type DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total.

After specifying the above six relationship types, we remove from the entity types in Figure 7.8 all attributes that have been refined into relationships. These include Manager and Manager_start_date from DEPARTMENT; Controlling_department from PROJECT; Department, Supervisor, and Works_on from EMPLOYEE; and Employee from DEPENDENT. It is important to have the least possible redundancy when we design the conceptual schema of a database. If some redundancy is desired at the storage level or at the user view level, it can be introduced later, as discussed in Section 1.6.1.

7.7 ER Diagrams, Naming Conventions, and Design Issues

7.7.1 Summary of Notation for ER Diagrams

Figures 7.9 through 7.13 illustrate examples of the participation of entity types in relationship types by displaying their sets or extensions—the individual entity instances in an entity set and the individual relationship instances in a relationship set. In ER diagrams the emphasis is on representing the schemas rather than the instances. This is more useful in database design because a database schema changes rarely, whereas the contents of the entity sets change frequently. In addition, the schema is obviously easier to display, because it is much smaller.

Figure 7.2 displays the COMPANY **ER database schema** as an **ER diagram**. We now review the full ER diagram notation. Entity types such as EMPLOYEE, DEPARTMENT, and PROJECT are shown in rectangular boxes. Relationship types such as WORKS_FOR, MANAGES, CONTROLS, and WORKS_ON are shown in diamond-shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached by a straight line to its entity type or relationship type. Component attributes of a composite attribute are attached to the oval representing the composite attribute, as illustrated by the Name attribute of EMPLOYEE. Multivalued attributes are shown in double ovals, as illustrated by the Locations attribute of DEPARTMENT. Key attributes have their names underlined. Derived attributes are shown in dotted ovals, as illustrated by the Number_of_employees attribute of DEPARTMENT.

Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds, as illustrated by the DEPENDENT entity type and the DEPENDENTS_OF identifying relationship type. The partial key of the weak entity type is underlined with a dotted line.

In Figure 7.2 the cardinality ratio of each *binary* relationship type is specified by attaching a 1, M, or N on each participating edge. The cardinality ratio of DEPARTMENT:EMPLOYEE in MANAGES is 1:1, whereas it is 1:N for DEPARTMENT: EMPLOYEE in WORKS_FOR, and M:N for WORKS_ON. The participation constraint is specified by a single line for partial participation and by double lines for total participation (existence dependency).

In Figure 7.2 we show the role names for the SUPERVISION relationship type because the same EMPLOYEE entity type plays two distinct roles in that relationship. Notice that the cardinality ratio is 1:N from supervisor to supervisee because each employee in the role of supervisee has at most one direct supervisor, whereas an employee in the role of supervisor can supervise zero or more employees.

Figure 7.14 summarizes the conventions for ER diagrams. It is important to note that there are many other alternative diagrammatic notations (see Section 7.7.4 and Appendix A).

7.7.2 Proper Naming of Schema Constructs

When designing a database schema, the choice of names for entity types, attributes, relationship types, and (particularly) roles is not always straightforward. One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema. We choose to use *singular names* for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type. In our ER diagrams, we will use the convention that entity type and relationship type names are uppercase letters, attribute names have their initial letter capitalized, and role names are lowercase letters. We have used this convention in Figure 7.2.

As a general practice, given a narrative description of the database requirements, the *nouns* appearing in the narrative tend to give rise to entity type names, and the *verbs* tend to indicate names of relationship types. Attribute names generally arise from additional nouns that describe the nouns corresponding to entity types.

Another naming consideration involves choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom. We have generally followed this guideline in Figure 7.2. To explain this naming convention further, we have one exception to the convention in Figure 7.2—the DEPENDENTS_OF relationship type, which reads from bottom to top. When we describe this relationship, we can say that the DEPENDENT entities (bottom entity type) are DEPENDENTS_OF (relationship name) an EMPLOYEE (top entity type). To change this to read from top to bottom, we could rename the relationship type to HAS_DEPENDENTS, which would then read as follows: An EMPLOYEE entity (top entity type) HAS_DEPENDENTS (relationship name) of type DEPENDENT (bottom entity type). Notice that this issue arises because each binary relationship can be described starting from either of the two participating entity types, as discussed in the beginning of Section 7.4.

7.7.3 Design Choices for ER Conceptual Design

It is occasionally difficult to decide whether a particular concept in the miniworld should be modeled as an entity type, an attribute, or a relationship type. In this

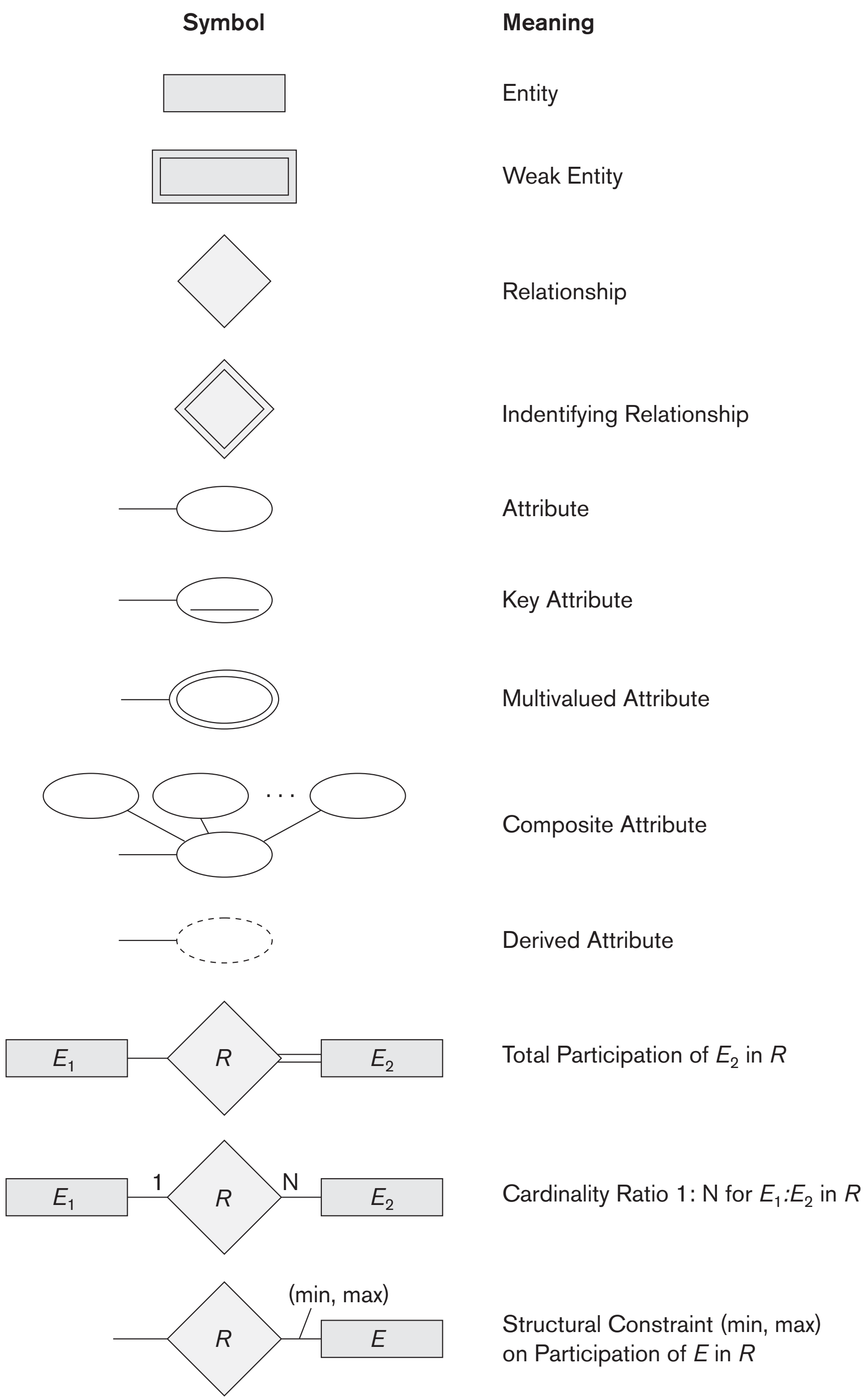


Figure 7.14
Summary of the notation for ER diagrams.

section, we give some brief guidelines as to which construct should be chosen in particular situations.

In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:

- A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the case that a pair of such attributes that are inverses of one another are refined into a binary relationship. We discussed this type of refinement in detail in Section 7.6. It is important to note that in our notation, once an attribute is replaced by a relationship, the attribute itself should be removed from the entity type to avoid duplication and redundancy.
- Similarly, an attribute that exists in several entity types may be elevated or promoted to an independent entity type. For example, suppose that several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, each has an attribute Department in the initial design; the designer may then choose to create an entity type DEPARTMENT with a single attribute Dept_name and relate it to the three entity types (STUDENT, INSTRUCTOR, and COURSE) via appropriate relationships. Other attributes/relationships of DEPARTMENT may be discovered later.
- An inverse refinement to the previous case may be applied—for example, if an entity type DEPARTMENT exists in the initial design with a single attribute Dept_name and is related to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.
- Section 7.9 discusses choices concerning the degree of a relationship. In Chapter 8, we discuss other refinements concerning specialization/generalization. Chapter 10 discusses additional top-down and bottom-up refinements that are common in large-scale conceptual schema design.

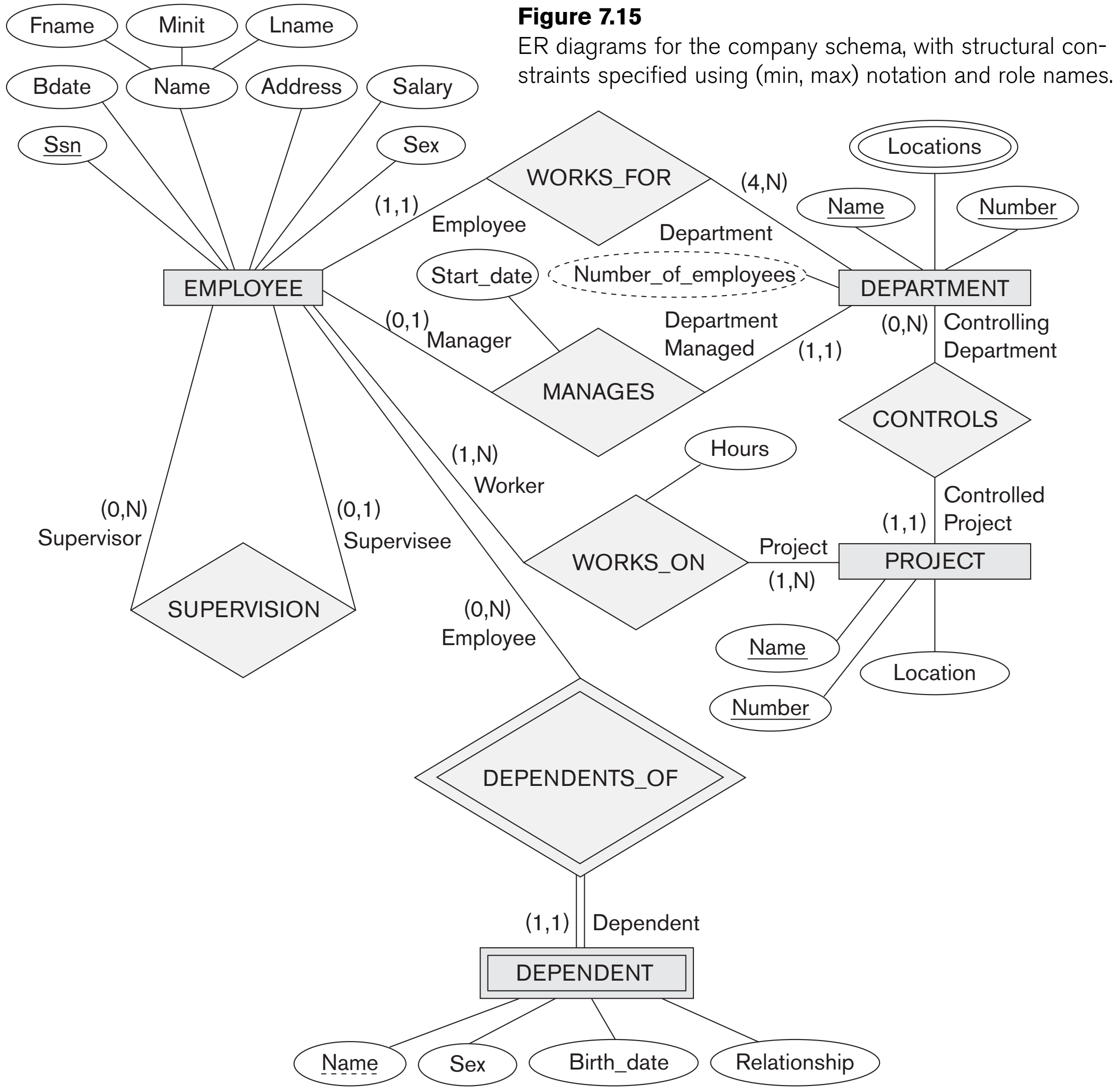
7.7.4 Alternative Notations for ER Diagrams

There are many alternative diagrammatic notations for displaying ER diagrams. Appendix A gives some of the more popular notations. In Section 7.8, we introduce the Unified Modeling Language (UML) notation for class diagrams, which has been proposed as a standard for conceptual object modeling.

In this section, we describe one alternative ER notation for specifying structural constraints on relationships, which replaces the cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints. This notation involves associating a pair of integer numbers (min, max) with each *participation* of an entity type E in a relationship type R, where $0 \le \text{min} \le \text{max}$ and $\text{max} \ge 1$. The numbers mean that for each entity e in E, e must participate in at least min and at most max relationship instances in *R at any point in time*. In this method, min = 0 implies partial participation, whereas min > 0 implies total participation.

Figure 7.15 displays the COMPANY database schema using the (min, max) notation.[14] Usually, one uses either the cardinality ratio/single-line/double-line notation *or* the (min, max) notation. The (min, max)

Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.



[14]In some notations, particularly those used in object modeling methodologies such as UML, the (min, max) is placed on the *opposite sides* to the ones we have shown. For example, for the WORKS_FOR relationship in Figure 7.15, the (1,1) would be on the DEPARTMENT side, and the (4,N) would be on the EMPLOYEE side. Here we used the original notation from Abrial (1974).

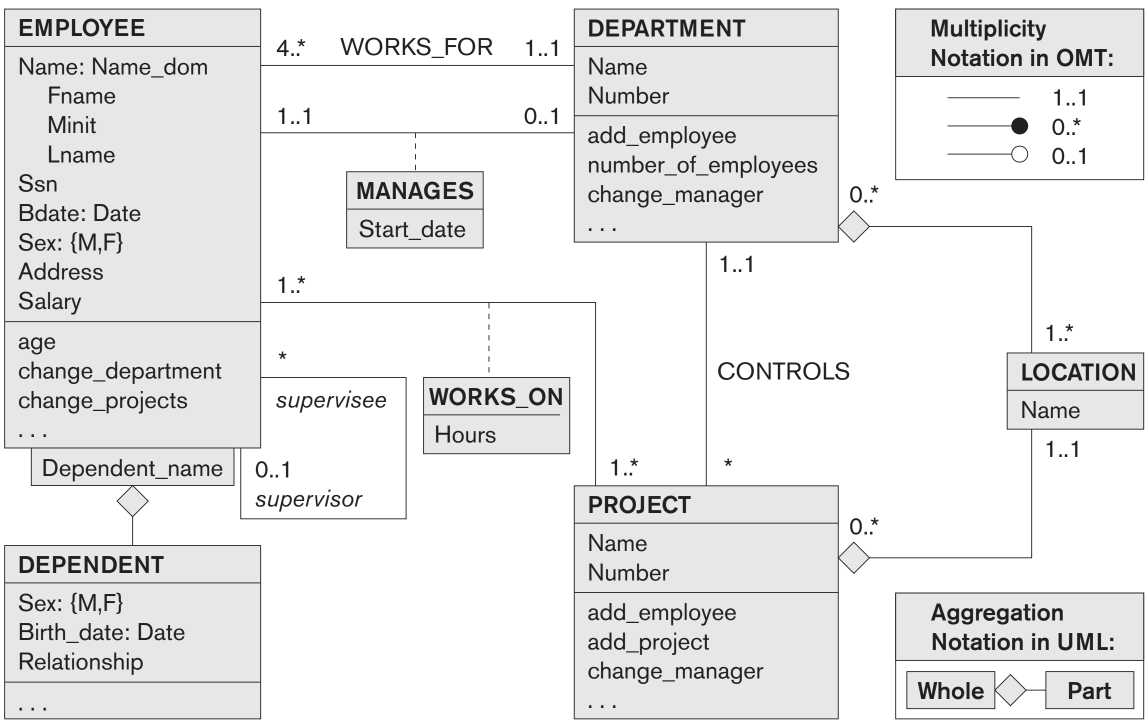
notation is more precise, and we can use it to specify some structural constraints for relationship types of *higher degree*. However, it is not sufficient for specifying some key constraints on higher-degree relationships, as discussed in Section 7.9.

Figure 7.15 also displays all the role names for the COMPANY database schema.

7.8 Example of Other Notation: UML Class Diagrams

The UML methodology is being used extensively in software design and has many types of diagrams for various software design purposes. We only briefly present the basics of **UML class diagrams** here, and compare them with ER diagrams. In some ways, class diagrams can be considered as an alternative notation to ER diagrams. Additional UML notation and concepts are presented in Section 8.6, and in Chapter 10. Figure 7.16 shows how the COMPANY ER database schema in Figure 7.15 can be displayed using UML class diagram notation. The *entity types* in Figure 7.15 are modeled as *classes* in Figure 7.16. An *entity* in ER corresponds to an *object* in UML.

Figure 7.16
The COMPANY conceptual schema in UML class diagram notation.



Review Questions

7.1. Discuss the role of a high-level data model in the database design process.

7.2. List the various cases where use of a NULL value would be appropriate.

7.3. Define the following terms: *entity, attribute, attribute value, relationship instance, composite attribute, multivalued attribute, derived attribute, complex attribute, key attribute,* and *value set* (*domain*).

7.4. What is an entity type? What is an entity set? Explain the differences among an entity, an entity type, and an entity set.

7.5. Explain the difference between an attribute and a value set.

7.6. What is a relationship type? Explain the differences among a relationship instance, a relationship type, and a relationship set.

7.7. What is a participation role? When is it necessary to use role names in the description of relationship types?

7.8. Describe the two alternatives for specifying structural constraints on relationship types. What are the advantages and disadvantages of each?

7.9. Under what conditions can an attribute of a binary relationship type be migrated to become an attribute of one of the participating entity types?

7.10. When we think of relationships as attributes, what are the value sets of these attributes? What class of data models is based on this concept?

7.11. What is meant by a recursive relationship type? Give some examples of recursive relationship types.

7.12. When is the concept of a weak entity used in data modeling? Define the terms *owner entity type, weak entity type, identifying relationship type,* and *partial key*.

7.13. Can an identifying relationship of a weak entity type be of a degree greater than two? Give examples to illustrate your answer.

7.14. Discuss the conventions for displaying an ER schema as an ER diagram.

7.15. Discuss the naming conventions used for ER schema diagrams.

Exercises

7.16. Consider the following set of requirements for a UNIVERSITY database that is used to keep track of students' transcripts. This is similar but not identical to the database shown in Figure 1.2:

a. The university keeps track of each student's name, student number, Social Security number, current address and phone number, permanent address and phone number, birth date, sex, class (freshman, sophomore, ..., graduate), major department, minor department (if any), and degree program

(B.A., B.S., ..., Ph.D.). Some user applications need to refer to the city, state, and ZIP Code of the student's permanent address and to the student's last name. Both Social Security number and student number have unique values for each student.

b. Each department is described by a name, department code, office number, office phone number, and college. Both name and code have unique values for each department.

c. Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of the course number is unique for each course.

d. Each section has an instructor, semester, year, course, and section number. The section number distinguishes sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the number of sections taught during each semester.

e. A grade report has a student, section, letter grade, and numeric grade (0, 1, 2, 3, or 4).

Design an ER schema for this application, and draw an ER diagram for the schema. Specify key attributes of each entity type, and structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.

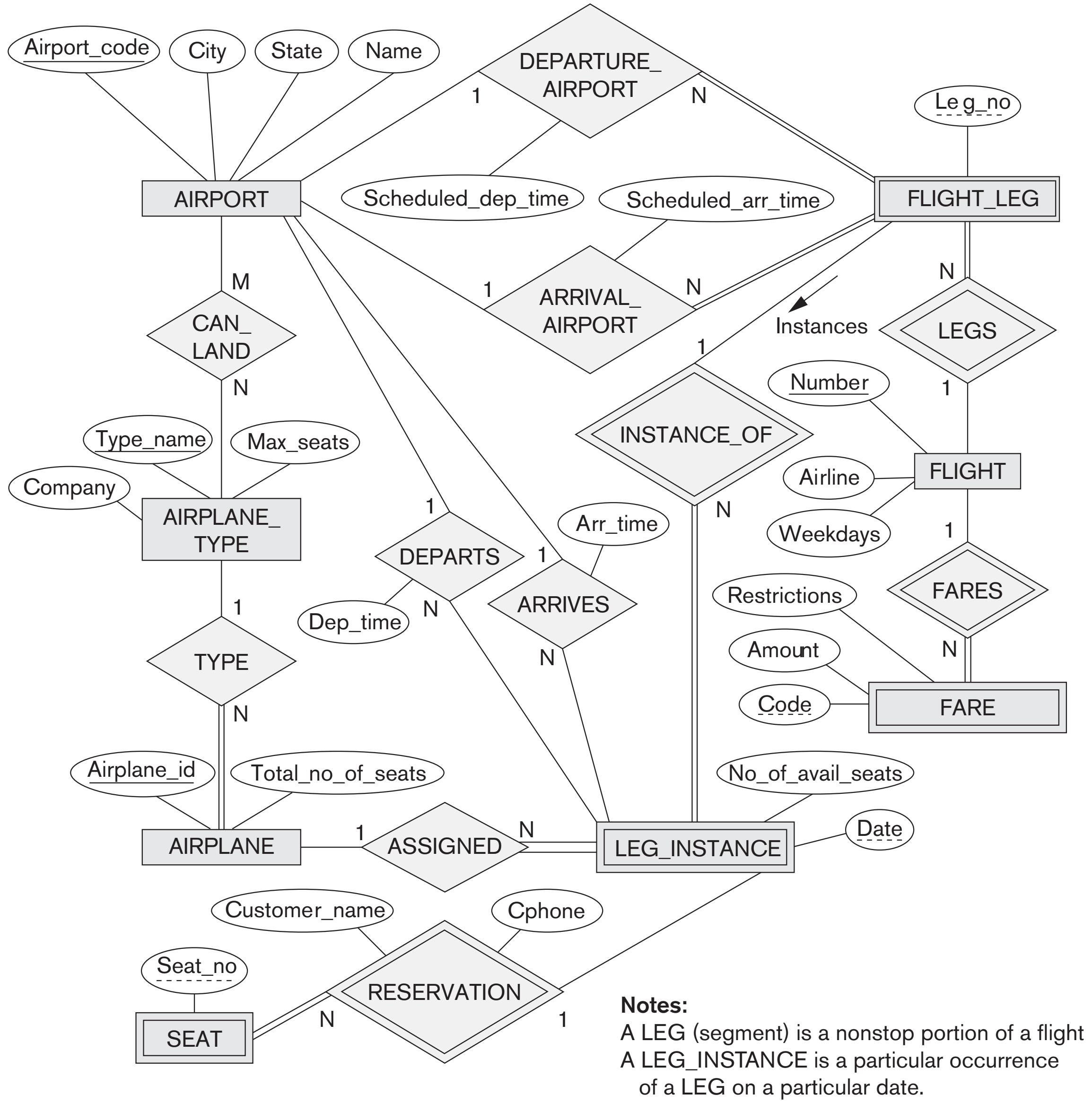
7.17. Composite and multivalued attributes can be nested to any number of levels. Suppose we want to design an attribute for a STUDENT entity type to keep track of previous college education. Such an attribute will have one entry for each college previously attended, and each such entry will be composed of college name, start and end dates, degree entries (degrees awarded at that college, if any), and transcript entries (courses completed at that college, if any). Each degree entry contains the degree name and the month and year the degree was awarded, and each transcript entry contains a course name, semester, year, and grade. Design an attribute to hold this information. Use the conventions in Figure 7.5.

7.18. Show an alternative design for the attribute described in Exercise 7.17 that uses only entity types (including weak entity types, if needed) and relationship types.

7.19. Consider the ER diagram in Figure 7.20, which shows a simplified schema for an airline reservations system. Extract from the ER diagram the requirements and constraints that produced this schema. Try to be as precise as possible in your requirements and constraints specification.

7.20. In Chapters 1 and 2, we discussed the database environment and database users. We can consider many entity types to describe such an environment, such as DBMS, stored database, DBA, and catalog/data dictionary. Try to specify all the entity types that can fully describe a database system and its environment; then specify the relationship types among them, and draw an ER diagram to describe such a general database environment.

Figure 7.20
An ER diagram for an AIRLINE database schema.



7.21. Design an ER schema for keeping track of information about votes taken in the U.S. House of Representatives during the current two-year congressional session. The database needs to keep track of each U.S. STATE's Name (e.g., 'Texas', 'New York', 'California') and include the Region of the state (whose domain is {'Northeast', 'Midwest', 'Southeast', 'Southwest', 'West'}). Each

CONGRESS_PERSON in the House of Representatives is described by his or her Name, plus the District represented, the Start_date when the congressperson was first elected, and the political Party to which he or she belongs (whose domain is {'Republican', 'Democrat', 'Independent', 'Other'}). The database keeps track of each BILL (i.e., proposed law), including the Bill_name, the Date_of_vote on the bill, whether the bill Passed_or_failed (whose domain is {'Yes', 'No'}), and the Sponsor (the congressperson(s) who sponsored—that is, proposed—the bill). The database also keeps track of how each congressperson voted on each bill (domain of Vote attribute is {'Yes', 'No', 'Abstain', 'Absent'}). Draw an ER schema diagram for this application. State clearly any assumptions you make.

7.22. A database is being constructed to keep track of the teams and games of a sports league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game for each team, the positions they played in that game, and the result of the game. Design an ER schema diagram for this application, stating any assumptions you make. Choose your favorite sport (e.g., soccer, baseball, football).

7.23. Consider the ER diagram shown in Figure 7.21 for part of a BANK database. Each bank can have multiple branches, and each branch can have multiple accounts and loans.

a. List the strong (nonweak) entity types in the ER diagram.

b. Is there a weak entity type? If so, give its name, partial key, and identifying relationship.

c. What constraints do the partial key and the identifying relationship of the weak entity type specify in this diagram?

d. List the names of all relationship types, and specify the (min, max) constraint on each participation of an entity type in a relationship type. Justify your choices.

e. List concisely the user requirements that led to this ER schema design.

f. Suppose that every customer must have at least one account but is restricted to at most two loans at a time, and that a bank branch cannot have more than 1,000 loans. How does this show up on the (min, max) constraints?

7.24. Consider the ER diagram in Figure 7.22. Assume that an employee may work in up to two departments or may not be assigned to any department. Assume that each department must have one and may have up to three phone numbers. Supply (min, max) constraints on this diagram. *State clearly any additional assumptions you make.* Under what conditions would the relationship HAS_PHONE be redundant in this example?

7.25. Consider the ER diagram in Figure 7.23. Assume that a course may or may not use a textbook, but that a text by definition is a book that is used in some course. A course may not use more than five books. Instructors teach from

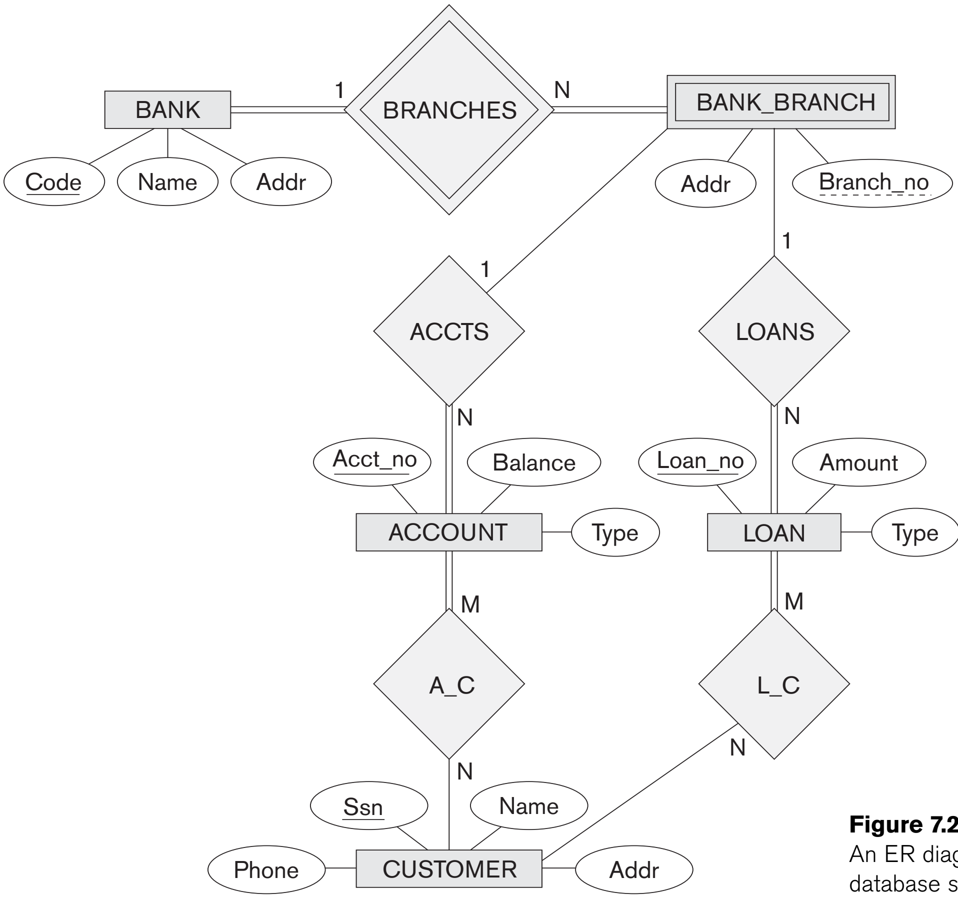


Figure 7.21
An ER diagram for a BANK database schema.

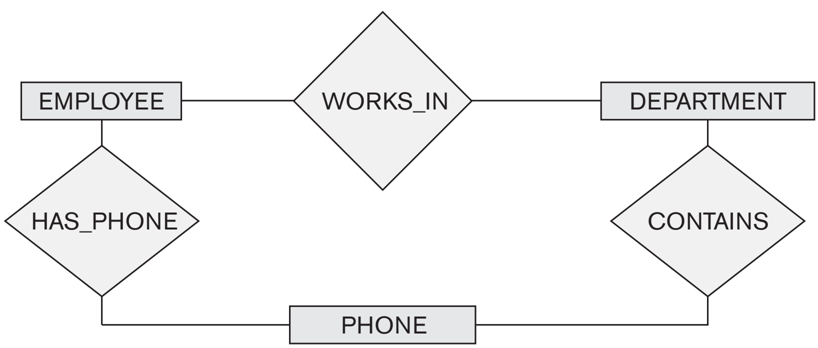


Figure 7.22
Part of an ER diagram for a COMPANY database.

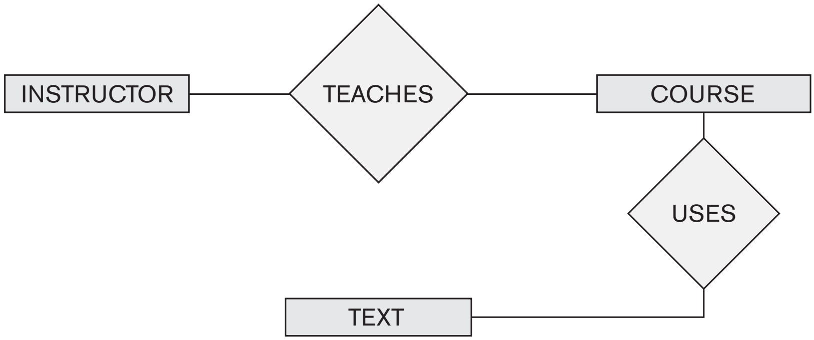


Figure 7.23
Part of an ER diagram for a COURSES database.

two to four courses. Supply (min, max) constraints on this diagram. *State clearly any additional assumptions you make.* If we add the relationship ADOPTS, to indicate the textbook(s) that an instructor uses for a course, should it be a binary relationship between INSTRUCTOR and TEXT, or a ternary relationship between all three entity types? What (min, max) constraints would you put on it? Why?

7.26. Consider an entity type SECTION in a UNIVERSITY database, which describes the section offerings of courses. The attributes of SECTION are Section_number, Semester, Year, Course_number, Instructor, Room_no (where section is taught), Building (where section is taught), Weekdays (domain is the possible combinations of weekdays in which a section can be offered {'MWF', 'MW', 'TT', and so on}), and Hours (domain is all possible time periods during which sections are offered {'9–9:50 A.M.', '10–10:50 A.M.', ..., '3:30–4:50 P.M.', '5:30–6:20 P.M.', and so on}). Assume that Section_number is unique for each course within a particular semester/year combination (that is, if a course is offered multiple times during a particular semester, its section offerings are numbered 1, 2, 3, and so on). There are several composite keys for section, and some attributes are components of more than one key. Identify three composite keys, and show how they can be represented in an ER schema diagram.

7.27. Cardinality ratios often dictate the detailed design of a database. The cardinality ratio depends on the real-world meaning of the entity types involved and is defined by the specific application. For the following binary relationships, suggest cardinality ratios based on the common-sense meaning of the entity types. Clearly state any assumptions you make.

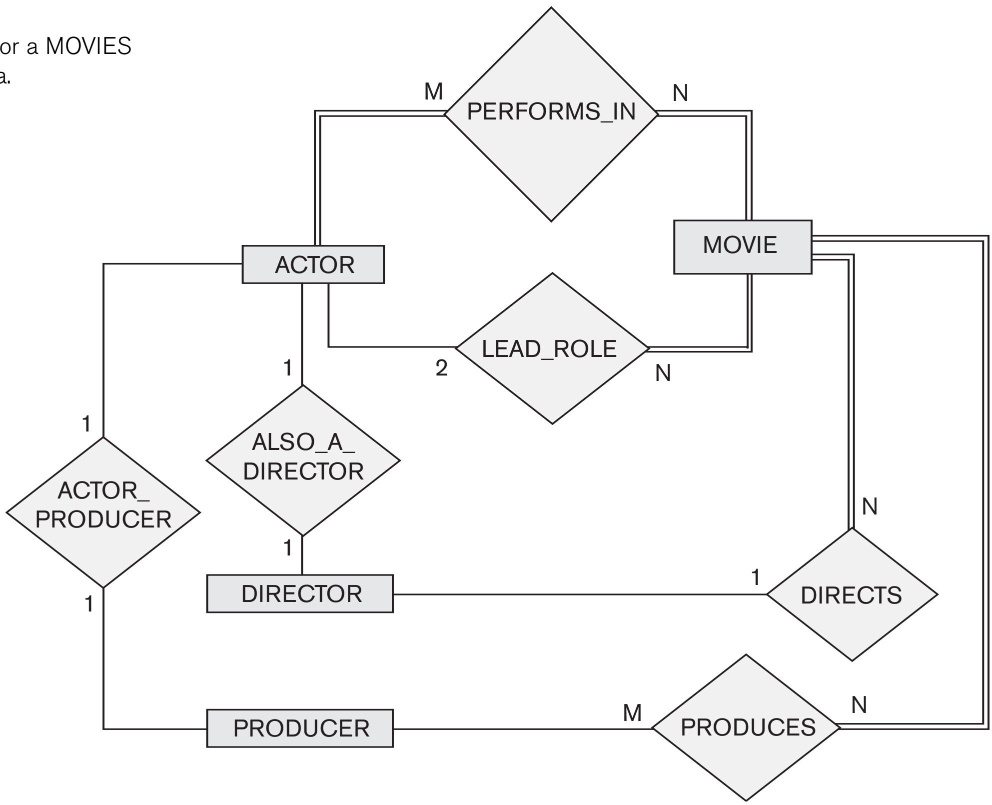
	Entity 1	Cardinality Ratio	Entity 2
1.	STUDENT	______________	SOCIAL_SECURITY_CARD
2.	STUDENT	______________	TEACHER
3.	CLASSROOM	______________	WALL

4. COUNTRY	_______________	CURRENT_PRESIDENT
5. COURSE	_______________	TEXTBOOK
6. ITEM (that can be found in an order)	_______________	ORDER
7. STUDENT	_______________	CLASS
8. CLASS	_______________	INSTRUCTOR
9. INSTRUCTOR	_______________	OFFICE
10. EBAY_AUCTION_ITEM	_______________	EBAY_BID

7.28. Consider the ER schema for the MOVIES database in Figure 7.24.

Assume that MOVIES is a populated database. ACTOR is used as a generic term and includes actresses. Given the constraints shown in the ER schema, respond to the following statements with *True, False*, or *Maybe*. Assign a response of *Maybe* to statements that, while not explicitly shown to be *True*, cannot be proven *False* based on the schema as shown. Justify each answer.

Figure 7.24
An ER diagram for a MOVIES database schema.



a. There are no actors in this database that have been in no movies.
b. There are some actors who have acted in more than ten movies.
c. Some actors have done a lead role in multiple movies.
d. A movie can have only a maximum of two lead actors.
e. Every director has been an actor in some movie.
f. No producer has ever been an actor.
g. A producer cannot be an actor in some other movie.
h. There are movies with more than a dozen actors.
i. Some producers have been a director as well.
j. Most movies have one director and one producer.
k. Some movies have one director but several producers.
l. There are some actors who have done a lead role, directed a movie, and produced some movie.
m. No movie has a director who also acted in that movie.

7.29. Given the ER schema for the MOVIES database in Figure 7.24, draw an instance diagram using three movies that have been released recently. Draw instances of each entity type: MOVIES, ACTORS, PRODUCERS, DIRECTORS involved; make up instances of the relationships as they exist in reality for those movies.

7.30. Illustrate the UML Diagram for Exercise 7.16. Your UML design should observe the following requirements:

a. A student should have the ability to compute his/her GPA and add or drop majors and minors.
b. Each department should be to able add or delete courses and hire or terminate faculty.
c. Each instructor should be able to assign or change a student's grade for a course.

Note: Some of these functions may be spread over multiple classes.

Laboratory Exercises

7.31. Consider the UNIVERSITY database described in Exercise 7.16. Build the ER schema for this database using a data modeling tool such as ERwin or Rational Rose.

7.32. Consider a MAIL_ORDER database in which employees take orders for parts from customers. The data requirements are summarized as follows:

- The mail order company has employees, each identified by a unique employee number, first and last name, and Zip Code.
- Each customer of the company is identified by a unique customer number, first and last name, and Zip Code.

- Each part sold by the company is identified by a unique part number, a part name, price, and quantity in stock.
- Each order placed by a customer is taken by an employee and is given a unique order number. Each order contains specified quantities of one or more parts. Each order has a date of receipt as well as an expected ship date. The actual ship date is also recorded.

Design an Entity-Relationship diagram for the mail order database and build the design using a data modeling tool such as ERwin or Rational Rose.

7.33. Consider a MOVIE database in which data is recorded about the movie industry. The data requirements are summarized as follows:

- Each movie is identified by title and year of release. Each movie has a length in minutes. Each has a production company, and each is classified under one or more genres (such as horror, action, drama, and so forth). Each movie has one or more directors and one or more actors appear in it. Each movie also has a plot outline. Finally, each movie has zero or more quotable quotes, each of which is spoken by a particular actor appearing in the movie.
- Actors are identified by name and date of birth and appear in one or more movies. Each actor has a role in the movie.
- Directors are also identified by name and date of birth and direct one or more movies. It is possible for a director to act in a movie (including one that he or she may also direct).
- Production companies are identified by name and each has an address. A production company produces one or more movies.

Design an Entity-Relationship diagram for the movie database and enter the design using a data modeling tool such as ERwin or Rational Rose.

7.34. Consider a CONFERENCE_REVIEW database in which researchers submit their research papers for consideration. Reviews by reviewers are recorded for use in the paper selection process. The database system caters primarily to reviewers who record answers to evaluation questions for each paper they review and make recommendations regarding whether to accept or reject the paper. The data requirements are summarized as follows:

- Authors of papers are uniquely identified by e-mail id. First and last names are also recorded.
- Each paper is assigned a unique identifier by the system and is described by a title, abstract, and the name of the electronic file containing the paper.
- A paper may have multiple authors, but one of the authors is designated as the contact author.
- Reviewers of papers are uniquely identified by e-mail address. Each reviewer's first name, last name, phone number, affiliation, and topics of interest are also recorded.

- Each paper is assigned between two and four reviewers. A reviewer rates each paper assigned to him or her on a scale of 1 to 10 in four categories: technical merit, readability, originality, and relevance to the conference. Finally, each reviewer provides an overall recommendation regarding each paper.
- Each review contains two types of written comments: one to be seen by the review committee only and the other as feedback to the author(s).

Design an Entity-Relationship diagram for the CONFERENCE_REVIEW database and build the design using a data modeling tool such as ERwin or Rational Rose.

7.35. Consider the ER diagram for the AIRLINE database shown in Figure 7.20. Build this design using a data modeling tool such as ERwin or Rational Rose.

Selected Bibliography

The Entity-Relationship model was introduced by Chen (1976), and related work appears in Schmidt and Swenson (1975), Wiederhold and Elmasri (1979), and Senko (1975). Since then, numerous modifications to the ER model have been suggested. We have incorporated some of these in our presentation. Structural constraints on relationships are discussed in Abrial (1974), Elmasri and Wiederhold (1980), and Lenzerini and Santucci (1983). Multivalued and composite attributes are incorporated in the ER model in Elmasri et al. (1985). Although we did not discuss languages for the ER model and its extensions, there have been several proposals for such languages. Elmasri and Wiederhold (1981) proposed the GORDAS query language for the ER model. Another ER query language was proposed by Markowitz and Raz (1983). Senko (1980) presented a query language for Senko's DIAM model. A formal set of operations called the ER algebra was presented by Parent and Spaccapietra (1985). Gogolla and Hohenstein (1991) presented another formal language for the ER model. Campbell et al. (1985) presented a set of ER operations and showed that they are relationally complete. A conference for the dissemination of research results related to the ER model has been held regularly since 1979. The conference, now known as the International Conference on Conceptual Modeling, has been held in Los Angeles (ER 1979, ER 1983, ER 1997), Washington, D.C. (ER 1981), Chicago (ER 1985), Dijon, France (ER 1986), New York City (ER 1987), Rome (ER 1988), Toronto (ER 1989), Lausanne, Switzerland (ER 1990), San Mateo, California (ER 1991), Karlsruhe, Germany (ER 1992), Arlington, Texas (ER 1993), Manchester, England (ER 1994), Brisbane, Australia (ER 1995), Cottbus, Germany (ER 1996), Singapore (ER 1998), Paris, France (ER 1999), Salt Lake City, Utah (ER 2000), Yokohama, Japan (ER 2001), Tampere, Finland (ER 2002), Chicago, Illinois (ER 2003), Shanghai, China (ER 2004), Klagenfurt, Austria (ER 2005), Tucson, Arizona (ER 2006), Auckland, New Zealand (ER 2007), Barcelona, Catalonia, Spain (ER 2008), and Gramado, RS, Brazil (ER 2009). The 2010 conference is to be held in Vancouver, BC, Canada.

chapter 9

Relational Database Design by ER- and EER-to-Relational Mapping

This chapter discusses how to **design a relational database schema** based on a conceptual schema design. Figure 7.1 presented a high-level view of the database design process, and in this chapter we focus on the logical database design or data model mapping step of database design. We present the procedures to create a relational schema from an Entity-Relationship (ER) or an Enhanced ER (EER) schema. Our discussion relates the constructs of the ER and EER models, presented in Chapters 7 and 8, to the constructs of the relational model, presented in Chapters 3 through 6. Many computer-aided software engineering (CASE) tools are based on the ER or EER models, or other similar models, as we have discussed in Chapters 7 and 8. Many tools use ER or EER diagrams or variations to develop the schema graphically, and then convert it automatically into a relational database schema in the DDL of a specific relational DBMS by employing algorithms similar to the ones presented in this chapter.

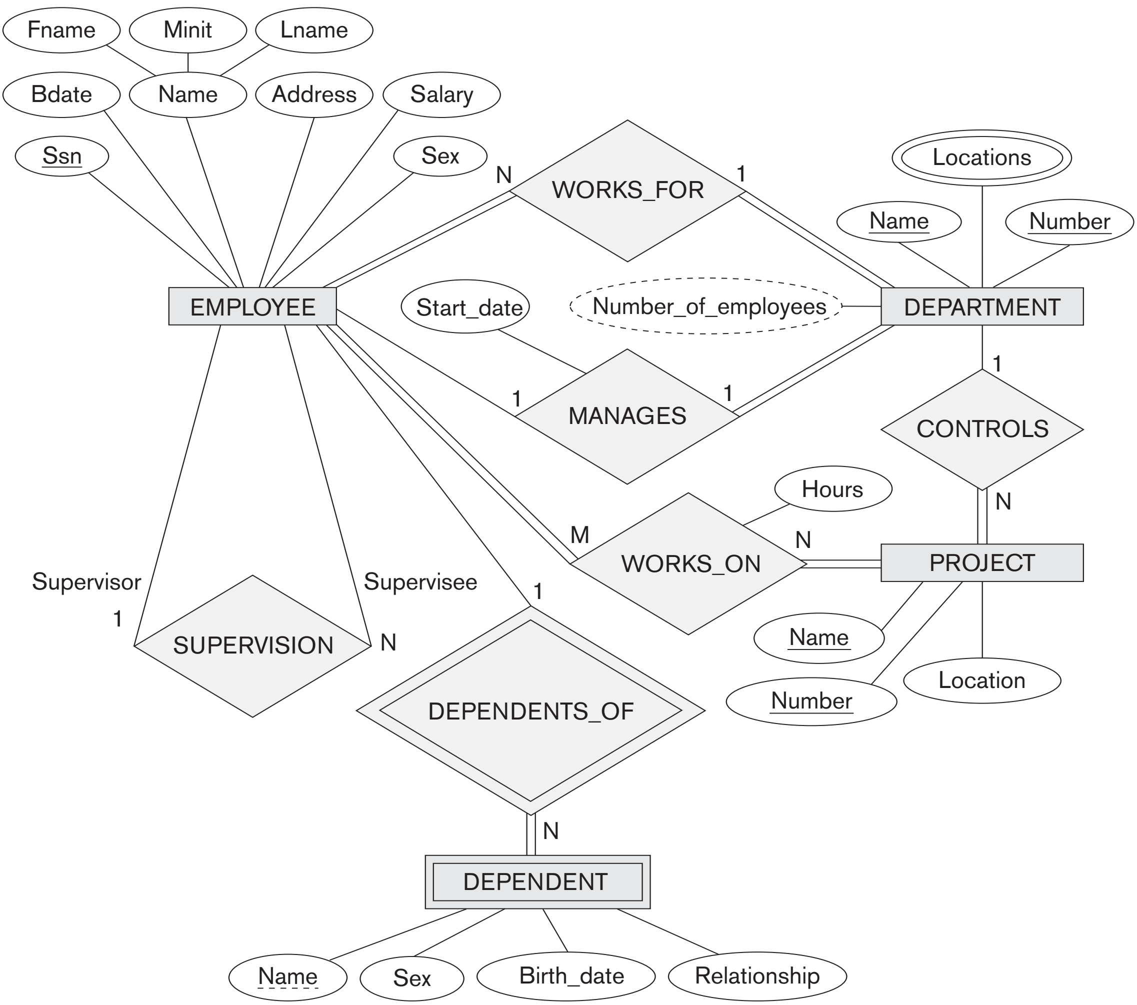
We outline a seven-step algorithm in Section 9.1 to convert the basic ER model constructs—entity types (strong and weak), binary relationships (with various structural constraints), *n*-ary relationships, and attributes (simple, composite, and multivalued)—into relations. Then, in Section 9.2, we continue the mapping algorithm by describing how to map EER model constructs—specialization/generalization and union types (categories)—into relations. Section 9.3 summarizes the chapter.

9.1 Relational Database Design Using ER-to-Relational Mapping

9.1.1 ER-to-Relational Mapping Algorithm

In this section we describe the steps of an algorithm for ER-to-relational mapping. We use the COMPANY database example to illustrate the mapping procedure. The COMPANY ER schema is shown again in Figure 9.1, and the corresponding COMPANY relational database schema is shown in Figure 9.2 to illustrate the map-

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



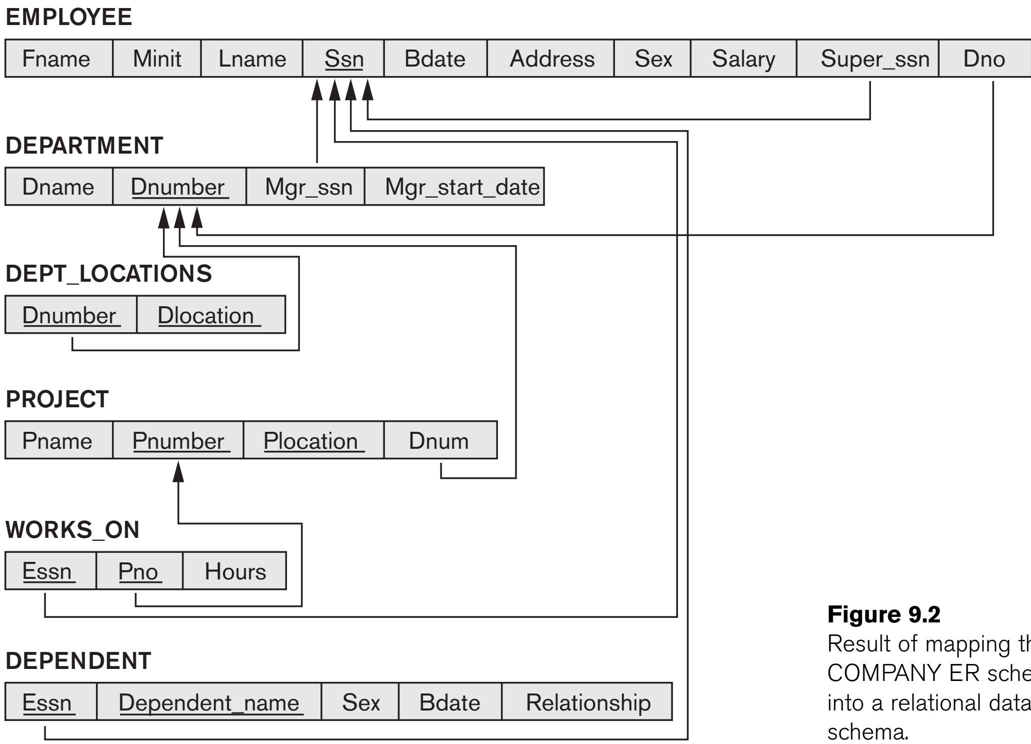


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

ping steps. We assume that the mapping will create tables with simple single-valued attributes. The relational model constraints defined in Chapter 3, which include primary keys, unique keys (if any), and referential integrity constraints on the relations, will also be specified in the mapping results.

Step 1: Mapping of Regular Entity Types. For each regular (strong) entity type *E* in the ER schema, create a relation *R* that includes all the simple attributes of *E*. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of *E* as the primary key for *R*. If the chosen key of *E* is a composite, then the set of simple attributes that form it will together form the primary key of *R*.

If multiple keys were identified for *E* during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation *R*. Knowledge about keys is also kept for indexing purposes and other types of analyses.

In our example, we create the relations EMPLOYEE, DEPARTMENT, and PROJECT in Figure 9.2 to correspond to the regular entity types EMPLOYEE, DEPARTMENT, and PROJECT in Figure 9.1. The foreign key and relationship attributes, if any, are not included yet; they will be added during subsequent steps. These include the attributes Super_ssn and Dno of EMPLOYEE, Mgr_ssn and Mgr_start_date of DEPARTMENT, and Dnum of PROJECT. In our example, we choose Ssn, Dnumber, and Pnumber as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT, respectively. Knowledge that Dname of DEPARTMENT and Pname of PROJECT are secondary keys is kept for possible use later in the design.

The relations that are created from the mapping of entity types are sometimes called **entity relations** because each tuple represents an entity instance. The result after this mapping step is shown in Figure 9.3(a).

Step 2: Mapping of Weak Entity Types. For each weak entity type *W* in the ER schema with owner entity type *E*, create a relation *R* and include all simple attributes (or simple components of composite attributes) of *W* as attributes of *R*. In addition, include as foreign key attributes of *R*, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the identifying relationship type of *W*. The primary key of *R* is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type *W*, if any.

If there is a weak entity type E_2 whose owner is also a weak entity type E_1, then E_1 should be mapped before E_2 to determine its primary key first.

In our example, we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT (see Figure 9.3(b)). We include the primary key Ssn of the EMPLOYEE relation—which corresponds to the owner entity type—as a foreign key attribute of DEPENDENT; we rename it Essn, although this is not necessary.

Figure 9.3
Illustration of some mapping steps.
(a) *Entity* relations after step 1.
(b) Additional *weak entity* relation after step 2.
(c) *Relationship* relation after step 5.
(d) Relation representing multivalued attribute after step 6.

(a) **EMPLOYEE**

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	<u>Dnumber</u>

PROJECT

Pname	<u>Pnumber</u>	Plocation

(b) **DEPENDENT**

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship

(c) **WORKS_ON**

<u>Essn</u>	<u>Pno</u>	Hours

(d) **DEPT_LOCATIONS**

<u>Dnumber</u>	<u>Dlocation</u>

The primary key of the DEPENDENT relation is the combination {Essn, Dependent_name}, because Dependent_name (also renamed from Name in Figure 9.1) is the partial key of DEPENDENT.

It is common to choose the propagate (CASCADE) option for the referential triggered action (see Section 4.2) on the foreign key in the relation corresponding to the weak entity type, since a weak entity has an existence dependency on its owner entity. This can be used for both ON UPDATE and ON DELETE.

Step 3: Mapping of Binary 1:1 Relationship Types. For each binary 1:1 relationship type *R* in the ER schema, identify the relations *S* and *T* that correspond to the entity types participating in *R*. There are three possible approaches: (1) the foreign key approach, (2) the merged relationship approach, and (3) the cross-reference or relationship relation approach. The first approach is the most useful and should be followed unless special conditions exist, as we discuss below.

1. **Foreign key approach:** Choose one of the relations—*S*, say—and include as a foreign key in *S* the primary key of *T*. It is better to choose an entity type with *total participation* in *R* in the role of *S*. Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type *R* as attributes of *S*.

   In our example, we map the 1:1 relationship type MANAGES from Figure 9.1 by choosing the participating entity type DEPARTMENT to serve in the role of *S* because its participation in the MANAGES relationship type is total (every department has a manager). We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it Mgr_ssn. We also include the simple attribute Start_date of the MANAGES relationship type in the DEPARTMENT relation and rename it Mgr_start_date (see Figure 9.2).

   Note that it is possible to include the primary key of *S* as a foreign key in *T* instead. In our example, this amounts to having a foreign key attribute, say Department_managed in the EMPLOYEE relation, but it will have a NULL value for employee tuples who do not manage a department. If only 2 percent of employees manage a department, then 98 percent of the foreign keys would be NULL in this case. Another possibility is to have foreign keys in both relations *S* and *T* redundantly, but this creates redundancy and incurs a penalty for consistency maintenance.

2. **Merged relation approach:** An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation. This is possible when *both participations are total,* as this would indicate that the two tables will have the exact same number of tuples at all times.

3. **Cross-reference or relationship relation approach:** The third option is to set up a third relation *R* for the purpose of cross-referencing the primary keys of the two relations *S* and *T* representing the entity types. As we will see, this approach is required for binary M:N relationships. The relation *R* is called a **relationship relation** (or sometimes a **lookup table**), because each

tuple in *R* represents a relationship instance that relates one tuple from *S* with one tuple from *T*. The relation *R* will include the primary key attributes of *S* and *T* as foreign keys to *S* and *T*. The primary key of *R* will be one of the two foreign keys, and the other foreign key will be a unique key of *R*. The drawback is having an extra relation, and requiring an extra join operation when combining related tuples from the tables.

Step 4: Mapping of Binary 1:N Relationship Types. For each regular binary 1:N relationship type *R*, identify the relation *S* that represents the participating entity type at the *N-side* of the relationship type. Include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*; we do this because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type. Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of *S*.

In our example, we now map the 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION from Figure 9.1. For WORKS_FOR we include the primary key Dnumber of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it Dno. For SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself—because the relationship is recursive—and call it Super_ssn. The CONTROLS relationship is mapped to the foreign key attribute Dnum of PROJECT, which references the primary key Dnumber of the DEPARTMENT relation. These foreign keys are shown in Figure 9.2.

An alternative approach is to use the **relationship relation** (cross-reference) option as in the third option for binary 1:1 relationships. We create a separate relation *R* whose attributes are the primary keys of *S* and *T*, which will also be foreign keys to *S* and *T*. The primary key of *R* is the same as the primary key of *S*. This option can be used if few tuples in *S* participate in the relationship to avoid excessive NULL values in the foreign key.

Step 5: Mapping of Binary M:N Relationship Types. For each binary M:N relationship type *R*, create a new relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types; their *combination* will form the primary key of *S*. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of *S*. Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations (as we did for 1:1 or 1:N relationship types) because of the M:N cardinality ratio; we must create a separate *relationship relation S*.

In our example, we map the M:N relationship type WORKS_ON from Figure 9.1 by creating the relation WORKS_ON in Figure 9.2. We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively. We also include an attribute Hours in WORKS_ON to represent the Hours attribute of the relationship type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {Essn, Pno}. This **relationship relation** is shown in Figure 9.3(c).

The propagate (CASCADE) option for the referential triggered action (see Section 4.2) should be specified on the foreign keys in the relation corresponding to the relationship *R*, since each relationship instance has an existence dependency on each of the entities it relates. This can be used for both ON UPDATE and ON DELETE.

Notice that we can always map 1:1 or 1:N relationships in a manner similar to M:N relationships by using the cross-reference (relationship relation) approach, as we discussed earlier. This alternative is particularly useful when few relationship instances exist, in order to avoid NULL values in foreign keys. In this case, the primary key of the relationship relation will be *only one* of the foreign keys that reference the participating entity relations. For a 1:N relationship, the primary key of the relationship relation will be the foreign key that references the entity relation on the N-side. For a 1:1 relationship, either foreign key can be used as the primary key of the relationship relation.

Step 6: Mapping of Multivalued Attributes. For each multivalued attribute *A*, create a new relation *R*. This relation *R* will include an attribute corresponding to *A*, plus the primary key attribute *K*—as a foreign key in *R*—of the relation that represents the entity type or relationship type that has *A* as a multivalued attribute. The primary key of *R* is the combination of *A* and *K*. If the multivalued attribute is composite, we include its simple components.

In our example, we create a relation DEPT_LOCATIONS (see Figure 9.3(d)). The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, while Dnumber—as foreign key—represents the primary key of the DEPARTMENT relation. The primary key of DEPT_LOCATIONS is the combination of {Dnumber, Dlocation}. A separate tuple will exist in DEPT_LOCATIONS for each location that a department has.

The propagate (CASCADE) option for the referential triggered action (see Section 4.2) should be specified on the foreign key in the relation *R* corresponding to the multivalued attribute for both ON UPDATE and ON DELETE. We should also note that the key of *R* when mapping a composite, multivalued attribute requires some analysis of the meaning of the component attributes. In some cases, when a multivalued attribute is composite, only some of the component attributes are required to be part of the key of *R*; these attributes are similar to a partial key of a weak entity type that corresponds to the multivalued attribute (see Section 7.5).

Figure 9.2 shows the COMPANY relational database schema obtained with steps 1 through 6, and Figure 3.6 shows a sample database state. Notice that we did not yet discuss the mapping of *n*-ary relationship types ($n > 2$) because none exist in Figure 9.1; these are mapped in a similar way to M:N relationship types by including the following additional step in the mapping algorithm.

Step 7: Mapping of *N*-ary Relationship Types. For each *n*-ary relationship type *R*, where $n > 2$, create a new relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the *n*-ary relationship type (or simple components of composite attributes) as attributes of *S*. The primary key of *S* is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types *E* participating in *R* is 1, then the primary key of *S* should not include the foreign key attribute that references the relation *E′* corresponding to *E* (see the discussion in Section 7.9.2 concerning constraints on *n*-ary relationships).

For example, consider the relationship type SUPPLY in Figure 7.17. This can be mapped to the relation SUPPLY shown in Figure 9.4, whose primary key is the combination of the three foreign keys {Sname, Part_no, Proj_name}.

9.1.2 Discussion and Summary of Mapping for ER Model Constructs

Table 9.1 summarizes the correspondences between ER and relational model constructs and constraints.

One of the main points to note in a relational schema, in contrast to an ER schema, is that relationship types are not represented explicitly; instead, they are represented by having two attributes *A* and *B*, one a primary key and the other a foreign key (over the same domain) included in two relations *S* and *T*. Two tuples in *S* and *T* are related when they have the same value for *A* and *B*. By using the EQUIJOIN operation (or NATURAL JOIN if the two join attributes have the same name) over *S.A* and *T.B*, we can combine all pairs of related tuples from *S* and *T* and materialize the relationship. When a binary 1:1 or 1:N relationship type is involved, a single join operation is usually needed. For a binary M:N relationship type, two join operations are needed, whereas for *n*-ary relationship types, *n* joins are needed to fully materialize the relationship instances.

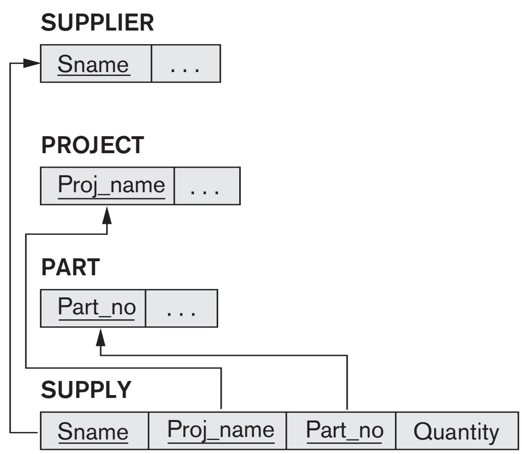


Figure 9.4
Mapping the *n*-ary relationship type SUPPLY from Figure 7.17(a).

Table 9.1 Correspondence between ER and Relational Models

ER MODEL	**RELATIONAL MODEL**
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

For example, to form a relation that includes the employee name, project name, and hours that the employee works on each project, we need to connect each EMPLOYEE tuple to the related PROJECT tuples via the WORKS_ON relation in Figure 9.2. Hence, we must apply the EQUIJOIN operation to the EMPLOYEE and WORKS_ON relations with the join condition Ssn = Essn, and then apply another EQUIJOIN operation to the resulting relation and the PROJECT relation with join condition Pno = Pnumber. In general, when multiple relationships need to be traversed, numerous join operations must be specified. A relational database user must always be aware of the foreign key attributes in order to use them correctly in combining related tuples from two or more relations. This is sometimes considered to be a drawback of the relational data model, because the foreign key/primary key correspondences are not always obvious upon inspection of relational schemas. If an EQUIJOIN is performed among attributes of two relations that do not represent a foreign key/primary key relationship, the result can often be meaningless and may lead to spurious data. For example, the reader can try joining the PROJECT and DEPT_LOCATIONS relations on the condition Dlocation = Plocation and examine the result (see the discussion of spurious tuples in Section 15.1.4).

In the relational schema we create a separate relation for *each* multivalued attribute. For a particular entity with a set of values for the multivalued attribute, the key attribute value of the entity is repeated once for each value of the multivalued attribute in a separate tuple because the basic relational model does *not* allow multiple values (a list, or a set of values) for an attribute in a single tuple. For example, because department 5 has three locations, three tuples exist in the DEPT_LOCATIONS relation in Figure 3.6; each tuple specifies one of the locations. In our example, we apply EQUIJOIN to DEPT_LOCATIONS and DEPARTMENT on the Dnumber attribute to get the values of all locations along with other DEPARTMENT attributes. In the resulting relation, the values of the other DEPARTMENT attributes are repeated in separate tuples for every location that a department has.

The basic relational algebra does not have a NEST or COMPRESS operation that would produce a set of tuples of the form {<'1', 'Houston'>, <'4', 'Stafford'>, <'5', {'Bellaire', 'Sugarland', 'Houston'}>} from the DEPT_LOCATIONS relation in Figure 3.6. This is a serious drawback of the basic normalized or *flat* version of the relational model. The object data model and object-relational systems (see Chapter 11) do allow multivalued attributes.

9.2 Mapping EER Model Constructs to Relations

Next, we discuss the mapping of EER model constructs to relations by extending the ER-to-relational mapping algorithm that was presented in Section 9.1.1.

9.2.1 Mapping of Specialization or Generalization

There are several options for mapping a number of subclasses that together form a specialization (or alternatively, that are generalized into a superclass), such as the {SECRETARY, TECHNICIAN, ENGINEER} subclasses of EMPLOYEE in Figure 8.4. We can add a further step to our ER-to-relational mapping algorithm from Section 9.1.1, which has seven steps, to handle the mapping of specialization. Step 8, which follows, gives the most common options; other mappings are also possible. We discuss the conditions under which each option should be used. We use Attrs(R) to denote *the attributes of relation R*, and PK(R) to denote the *primary key of R*. First we describe the mapping formally, then we illustrate it with examples.

Step 8: Options for Mapping Specialization or Generalization. Convert each specialization with m subclasses $\{S_1, S_2, ..., S_m\}$ and (generalized) superclass C, where the attributes of C are $\{k, a_1, ...a_n\}$ and k is the (primary) key, into relation schemas using one of the following options:

- **Option 8A: Multiple relations—superclass and subclasses.** Create a relation L for C with attributes Attrs(L) = $\{k, a_1, ..., a_n\}$ and PK(L) = k. Create a relation L_i for each subclass S_i, $1 \leq i \leq m$, with the attributes Attrs(L_i) = $\{k\} \cup$ {attributes of S_i} and PK(L_i) = k. This option works for any specialization (total or partial, disjoint or overlapping).
- **Option 8B: Multiple relations—subclass relations only.** Create a relation L_i for each subclass S_i, $1 \leq i \leq m$, with the attributes Attrs(L_i) = {attributes of S_i} $\cup$ $\{k, a_1, ..., a_n\}$ and PK(L_i) = k. This option only works for a specialization whose subclasses are *total* (every entity in the superclass must belong to (at least) one of the subclasses). Additionally, it is only recommended if the specialization has the *disjointedness constraint* (see Section 8.3.1).If the specialization is *overlapping*, the same entity may be duplicated in several relations.
- **Option 8C: Single relation with one type attribute.** Create a single relation L with attributes Attrs(L) = $\{k, a_1, ..., a_n\}$ $\cup$ {attributes of S_1} $\cup$ ... $\cup$ {attributes of S_m} $\cup$ $\{t\}$ and PK(L) = k. The attribute t is called a **type** (or

For a category whose superclasses have the same key, such as VEHICLE in Figure 8.8, there is no need for a surrogate key. The mapping of the REGISTERED_VEHICLE category, which illustrates this case, is also shown in Figure 9.7.

9.3 Summary

In Section 9.1, we showed how a conceptual schema design in the ER model can be mapped to a relational database schema. An algorithm for ER-to-relational mapping was given and illustrated by examples from the COMPANY database. Table 9.1 summarized the correspondences between the ER and relational model constructs and constraints. Next, we added additional steps to the algorithm in Section 9.2 for mapping the constructs from the EER model into the relational model. Similar algorithms are incorporated into graphical database design tools to create a relational schema from a conceptual schema design automatically.

Review Questions

9.1. Discuss the correspondences between the ER model constructs and the relational model constructs. Show how each ER model construct can be mapped to the relational model and discuss any alternative mappings.

9.2. Discuss the options for mapping EER model constructs to relations.

Exercises

9.3. Try to map the relational schema in Figure 6.14 into an ER schema. This is part of a process known as *reverse engineering*, where a conceptual schema is created for an existing implemented database. State any assumptions you make.

9.4. Figure 9.8 shows an ER schema for a database that can be used to keep track of transport ships and their locations for maritime authorities. Map this schema into a relational schema and specify all primary keys and foreign keys.

9.5. Map the BANK ER schema of Exercise 7.23 (shown in Figure 7.21) into a relational schema. Specify all primary keys and foreign keys. Repeat for the AIRLINE schema (Figure 7.20) of Exercise 7.19 and for the other schemas for Exercises 7.16 through 7.24.

9.6. Map the EER diagrams in Figures 8.9 and 8.12 into relational schemas. Justify your choice of mapping options.

9.7. Is it possible to successfully map a binary M:N relationship type without requiring a new relation? Why or why not?

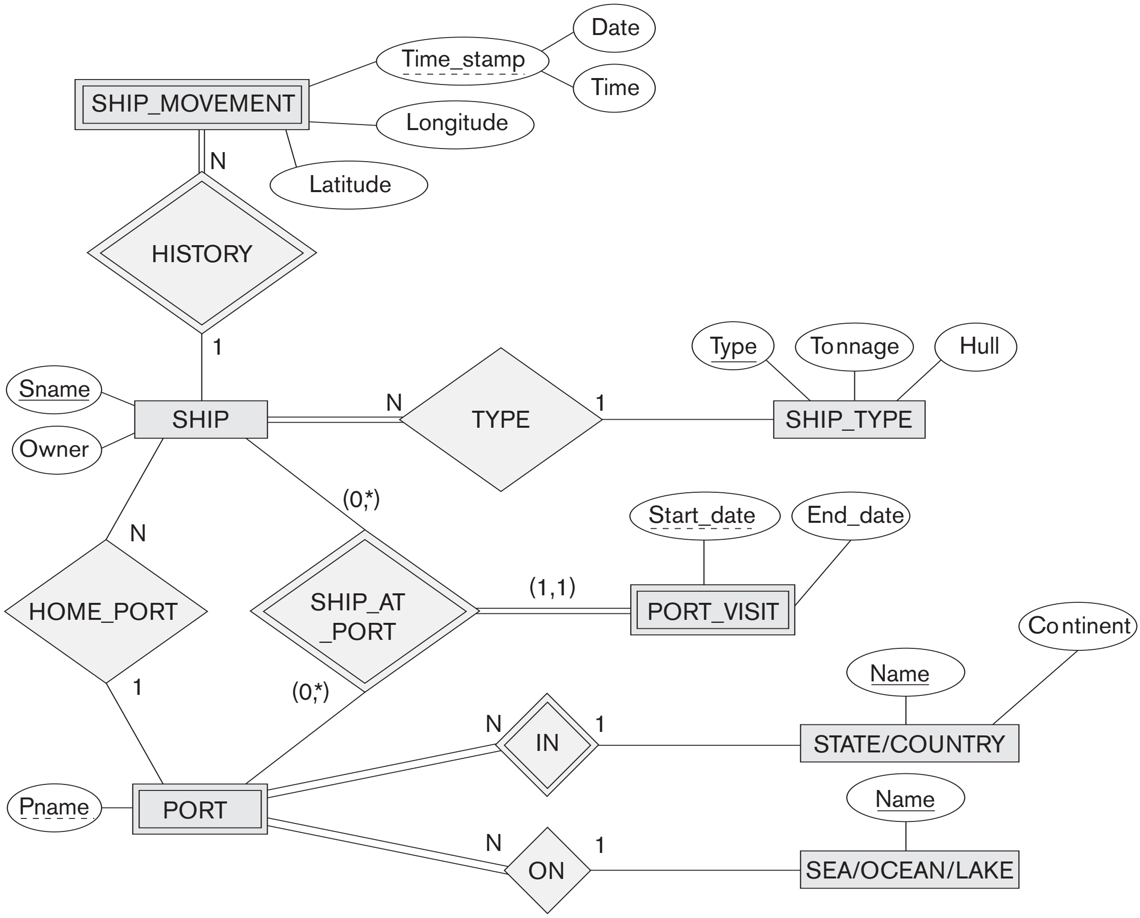


Figure 9.8
An ER schema for a SHIP_TRACKING database.

9.8. Consider the EER diagram in Figure 9.9 for a car dealer.

Map the EER schema into a set of relations. For the VEHICLE to CAR/TRUCK/SUV generalization, consider the four options presented in Section 9.2.1 and show the relational schema design under each of those options.

9.9. Using the attributes you provided for the EER diagram in Exercise 8.27, map the complete schema into a set of relations. Choose an appropriate option out of 8A thru 8D from Section 9.2.1 in doing the mapping of generalizations and defend your choice.

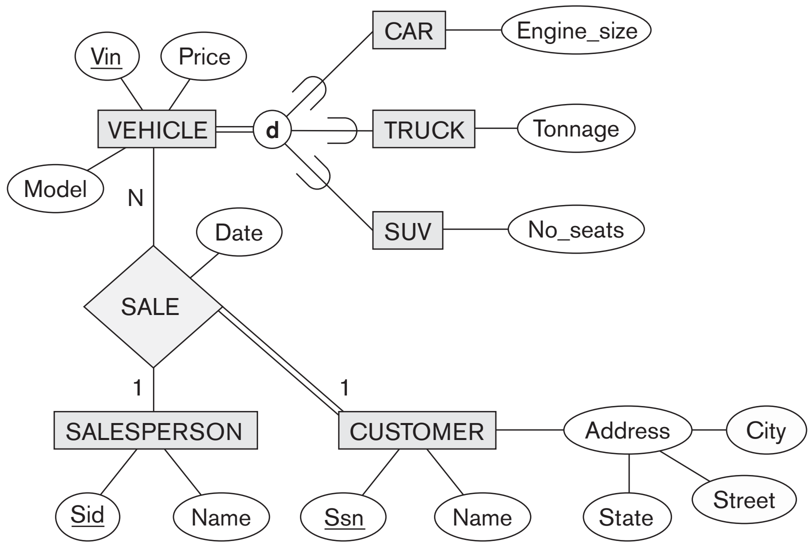


Figure 9.9
EER diagram for a car dealer

Laboratory Exercises

9.10. Consider the ER design for the UNIVERSITY database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 7.31. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

9.11. Consider the ER design for the MAIL_ORDER database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 7.32. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

9.12. Consider the ER design for the CONFERENCE_REVIEW database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 7.34. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

9.13. Consider the EER design for the GRADE_BOOK database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 8.28. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

9.14. Consider the EER design for the ONLINE_AUCTION database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 8.29. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.