of columns (attributes) and a variable number of rows (tuples). (Strictly speaking, a table depiction of a relation is an approximation; for example, rows and columns are unordered, and no duplicate rows are allowed.) We say a relation is *n*-ary if there are *n* columns in the table. Each column has a "domain" comprising the possible entries for that column. Thus, a relational database consists of a number of *n*-ary relations and a collection of underlying domains. Fig. 2 shows an example relation named FRIENDS which contains information on four friends. The relation is presented in tabular form. The relation has three columns: Name, Age, and Telephone. Each row represents information on a friend. For example, the first row of the relation indicates that Joe's age is 48 and his telephone number is 740–4312. The domain of a column specifies possible values a column entry may take. In this example, the domains of Name and Telephone columns are strings, and the domain of the Age column is numbers. In the relational model, the domains consist of atomic values, which may be built in (e.g., Numbers, Strings) or user-defined (e.g., Phone-numbers). Note that the user-defined domain is a subset of one of the built-in domains. For instance, the domain of the Phone-numbers column is the set of strings that represent meaningful telephone numbers. Each database may contain as many relations as necessary. The relational database model captures inter-record relationships in a uniform manner: by means of common data values. Each relation has a column or a collection of columns whose values taken together uniquely determine rows in the relation. Such a column or column collection is termed a *primary key* (logical unique identifier) and is underlined in the figure. In this example, the primary key is the Name column if no two friends have the same name.

Fig. 3 presents an example database which contains four relations. This database includes information on publications, authors, reviewers, and recommendations of publications by reviewers. By relating the common values of the Title columns in both the AUTHORS and RECOMMENDATIONS relations, it shows that the book "*The Metaphorical Brain 2*" is written by the author named Arbib and the book is recommended by two reviewers named Campbell and Oneil.

Fig. 3 also shows examples of three fundamental relational model constraints, which are termed *domain integrity, entity integrity,* and *referential integrity*. The domain integrity constraint specifies that the value of a column entry must come from the underlying domain of that column. The entity integrity constraint specifies the uniqueness of rows in a relation. The value(s) of column(s) of the primary key in a relation taken together uniquely determine the row in the relation. For example, the primary key of RECOMMENDATIONS is the combination of Title and Reviewer, which means that given the title of a publication and the name of a reviewer, there is at most a single value of Rating for that pair. Referential integrity constraints, indicated by dashed lines in Fig. 3, specify that values in some columns must occur as values in some other column. For example, the arrow from Title in RECOMMENDATIONS to Title in PUBLICATIONS means informally that for a recommendation to exist for a publication, that publication must exist in the PUBLICATIONS relation; more precisely, the title of the publication as indicated in RECOMMENDATIONS must also exist as the title of the publication in PUBLICATIONS.

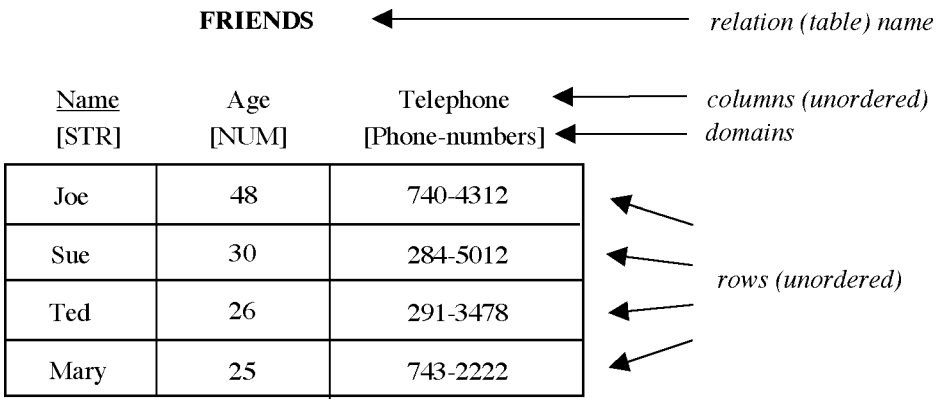

Figure 2 Tabular depiction of relations.

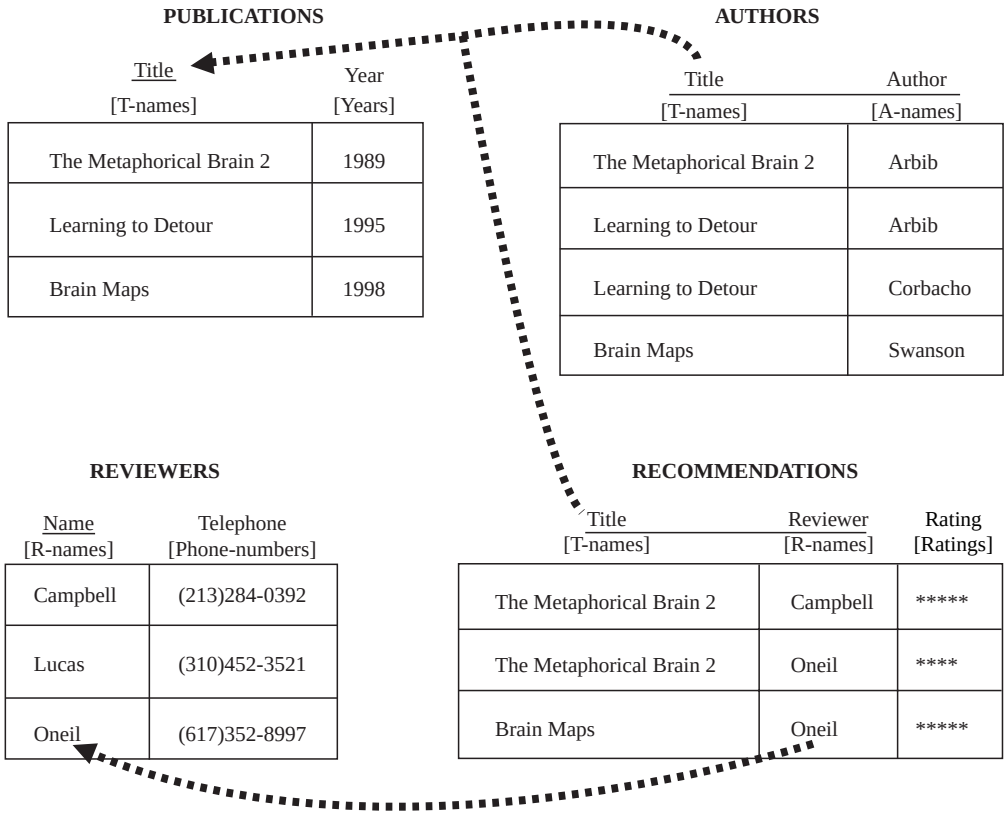

Figure 3 Example relational database.

1.2.4 SQL

The acronym SQL was originally an abbreviation for Structured Query Language. The SQL language consists of a set of primitives for defining, accessing, and managing data stored in a relational database (Date, 1990; Elmasri and Navathe, 1994). It is a high-level declarative language in that the user only specifies what the query result is and leaves the actual execution and optimization decisions to the DBMS. There are many commercial SQL implementations which comply to various SQL standards (SQL1, SQL2, or SQL3) and also provide their own extensions (Date and Hugh, 1997). The goal of this section is to present a brief and informal overview of some important primitives of standard SQL. The details of SQL can be found in Appendix B1 as well as in the references. The database in Fig. 3 will be used to illustrate using SQL primitives in defining tables, specifying constraints, updating the tables, and querying the tables. The SQL extensions for object-relational databases will be discussed later after the object-relational database model is introduced.

In Fig. 4, four CREATE TABLE statements are used to define the four tables in Fig. 3 with the specified names and specified named columns (with specified data types). Column names must be unique within their containing table. All four tables will initially be empty (i.e., contain no rows). There is a primary key specified for each of the four tables using the PRIMARY KEY keyword. The FOREIGN KEY keyword is used to indicate referential integrity constraints. Domain integrity, entity integrity, and referential integrity will be enforced by the underlying DBMS.

There are four basic SQL data manipulation operations: INSERT, DELETE, UPDATE, and SELECT. The first three operations are used to modify the contents of a table, and the SELECT statement is used to retrieve information from a database. The three INSERT statements in Fig. 5 illustrate how to add rows into the PUBLICATIONS table. In SQL syntax, entries for strings are quoted and entries for numbers are not quoted as shown in the examples. The UPDATE statement shows how to change the year of publication from 1993 to 1995 on the article entitled "Learning to Detour." The DELETE statement illustrates how to delete the row of publication "*Brain Maps*" from the PUBLICATIONS table. Note that the UPDATE and DELETE operations can be applied to several rows at once, although this is not shown in the examples. The same is also true in general for INSERT operations.

The SELECT operation has the general form "SELECT-FROM-WHERE" as illustrated in Fig 6. The SELECT clause indicates the columns to be retrieved and displayed. The FROM clause specifies the tables from which the results should be retrieved. The WHERE clause defines the criteria for the information

```
CREATE TABLE PUBLICATIONS
    ( Title VARCHAR(80),
      Year INTEGER,
    PRIMARY KEY (Title));

CREATE TABLE AUTHORS
    ( Title VARCHAR(80),
      Author CHAR(20),
    PRIMARY KEY (Title, Author),
      FOREIGN KEY (Title)
    REFERENCES PUBLICATIONS (Title));

CREATE TABLE REVIEWERS
    ( Name CHAR(20),
      Telephone CHAR(13),
    PRIMARY KEY (Name));

CREATE TABLE RECOMMENDATIONS
    ( Title VARCHAR(80),
      Reviewer CHAR(20),
      Rating CHAR(5),
    PRIMARY KEY (Title, Reviewer),
      FOREIGN KEY (Title)
    REFERENCES PUBLICATIONS(Title),
      FOREIGN KEY (Reviewer)
    REFERENCES REVIEWERS(Name));
```

Figure 4 Examples of SQL data definition.

```
INSERT INTO PUBLICATIONS VALUES
        ( 'The Metaphorical Brain 2', 1989);
INSERT INTO PUBLICATIONS VALUES
        ( 'Learning to Detour', 1993);
INSERT INTO PUBLICATIONS VALUES
        ( 'Brain Maps', 1998);

UPDATE PUBLICATIONS SET Year = 1995
        WHERE PUBLICATIONS.Title = 'Learning to Detour';

DELETE FROM PUBLICATIONS
        WHERE PUBLICATIONS.Title = 'Brain Maps';
```

Figure 5 Examples of SQL update.

that should be retrieved. Only the data that satisfy the predicate are returned as the result of the SELECT operation. If the WHERE clause is omitted, the result contains all rows of the tables in the FROM clause. The simple SELECT operation in Fig. 6 retrieves the title and rating of those publications that are recommended by Oneil. The operation returns two rows: *The Metaphorical Brain 2* with a four-star rating and *Brain Maps* with a five-star rating.

One of the important features of SQL is its support for the relational join operator. This operator makes it possible to retrieve data by "joining" two, three, or any number of tables, all in a single SELECT statement. Fig. 7 shows a query that retrieves the title, year of publication, and authors of each publication by joining PUBLICATIONS and AUTHORS on matching titles. The query produces a ternary (i.e., 3-ary) relation, with columns containing a title, the year of publication, and

```
SELECT Title, Year, Author
FROM PUBLICATIONS, AUTHORS
WHERE PUBLICATIONS.Title = AUTHORS.Title
```

Figure 7 An example of relational join operation.

an author of that publication. The join operation is illustrated in Fig. 8.

The SELECT operation can be nested. The result of a select statement can be used in another select statement. Fig. 9 shows a query that retrieves the title, year of publication, and authors of each publication that has been recommended by the reviewer named Oneil. This query is the same as the one in Fig. 7 except that only those publications recommended by Oneil are retrieved. The titles of publications recommended by Oneil are first selected in the inner select statement. A subset of the join on the PUBLICATIONS and AUTHORS relations on matching publication title is then selected corresponding to those publications recommended by Oneil in the inner select statement.

1.2.5 Object-Based Database Models

Object-based database models are placed in contradistinction to the relational database model. The term "object-based" refers to the class of database models that include those identified as "semantic" and "object-oriented" (Shipman, 1981; Hammer and McLeod, 1981; Fishman *et al.*, 1987; Hull and King, 1987). In what

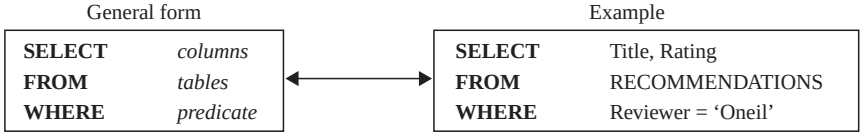

Figure 6 General form of SELECT operations.

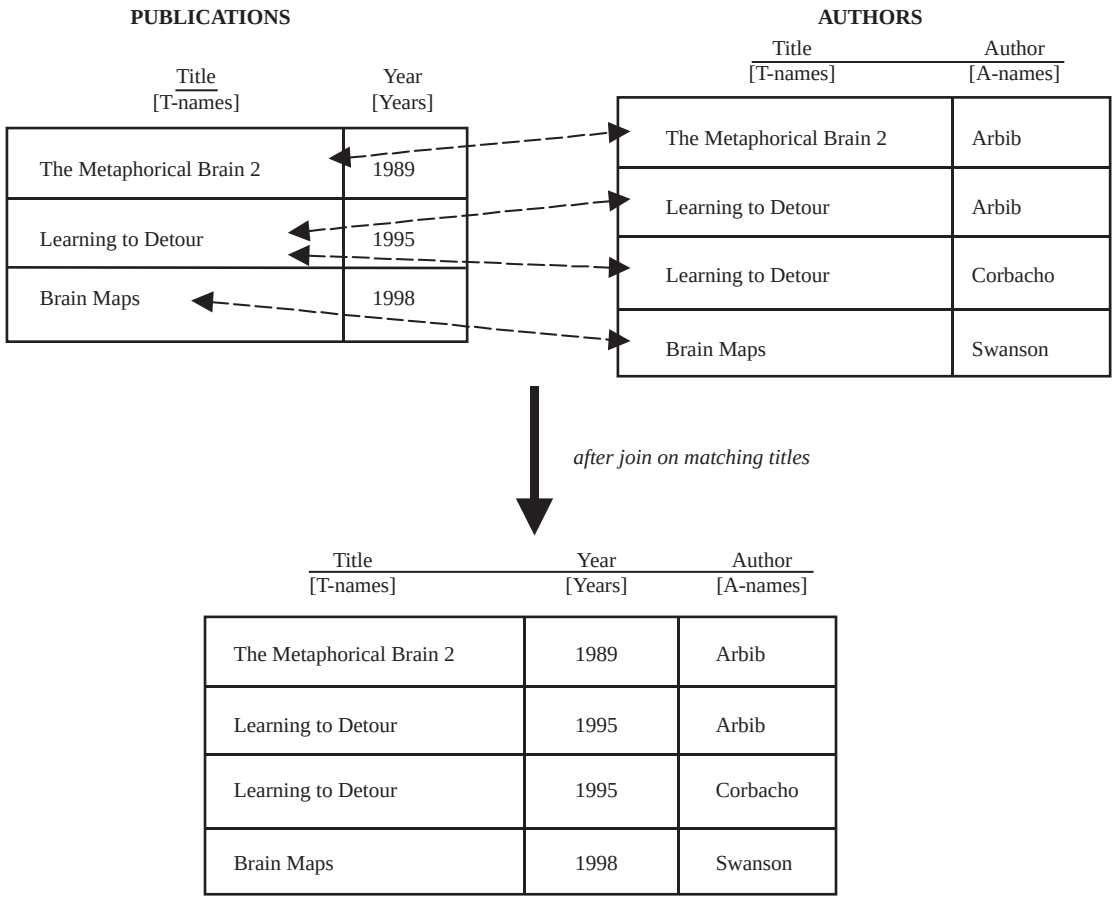

Figure 8 Illustration of the join operation on two tables.

```
SELECT Title, Year, Author
FROM PUBLICATIONS, AUTHORS
WHERE PUBLICATIONS.Title  = AUTHORS.Title
      And PUBLICATIONS.Title in
        (SELECT Title
          FROM RECOMMENDATIONS
          WHERE Reviewer = 'Oneil' );
```

Figure 9 An example of relational database query.

follows, first the principal characteristics of object-based database models are examined. The main components of the object-based model are then discussed and illustrated using an example database which is an extension of the database previously used in illustrating the relational database model.

The term "object-based" refers to the following characteristics, as exhibited by a database model and a database system that embodies that model (McLeod, 1991):

1. *Individual object identity:* Objects in a database can include not only primitive (atomic) data values, such as strings and numbers, but also abstract objects representing entities in the real world and intangible concepts. Relationships among and classifications of such objects can themselves be considered as abstract objects in the database. Graphical, image, and voice objects can also be accommodated. Such "abstract" objects can be directly represented and manipulated.

2. *Explicit semantic primitives:* Primitives are provided to support object classification, structuring, semantic integrity constraints, and derived data. These primitive abstraction mechanisms support such features as aggregation, classification, instantiation, and inheritance.

3. *Active objects:* Database objects can be active as well as passive in that they exhibit behavior. Various specific approaches to the modeling of object behavior can be adopted, such as an inter-object message-passing paradigm or abstract data type encapsulation. The important point is that behavioral abstraction is supported, and procedures to manipulate data are represented in the database.

4. *Object uniformity:* All (or nearly all) information in a database is described using the same object model. Thus, descriptive information about objects, referred to here as *meta-data*, is conceptually represented in the same way as specific "fact" objects. Meta-data are considered dynamic, and can be modified in a manner analogous to that used to alter "fact" objects.

To provide a more substantive analysis of the concepts underlying object-based database models in general, and the notions of object identity and explicit semantic primitives in particular, the following main components of an object-based model are considered:

1. Objects are abstract, complex, or atomic entities that correspond to things in the application environment being represented in the database and may be at various levels of abstraction and of various modalities (media).
2. Inter-object relationships describe associations among objects. Such relationships are modeled as attributes of objects (logical mapping from one object to another) and their inverses, as well as by association objects (objects that represent relationships as entities).
3. Object classifications group together objects that have some commonalities. The term "object type" often refers to such classification, and the term "class" to the set of objects classified according to the type. (The term type and class are sometimes informally used somewhat synonymously.) Relationships among object classifications specify inter-classification associations (e.g., the subclass/superclass inter-class relationship supports specialization/generalization). Object classes or types can themselves be considered objects at a higher level of abstraction (e.g., with attributes of their own).

Classes and Attributes

To examine the fundamentals of object-based database models, consider again the example application environment involving publications, authors, reviewers, and recommendations. Fig. 10 illustrates the concepts of classes and attributes. Classes are classifications of objects, and attributes are mappings from one object class to another. In this figure, classes are indicated as ovals. Example classes include abstract entity classifications, such as PUBLICATIONS, AUTHORS, REVIEWERS, and RECOMMENDATIONS, as well as atomic value classes, such as Phone-numbers, T-names (title names), A-names (author names), Ratings, etc.

In Fig. 10, attributes are indicated by labeled directed arrows from the described class to the describing class (value class). Attributes are labeled with a name and are indicated as single valued (1) or multi-valued (m); attribute inverses are specified by a line connecting two small circles (or by two tracking small circles). An attribute A from class C1 to a class C2 means that an object classified as being of class C1 may be related to object(s) of class C2 via A; the inverse of A, which is always logically present, may be explicitly indicated and named. For example, class PUBLICATIONS has an attribute called has-authors that is multi-valued; for a given publication object, its authors are indicated by the values of this attribute, which relates the publication to zero or more objects of type AUTHORS. The inverse of this attribute

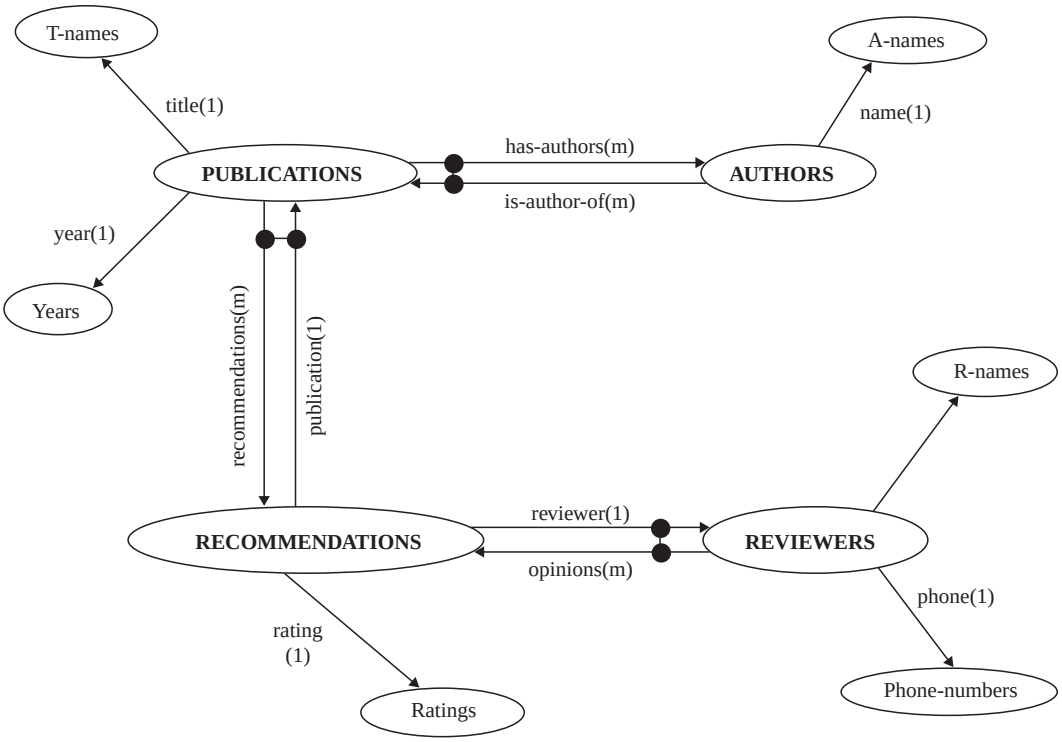

Figure 10 Classes and attributes.

is called is-author-of of AUTHORS, which is also multi-valued. Another example is the attribute year of PUBLICATIONS, which is single valued; its inverse is not explicitly present here but can be referenced as "the inverse of year of PUBLICATIONS" (the publications published in a given year), which is of course an attribute of class Years.

One-to-one, one-to-many, many-to-one, and many-to-many binary relationships between object classes can be expressed by attribute and attribute inverse pairs. The many-to-many relationships between classes PUBLICATIONS and AUTHORS are represented by the multi-valued attribute has-authors of PUBLICATIONS and its multi-valued inverse, is-author-of of AUTHORS. An example of a one-to-many relationship is indicated between the PUBLICATIONS and RECOMMENDATIONS classes by the single-valued attribute publication of RECOMMENDATIONS and its multi-valued inverse attribute recommendations of PUBLICATIONS; this means informally that an object of class RECOMMENDATIONS represents the evaluation of a single publication, and that a given publication may have many evaluations. An example of a one-to-one relationship, although not indicated as such in Fig. 8, might be between PUBLICATIONS and T-names; here the attribute called title of PUBLICATIONS is single valued, and its inverse (from T-names to PUBLICATIONS; e.g., called is-title-of) would also be single valued.

Fig. 11 illustrates the RECOMMENDATIONS class in more detail. Here RECOMMENDATIONS has three attributes (publication, reviewer, and rating), each of which is single valued. Objects of class RECOMMENDATIONS represent abstract entities that correspond to recommendations of publications by reviewers with a given rating; they in effect model a ternary relationship. It is also possible to consider a derived attribute called

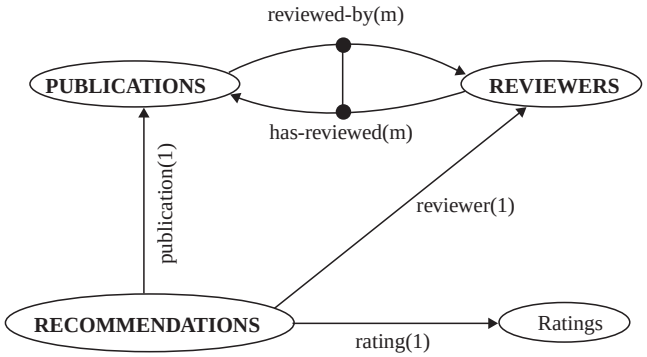

Figure 11 Association objects.

reviewed-by of PUBLICATIONS and its inverse (has-reviewed of REVIEWERS), both of which can be derived from information carried by the values of attributes of RECOMMENDATIONS. This is an example of derived data in general and derived attributes in particular; note that attribute and attribute inverse pairs are in a sense also derived or, more precisely, logically redundant information. A database system that supports an object-based database model must of course support this redundancy and maintain consistency in its presence.

Subtype and Inheritance

The concept of specialization (and its inverse, generalization) is an important kind of inter-class relationship, which is supported by subclass/superclass construct. Fig. 12 illustrates this relationship between a class and its subclass(es), using a large arrow from a class to a subclass. In the example, the class PUBLICATIONS has two subclasses, namely BOOKS and PAPERS. PAPERS in turn has two subclasses, namely JOURNAL PAPERS and CONFERENCE PAPERS. Attributes are inherited by a subclass from its superclass, and the subclass may

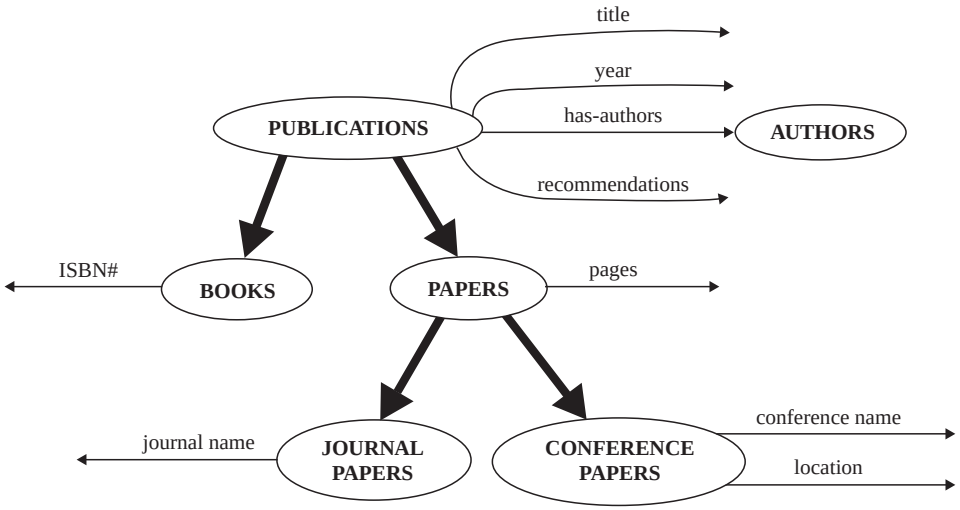

Figure 12 Subclasses, superclasses, and attribute inheritance.

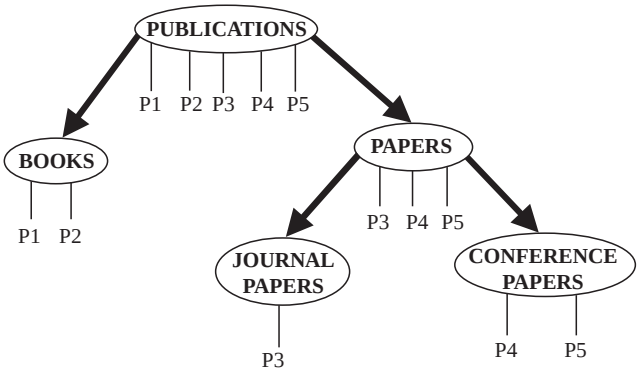

Figure 13 Subclasses and instances.

have additional attributes as well. Thus, BOOKS has its own attribute ISBN, as well as the attributes called title, year, has-authors, and recommendations, which are inherited from PUBLICATIONS. (For simplicity in the figure, the value classes of some attributes are omitted.)

The objects classified by a class may be termed the *instances* of that class. The set of instances of a subclass must be a subset of the set of instances of its superclass. Fig. 13 shows some instances of class PUBLICATIONS, indicated as P1, P2, P3, P4, and P5. Each instance of CONFERENCE PAPERS is an instance of three classes; for example, P5 can be viewed as a conference paper, a paper, and a publication. The subset constraint involving instances of a subclass must, of course, be enforced by the database system.

It is also possible for a class to have multiple superclasses. In this case, the specialization structure among classes is not necessarily a forest of trees but rather a directed acyclic graph. For example, suppose that FRIENDS and COLLEAGUES are defined as subclasses of PERSONS; the class FRIENDS AT WORK might then be defined as a subclass of both FRIENDS and COLLEAGUES. In this example, the instance subset constraint implies that the instances of FRIENDS AT WORK must be a subset of the instances of FRIENDS as well as a subset of the instances of COLLEAGUES. When multiple superclasses are permitted, some rules must be present in the database model to accommodate the problem of multiple inheritance (namely, the inheritance of attributes from more than one superclass).

1.2.6 Object-Relational Database Model

The object-relational database model is based on the relational database model with extensions of several key ideas from object-based database models (Rowe and Stonebraker, 1987; Stonebraker *et al.*, 1999). The basic structure of the object-relational model is still a relation, but the domains of a column in a relation are extended to include abstract data types in addition to atomic data

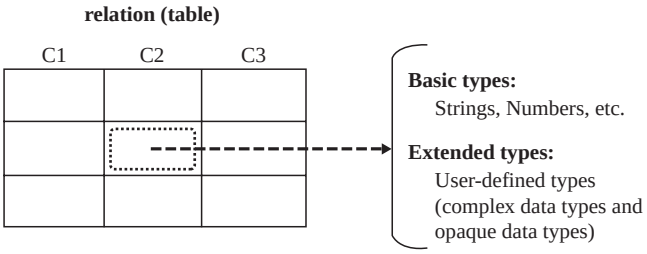

Figure 14 Extending domain types in a relation.

types like strings and numbers (Fig. 14). Subclasses (subrelations) and inheritance are added to incorporate specialization/generalization inter-class (inter-relation) relationships. With the ability for users to define abstract data types with subclasses and inheritance, the object-relational database model has more capability for modeling complex data than the traditional relational database model. Unlike most object-based database models, the object-relational model maintains the top-level structure as relations and make it possible to use the standard SQL with extensions to define, access, and manipulate its databases. In what follows, extended data types and subclasses/inheritance of the object-relational database model will be further illustrated using Informix universal server DBMS (Informix Inc., 1997a,b).

The object-relational database model is an extension to the relational database model. By ignoring the additional types and constructs provided by the object-relational database model, one could just use an object-relational database system as a relational database system. In the relational database model, the domains of a column in a relation (table) are limited to basic atomic data types such as strings and numbers, etc. The object-relational database model eases the limitation to include user-defined extended types as the domains of columns in a relation. By doing so, it enables users to capture more semantics of complex application environments. There are two main categories of extended types. The first category is made up of the *complex data types* including row types and collections. A row type is a data type consisting of a record of values, which is analogous to a C structure or C++ class. Row types could be specifically named or created automatically as an anonymous (unnamed) row type during table creation using CREATE TABLE statements. In Fig. 15, the left side creates the PUBLICATIONS table with an unnamed row type. The right side creates a named row type PUBLICATIONS_T and uses it create the table PUBLICATIONS. In addition to using a named row type to create tables, you can also use it to define individual columns of a table. The collection types include set, list, and multiset types that can hold multiple values.

The second category of extended types is the *opaque type*. An opaque data type is a user-defined type to be used like an atomic data type. Opaque types are totally encapsulated. The database server knows nothing about

the representation and implementation of an opaque type. All access to an opaque type is through the functions written by users. They are accessed through the functions defined on them. Once an opaque type is defined and implemented, it could be used just like built-in strings and numbers. For example, a complex number is not directly supported by most database system, but it can be implemented as an opaque type with input, output, and arithmetic functions. After complex number type is defined and implemented, operations on complex numbers can thus be supported in the database system. The object-relational database model also supports subclasses and inheritance. Subclasses can be applied to tables, row types, or opaque types. Both the data and functions defined on the superclasses are inherited by their subclasses; however, there is one restriction on creating sub-tables using named row types. If named row types are used in creating tables, the type hierarchy of the named rows must match the type hierarchy of the corresponding tables. Fig. 16 shows how to create PUBLICATIONS and BOOKS tables (see Fig. 12) using named row types. Note that the named row type BOOKS_T is a subtype of the type PUBLICATIONS_T. The corresponding BOOKS table is also a sub-table of the PUBLICATIONS table.

As a further extension to the basic relational database model, Informix as well as many other object-relational database management systems also add the capability of allowing multi-valued attributes. Thus, a many-to-many relationship can be expressed as set-valued attribute. Such set-valued attributes can be useful but introduce the problem that one must select which "side" to represent the relationship; using a separate table for many-to-many relationships avoids this problem by symmetrically representing the relationship as a pair of references to the tuples/objects involved in it. For example, we can represent the many-to-many relationship between authors and publications as a set-valued attribute for tuples in the AUTHORS relation or a set-valued attribute in the PUBLICATIONS relation. Alternatively, we can use a new table, say AUTHORING, which contains single-valued attributes that refer to the AUTHORS and PUBLICATIONS tuples; in the latter approach, there will be a tuple in AUTHORING for each binary pairing of an author and a publication; if there are attributes of this relationship, they can be placed in the AUTHORING relation.

1.2.7 An Overview of Federated Database Systems

With the rapid growth of computer communication networks over the last decade, a vast amount of diverse information has become available on the networks. Users often find it desirable to share and exchange information that resides either within the same organization or beyond organizational boundaries. As we shall explain in detail in Chapter 5.1, a federated database system is a collection of autonomous, heterogeneous, and cooperating component database systems (Heimbigner and McLeod, 1985; Sheth and Larson, 1990). Component database systems unite into a loosely coupled federation in order to achieve a common goal: to share and exchange information by cooperating with other components in the federation without compromising the autonomy of each component database system. Such environments are increasingly common in various application domains, including office information systems, computer-integrated manufacturing, scientific databases, etc.

The issues related to sharing and exchanging of information among federated database systems can be considered in three different phases (Kahng and McLeod 1996; Aslan 1998; Fang *et al.*, 1996). First, the user of a component database in the federation needs to find out what information can be shared and identify the location

```
CREATE TABLE PUBLICATIONS
    ( Title VARCHAR(80),
      Year INTEGER,
      PRIMARY KEY ( Title ) );
```

```
CREATE ROW TYPE PUBLICATIONS_T
    ( Title VARCHAR(80),
      Year INTEGER,
      PRIMARY KEY ( Title ) );

CREATE TABLE PUBLICATIONS
    OF TYPE PUBLICATIONS_T;
```

Figure 15 Creating tables using named and unnamed row types.

```
CREATE ROW TYPE PUBLICATIONS_T
    ( Title VARCHAR(80),
      Year INTEGER,
      PRIMARY KEY ( Title ) );

CREATE ROW TYPE BOOKS_T
    ( ISBN CHAR(13) )
    UNDER PUBLICATIONS_T;
```

```
CREATE TABLE PUBLICATIONS
    OF TYPE PUBLICATIONS_T;

CREATE TABLE BOOKS
    UNDER PUBLICATIONS;
```

Figure 16 Named row type hierarchy must match corresponding table hierarchy.

and content of relevant information units which reside in other component databases in the federation. This is categorized as the *information discovery phase*. Once the information units of interest have been identified and located, the next phase is to resolve the differences between the information units to be imported and the information units contained in the user's local component database. This is the *semantic heterogeneity resolution phase*. After the various kinds of heterogeneity have been resolved, the next phase is to actually import the information units into the user's local environment so they can be used in an efficient and integrated manner. This is called the *system-level interconnection phase*. In Chapter 5.1, we will discuss federated database systems in detail and introduce the approaches to the three layers of issues related to federated database systems.

References

Aslan, G. (1998). Semantic Heterogeneity Resolution in Federated Databases by Meta-Data Implantation and Stepwise Evolution, Ph.D. dissertation, University of Southern California, Los Angeles.

Chen, P. (1976). The entity-relationship model: toward a unified view of data. *ACM Trans Database Syst.* **1(1)**, 9–36.

Codd, E. F. (1970). A relational model of data for large shared data banks. *CACM*. **13(6)**, 37–387.

Date, C. J. (1990). *An Introduction to Database Systems.* Vol. 1, 5th ed. Addison-Wesley, Reading, MA.

Date, C. J., and Hugh, D. (1997). *A Guide to the SQL Standard.* 4th ed. Addison-Wesley, Reading, MA.

Elmasri, R., and Navathe, S. (1994). *Fundamentals of Database Systems.* 2nd ed. Benjamin/Cummings, Menlo Park, CA.

Fang, D., Ghandeharizadel, S. and McLeod, D. (1996). An experimental object-based sharing system for networked databases. *VLDB J.* **5**, 151–165.

Fishman, D., Beech, D., Cate, H. *et al.* (1987). Iris: an object-oriented database management system. *ACM Trans. Office Inf. Syst.* **5(1)**, 48–69.

Hammer, M., and McLeod, D. (1981). Database description with SDM: a semantic database model. *ACM Trans. Database System* **6(3)**, 351–386.

Heimbigner, D., and McLeod, D. (1985). A federated architecture for information management. *ACM Trans. Office Inf. Syst.* **3(3)**, 253–278.

Hull, R., and King, R. (1987). Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys* **19(3)**, 201–260.

Informix, Inc. (1997a). *Informix Guide to SQL: Tutorial Version 9.1.* Informix Press, Upper Saddle River, NJ.

Informix, Inc. (1997b). An introduction to Informix universal server extended features. Informix Press, Upper Saddle River, NJ.

Kahng, J., and McLeod, D. (1996). Dynamic classificational ontologies for discovery in cooperative federated databases. In *Proceedings of the First International Conference on Cooperative Information Systems.* Brussels, Belgium, pp. 26–36.

McLeod, D. (1991). Perspective on object databases. *Inf. Software Technol.* **33(1)**, 13–21.

Rowe, L., and Stonebraker, R. (1987). The POSTGRES data model. In Proceedings of the International Conference on Very Large Databases, September 1–4, 1987, Brighton, England, pp. 83–96.

Sheth, A., and Larson, J. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys* **22(3)**, 183–236.

Shipman, D. (1981). The functional data model and the data language daplex. *ACM Trans. Database Syst.* **2(3)**, 140–173.

Stonebraker, M., Brown, P. and Moore, D. (1999). *Object-Relational DBMSs: Tracking the Next Great Wave*. Morgan Kaufmann, San Mateo, CA.