Chapter 4

High-Level Database Models

Let us consider the process whereby a new database, such as our movie database, is created. Figure 4.1 suggests the process. We begin with a design phase, in which we address and answer questions about what information will be stored, how information elements will be related to one another, what constraints such as keys or referential integrity may be assumed, and so on. This phase may last for a long time, while options are evaluated and opinions are reconciled. We show this phase in Fig. 4.1 as the conversion of ideas to a high-level design.

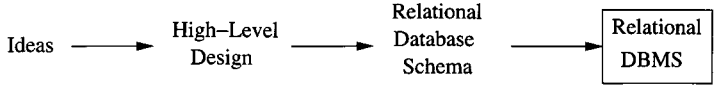

Figure 4.1: The database modeling and implementation process

Since the great majority of commercial database systems use the relational model, we might suppose that the design phase should use this model too. However, in practice it is often easier to start with a higher-level model and then convert the design to the relational model. The primary reason for doing so is that the relational model has only one concept — the relation — rather than several complementary concepts that more closely model real-world situations. Simplicity of concepts in the relational model is a great strength of the model, especially when it comes to efficient implementation of database operations. Yet that strength becomes a weakness when we do a preliminary design, which is why it often is helpful to begin by using a high-level design model.

There are several options for the notation in which the design is expressed. The first, and oldest, method is the "entity-relationship diagram," and here is where we shall start in Section 4.1. A more recent trend is the use of UML ("Unified Modeling Language"), a notation that was originally designed for

describing object-oriented software projects, but which has been adapted to describe database schemas as well. We shall see this model in Section 4.7. Finally, in Section 4.9, we shall consider ODL ("Object Description Language"), which was created to describe databases as collections of classes and their objects.

The next phase shown in Fig. 4.1 is the conversion of our high-level design to a relational design. This phase occurs only when we are confident of the high-level design. Whichever of the high-level models we use, there is a fairly mechanical way of converting the high-level design into a relational database schema, which then runs on a conventional DBMS. Sections 4.5 and 4.6 discuss conversion of E/R diagrams to relational database schemas. Section 4.8 does the same for UML, and Section 4.10 serves for ODL.

4.1 The Entity/Relationship Model

In the *entity-relationship model* (or *E/R model*). the structure of data is represented graphically, as an "entity-relationship diagram," using three principal element types:

1. Entity sets,
2. Attributes, and
3. Relationships.

We shall cover each in turn.

4.1.1 Entity Sets

An *entity* is an abstract object of some sort, and a collection of similar entities forms an *entity set*. An entity in some ways resembles an "object" in the sense of object-oriented programming. Likewise, an entity set bears some resemblance to a class of objects. However, the E/R model is a static concept, involving the structure of data and not the operations on data. Thus, one would not expect to find methods associated with an entity set as one would with a class.

Example 4.1: Let us consider the design of our running movie-database example. Each movie is an entity, and the set of all movies constitutes an entity set. Likewise, the stars are entities, and the set of stars is an entity set. A studio is another kind of entity, and the set of studios is a third entity set that will appear in our examples. □

4.1.2 Attributes

Entity sets have associated *attributes*, which are properties of the entities in that set. For instance, the entity set *Movies* might be given attributes such as *title* and *length*. It should not surprise you if the attributes for the entity

E/R Model Variations

In some versions of the E/R model, the type of an attribute can be either:

1. A primitive type, as in the version presented here.
2. A "struct," as in C, or tuple with a fixed number of primitive components.
3. A set of values of one type: either primitive or a "struct" type.

For example, the type of an attribute in such a model could be a set of pairs, each pair consisting of an integer and a string.

set *Movies* resemble the attributes of the relation `Movies` in our example. It is common for entity sets to be implemented as relations, although not every relation in our final relational design will come from an entity set.

In our version of the E/R model, we shall assume that attributes are of primitive types, such as strings, integers, or reals. There are other variations of this model in which attributes can have some limited structure; see the box on "E/R Model Variations."

4.1.3 Relationships

Relationships are connections among two or more entity sets. For instance, if *Movies* and *Stars* are two entity sets, we could have a relationship *Stars-in* that connects movies and stars. The intent is that a movie entity m is related to a star entity s by the relationship *Stars-in* if s appears in movie m. While binary relationships, those between two entity sets, are by far the most common type of relationship, the E/R model allows relationships to involve any number of entity sets. We shall defer discussion of these multiway relationships until Section 4.1.7.

4.1.4 Entity-Relationship Diagrams

An *E/R diagram* is a graph representing entity sets, attributes, and relationships. Elements of each of these kinds are represented by nodes of the graph, and we use a special shape of node to indicate the kind, as follows:

- Entity sets are represented by rectangles.
- Attributes are represented by ovals.
- Relationships are represented by diamonds.

Edges connect an entity set to its attributes and also connect a relationship to its entity sets.

Example 4.2: In Fig. 4.2 is an E/R diagram that represents a simple database about movies. The entity sets are *Movies*, *Stars*, and *Studios*.

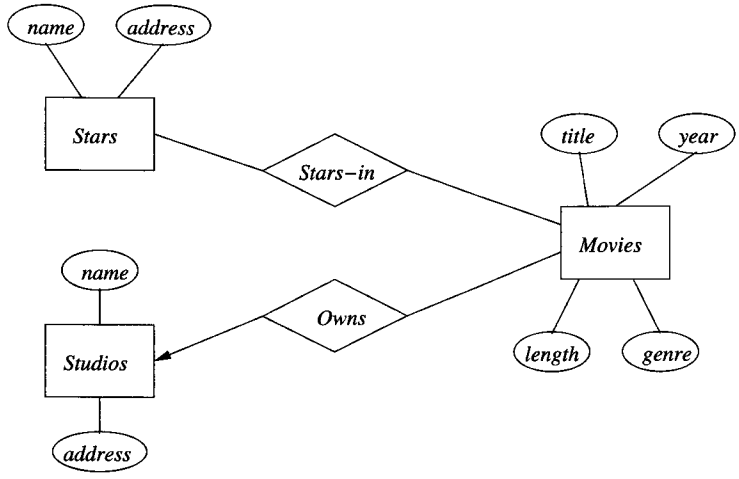

Figure 4.2: An entity-relationship diagram for the movie database

The *Movies* entity set has four of our usual attributes: *title*, *year*, *length*, and *genre*. The other two entity sets *Stars* and *Studios* happen to have the same two attributes: *name* and *address*, each with an obvious meaning. We also see two relationships in the diagram:

1. *Stars-in* is a relationship connecting each movie to the stars of that movie. This relationship consequently also connects stars to the movies in which they appeared.

2. *Owns* connects each movie to the studio that owns the movie. The arrow pointing to entity set *Studios* in Fig. 4.2 indicates that each movie is owned by at most one studio. We shall discuss uniqueness constraints such as this one in Section 4.1.6.

□

4.1.5 Instances of an E/R Diagram

E/R diagrams are a notation for describing schemas of databases. We may imagine that a database described by an E/R diagram contains particular data, an "instance" of the database. Since the database is not implemented in the E/R model, only designed, the instance never exists in the sense that a relation's

instances exist in a DBMS. However, it is often useful to visualize the database being designed as if it existed.

For each entity set, the database instance will have a particular finite set of entities. Each of these entities has particular values for each attribute. A relationship R that connects n entity sets $E_1, E_2, \ldots, E_n$ may be imagined to have an "instance" that consists of a finite set of tuples $(e_1, e_2, \ldots, e_n)$, where each e_i is chosen from the entities that are in the current instance of entity set E_i. We regard each of these tuples as "connected" by relationship R.

This set of tuples is called the *relationship set* for R. It is often helpful to visualize a relationship set as a table or relation. However, the "tuples" of a relationship set are not really tuples of a relation, since their components are entities rather than primitive types such as strings or integers. The columns of the table are headed by the names of the entity sets involved in the relationship, and each list of connected entities occupies one row of the table. As we shall see, however, when we convert relationships to relations, the resulting relation is not the same as the relationship set.

Example 4.3: An instance of the *Stars-in* relationship could be visualized as a table with pairs such as:

Movies	*Stars*
Basic Instinct	Sharon Stone
Total Recall	Arnold Schwarzenegger
Total Recall	Sharon Stone

The members of the relationship set are the rows of the table. For instance, (Basic Instinct, Sharon Stone) is a tuple in the relationship set for the current instance of relationship *Stars-in*. □

4.1.6 Multiplicity of Binary E/R Relationships

In general, a binary relationship can connect any member of one of its entity sets to any number of members of the other entity set. However, it is common for there to be a restriction on the "multiplicity" of a relationship. Suppose R is a relationship connecting entity sets E and F. Then:

- If each member of E can be connected by R to at most one member of F, then we say that R is *many-one* from E to F. Note that in a many-one relationship from E to F, each entity in F can be connected to many members of E. Similarly, if instead a member of F can be connected by R to at most one member of E, then we say R is many-one from F to E (or equivalently, one-many from E to F).

- If R is both many-one from E to F and many-one from F to E, then we say that R is *one-one*. In a one-one relationship an entity of either entity set can be connected to at most one entity of the other set.

- If R is neither many-one from E to F or from F to E, then we say R is *many-many*.

As we mentioned in Example 4.2, arrows can be used to indicate the multiplicity of a relationship in an E/R diagram. If a relationship is many-one from entity set E to entity set F, then we place an arrow entering F. The arrow indicates that each entity in set E is related to at most one entity in set F. Unless there is also an arrow on the edge to E, an entity in F may be related to many entities in E.

Example 4.4: A one-one relationship between entity sets E and F is represented by arrows pointing to both E and F. For instance, Fig. 4.3 shows two entity sets, *Studios* and *Presidents*, and the relationship *Runs* between them (attributes are omitted). We assume that a president can run only one studio and a studio has only one president, so this relationship is one-one, as indicated by the two arrows, one entering each entity set.

Figure 4.3: A one-one relationship

Remember that the arrow means "at most one"; it does not guarantee existence of an entity of the set pointed to. Thus, in Fig. 4.3, we would expect that a "president" is surely associated with some studio; how could they be a "president" otherwise? However, a studio might not have a president at some particular time, so the arrow from *Runs* to *Presidents* truly means "at most one" and not "exactly one." We shall discuss the distinction further in Section 4.3.3. □

4.1.7 Multiway Relationships

The E/R model makes it convenient to define relationships involving more than two entity sets. In practice, ternary (three-way) or higher-degree relationships are rare, but they occasionally are necessary to reflect the true state of affairs. A multiway relationship in an E/R diagram is represented by lines from the relationship diamond to each of the involved entity sets.

Example 4.5: In Fig. 4.4 is a relationship *Contracts* that involves a studio, a star, and a movie. This relationship represents that a studio has contracted with a particular star to act in a particular movie. In general, the value of an E/R relationship can be thought of as a relationship set of tuples whose components are the entities participating in the relationship, as we discussed in Section 4.1.5. Thus, relationship *Contracts* can be described by triples of the form (studio, star, movie).

Implications Among Relationship Types

We should be aware that a many-one relationship is a special case of a many-many relationship, and a one-one relationship is a special case of a many-one relationship. Thus, any useful property of many-one relationships holds for one-one relationships too. For example, a data structure for representing many-one relationships will work for one-one relationships, although it might not be suitable for many-many relationships.

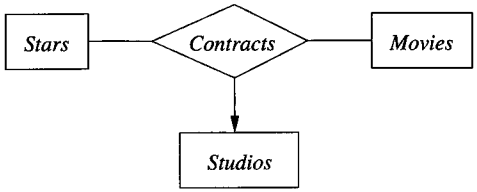

Figure 4.4: A three-way relationship

In multiway relationships, an arrow pointing to an entity set E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E. (Note that this rule generalizes the notation used for many-one, binary relationships.) Informally, we may think of a functional dependency with E on the right and all the other entity sets of the relationship on the left.

In Fig. 4.4 we have an arrow pointing to entity set *Studios*, indicating that for a particular star and movie, there is only one studio with which the star has contracted for that movie. However, there are no arrows pointing to entity sets *Stars* or *Movies*. A studio may contract with several stars for a movie, and a star may contract with one studio for more than one movie. □

4.1.8 Roles in Relationships

It is possible that one entity set appears two or more times in a single relationship. If so, we draw as many lines from the relationship to the entity set as the entity set appears in the relationship. Each line to the entity set represents a different *role* that the entity set plays in the relationship. We therefore label the edges between the entity set and relationship by names, which we call "roles."

Example 4.6: In Fig. 4.5 is a relationship *Sequel-of* between the entity set *Movies* and itself. Each relationship is between two movies, one of which is the sequel of the other. To differentiate the two movies in a relationship, one line is labeled by the role *Original* and one by the role *Sequel*, indicating the

Limits on Arrow Notation in Multiway Relationships

There are not enough choices of arrow or no-arrow on the lines attached to a relationship with three or more participants. Thus, we cannot describe every possible situation with arrows. For instance, in Fig. 4.4, the studio is really a function of the movie alone, not the star and movie jointly, since only one studio produces a movie. However, our notation does not distinguish this situation from the case of a three-way relationship where the entity set pointed to by the arrow is truly a function of both other entity sets. To handle all possible situations, we would have to give a set of functional dependencies involving the entity sets of the relationship.

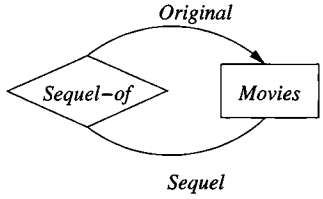

Figure 4.5: A relationship with roles

original movie and its sequel, respectively. We assume that a movie may have many sequels, but for each sequel there is only one original movie. Thus, the relationship is many-one from *Sequel* movies to *Original* movies, as indicated by the arrow in the E/R diagram of Fig. 4.5. □

Example 4.7: As a final example that includes both a multiway relationship and an entity set with multiple roles, in Fig. 4.6 is a more complex version of the *Contracts* relationship introduced earlier in Example 4.5. Now, relationship *Contracts* involves two studios, a star, and a movie. The intent is that one studio, having a certain star under contract (in general, not for a particular movie), may further contract with a second studio to allow that star to act in a particular movie. Thus, the relationship is described by 4-tuples of the form (studio1, studio2, star, movie), meaning that studio2 contracts with studio1 for the use of studio1's star by studio2 for the movie.

We see in Fig. 4.6 arrows pointing to *Studios* in both of its roles, as "owner" of the star and as producer of the movie. However, there are not arrows pointing to *Stars* or *Movies*. The rationale is as follows. Given a star, a movie, and a studio producing the movie, there can be only one studio that "owns" the star. (We assume a star is under contract to exactly one studio.) Similarly, only one studio produces a given movie, so given a star, a movie, and the star's studio, we can determine a unique producing studio. Note that in both

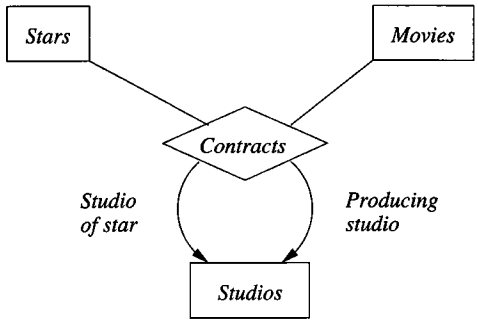

Figure 4.6: A four-way relationship

cases we actually needed only one of the other entities to determine the unique entity—for example, we need only know the movie to determine the unique producing studio—but this fact does not change the multiplicity specification for the multiway relationship.

There are no arrows pointing to *Stars* or *Movies*. Given a star, the star's studio, and a producing studio, there could be several different contracts allowing the star to act in several movies. Thus, the other three components in a relationship 4-tuple do not necessarily determine a unique movie. Similarly, a producing studio might contract with some other studio to use more than one of their stars in one movie. Thus, a star is not determined by the three other components of the relationship. □

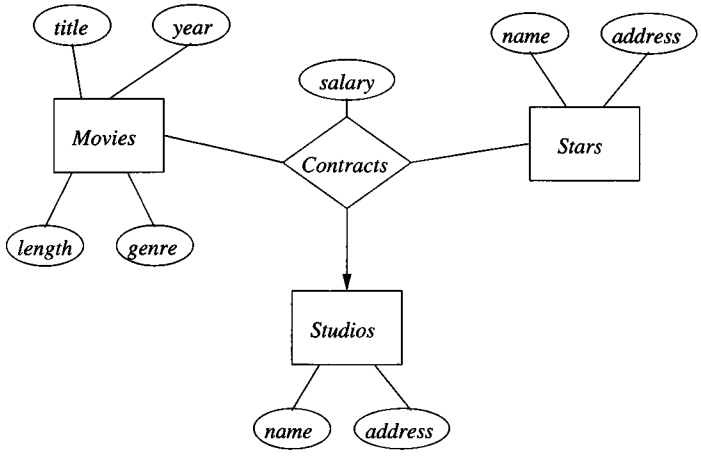

Figure 4.7: A relationship with an attribute

4.1.9 Attributes on Relationships

Sometimes it is convenient, or even essential, to associate attributes with a relationship, rather than with any one of the entity sets that the relationship connects. For example, consider the relationship of Fig. 4.4, which represents contracts between a star and studio for a movie.[1] We might wish to record the salary associated with this contract. However, we cannot associate it with the star; a star might get different salaries for different movies. Similarly, it does not make sense to associate the salary with a studio (they may pay different salaries to different stars) or with a movie (different stars in a movie may receive different salaries).

However, we can associate a unique salary with the (star, movie, studio) triple in the relationship set for the *Contracts* relationship. In Fig. 4.7 we see Fig. 4.4 fleshed out with attributes. The relationship has attribute *salary*, while the entity sets have the same attributes that we showed for them in Fig. 4.2.

In general, we may place one or more attributes on any relationship. The values of these attributes are functionally determined by the entire tuple in the relationship set for that relation. In some cases, the attributes can be determined by a subset of the entity sets involved in the relation, but presumably not by any single entity set (or it would make more sense to place the attribute on that entity set). For instance, in Fig. 4.7, the salary is really determined by the movie and star entities, since the studio entity is itself determined by the movie entity.

It is never necessary to place attributes on relationships. We can instead invent a new entity set, whose entities have the attributes ascribed to the relationship. If we then include this entity set in the relationship, we can omit the attributes on the relationship itself. However, attributes on a relationship are a useful convention, which we shall continue to use where appropriate.

Example 4.8: Let us revise the E/R diagram of Fig. 4.7, which has the salary attribute on the *Contracts* relationship. Instead, we create an entity set *Salaries*, with attribute *salary*. *Salaries* becomes the fourth entity set of relationship *Contracts*. The whole diagram is shown in Fig. 4.8.

Notice that there is an arrow into the *Salaries* entity set in Fig. 4.8. That arrow is appropriate, since we know that the salary is determined by all the other entity sets involved in the relationship. In general, when we do a conversion from attributes on a relationship to an additional entity set, we place an arrow into that entity set. □

4.1.10 Converting Multiway Relationships to Binary

There are some data models, such as UML (Section 4.7) and ODL (Section 4.9), that limit relationships to be binary. Thus, while the E/R model does not

[1] Here, we have reverted to the earlier notion of three-way contracts in Example 4.5, not the four-way relationship of Example 4.7.

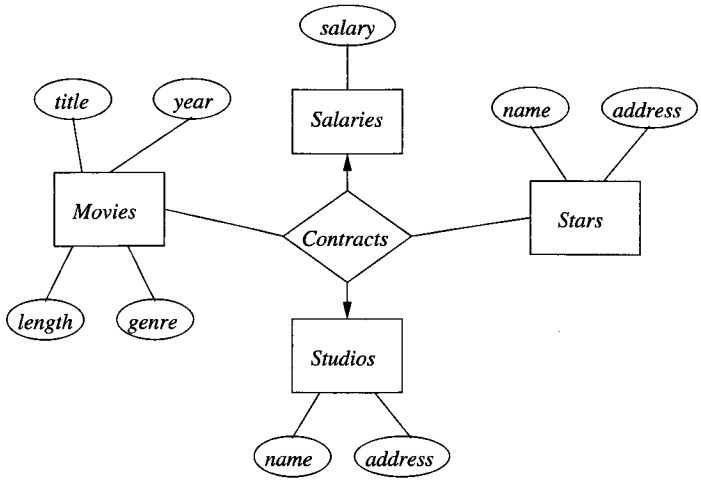

Figure 4.8: Moving the attribute to an entity set

require binary relationships, it is useful to observe that any relationship connecting more than two entity sets can be converted to a collection of binary, many-one relationships. To do so, introduce a new entity set whose entities we may think of as tuples of the relationship set for the multiway relationship. We call this entity set a *connecting* entity set. We then introduce many-one relationships from the connecting entity set to each of the entity sets that provide components of tuples in the original, multiway relationship. If an entity set plays more than one role, then it is the target of one relationship for each role.

Example 4.9: The four-way *Contracts* relationship in Fig. 4.6 can be replaced by an entity set that we may also call *Contracts*. As seen in Fig. 4.9, it participates in four relationships. If the relationship set for the relationship *Contracts* has a 4-tuple (studio1, studio2, star, movie) then the entity set *Contracts* has an entity e. This entity is linked by relationship *Star-of* to the entity *star* in entity set *Stars*. It is linked by relationship *Movie-of* to the entity *movie* in *Movies*. It is linked to entities *studio1* and *studio2* of *Studios* by relationships *Studio-of-star* and *Producing-studio*, respectively.

Note that we have assumed there are no attributes of entity set *Contracts*, although the other entity sets in Fig. 4.9 have unseen attributes. However, it is possible to add attributes, such as the date of signing, to entity set *Contracts*. □

4.1.11 Subclasses in the E/R Model

Often, an entity set contains certain entities that have special properties not associated with all members of the set. If so, we find it useful to define certain

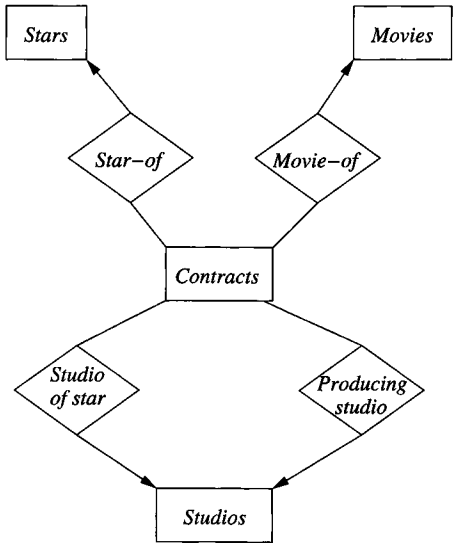

Figure 4.9: Replacing a multiway relationship by an entity set and binary relationships

special-case entity sets, or *subclasses*, each with its own special attributes and/or relationships. We connect an entity set to its subclasses using a relationship called *isa* (i.e., "an A is a B" expresses an "isa" relationship from entity set A to entity set B).

An isa relationship is a special kind of relationship, and to emphasize that it is unlike other relationships, we use a special notation: a triangle. One side of the triangle is attached to the subclass, and the opposite point is connected to the superclass. Every isa relationship is one-one, although we shall not draw the two arrows that are associated with other one-one relationships.

Example 4.10: Among the special kinds of movies we might store in our example database are cartoons and murder mysteries. For each of these special movie types, we could define a subclass of the entity set *Movies*. For instance, let us postulate two subclasses: *Cartoons* and *Murder-Mysteries*. A cartoon has, in addition to the attributes and relationships of *Movies*, an additional relationship called *Voices* that gives us a set of stars who speak, but do not appear in the movie. Movies that are not cartoons do not have such stars. Murder-mysteries have an additional attribute *weapon*. The connections among the three entity sets *Movies*, *Cartoons*, and *Murder-Mysteries* is shown in Fig. 4.10. □

While, in principle, a collection of entity sets connected by *isa* relationships could have any structure, we shall limit isa-structures to trees, in which there

Parallel Relationships Can Be Different

Figure 4.9 illustrates a subtle point about relationships. There are two different relationships, *Studio-of-Star* and *Producing-Studio*, that each connect entity sets *Contracts* and *Studios*. We should not presume that these relationships therefore have the same relationship sets. In fact, in this case, it is unlikely that both relationships would ever relate the same contract to the same studios, since a studio would then be contracting with itself.

More generally, there is nothing wrong with an E/R diagram having several relationships that connect the same entity sets. In the database, the instances of these relationships will normally be different, reflecting the different meanings of the relationships.

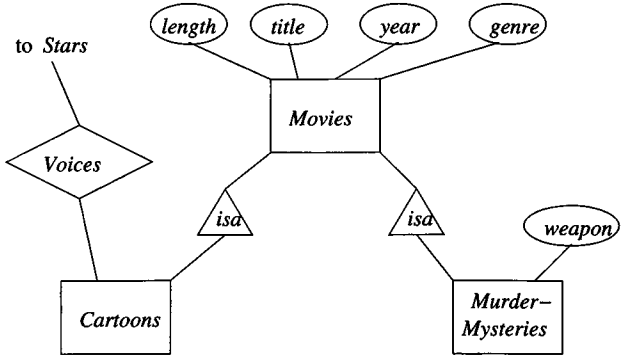

Figure 4.10: Isa relationships in an E/R diagram

is one *root* entity set (e.g., *Movies* in Fig. 4.10) that is the most general, with progressively more specialized entity sets extending below the root in a tree.

Suppose we have a tree of entity sets, connected by *isa* relationships. A single entity consists of *components* from one or more of these entity sets, as long as those components are in a subtree including the root. That is, if an entity e has a component c in entity set E, and the parent of E in the tree is F, then entity e also has a component d in F. Further, c and d must be paired in the relationship set for the *isa* relationship from E to F. The entity e has whatever attributes any of its components has, and it participates in whatever relationships any of its components participate in.

Example 4.11: The typical movie, being neither a cartoon nor a murder-mystery, will have a component only in the root entity set *Movies* in Fig. 4.10. These entities have only the four attributes of *Movies* (and the two relationships

The E/R View of Subclasses

There is a significant resemblance between "isa" in the E/R model and subclasses in object-oriented languages. In a sense, "isa" relates a subclass to its superclass. However, there is also a fundamental difference between the conventional E/R view and the object-oriented approach: entities are allowed to have representatives in a tree of entity sets, while objects are assumed to exist in exactly one class or subclass.

The difference becomes apparent when we consider how the movie *Roger Rabbit* was handled in Example 4.11. In an object-oriented approach, we would need for this movie a fourth entity set, "cartoon-murder-mystery," which inherited all the attributes and relationships of *Movies*, *Cartoons*, and *Murder-Mysteries*. However, in the E/R model, the effect of this fourth subclass is obtained by putting components of the movie *Roger Rabbit* in both the *Cartoons* and *Murder-Mysteries* entity sets.

of *Movies* — *Stars-in* and *Owns* — that are not shown in Fig. 4.10).

A cartoon that is not a murder-mystery will have two components, one in *Movies* and one in *Cartoons*. Its entity will therefore have not only the four attributes of *Movies*, but the relationship *Voices*. Likewise, a murder-mystery will have two components for its entity, one in *Movies* and one in *Murder-Mysteries* and thus will have five attributes, including *weapon*.

Finally, a movie like *Roger Rabbit*, which is both a cartoon and a murder-mystery, will have components in all three of the entity sets *Movies*, *Cartoons*, and *Murder-Mysteries*. The three components are connected into one entity by the *isa* relationships. Together, these components give the *Roger Rabbit* entity all four attributes of *Movies* plus the attribute *weapon* of entity set *Murder-Mysteries* and the relationship *Voices* of entity set *Cartoons*. □

4.1.12 Exercises for Section 4.1

Exercise 4.1.1: Design a database for a bank, including information about customers and their accounts. Information about a customer includes their name, address, phone, and Social Security number. Accounts have numbers, types (e.g., savings, checking) and balances. Also record the customer(s) who own an account. Draw the E/R diagram for this database. Be sure to include arrows where appropriate, to indicate the multiplicity of a relationship.

Exercise 4.1.2: Modify your solution to Exercise 4.1.1 as follows:

a) Change your diagram so an account can have only one customer.

b) Further change your diagram so a customer can have only one account.

! c) Change your original diagram of Exercise 4.1.1 so that a customer can have a set of addresses (which are street-city-state triples) and a set of phones. Remember that we do not allow attributes to have nonprimitive types, such as sets, in the E/R model.

! d) Further modify your diagram so that customers can have a set of addresses, and at each address there is a set of phones.

Exercise 4.1.3: Give an E/R diagram for a database recording information about teams, players, and their fans, including:

1. For each team, its name, its players, its team captain (one of its players), and the colors of its uniform.
2. For each player, his/her name.
3. For each fan, his/her name, favorite teams, favorite players, and favorite color.

Remember that a set of colors is not a suitable attribute type for teams. How can you get around this restriction?

Exercise 4.1.4: Suppose we wish to add to the schema of Exercise 4.1.3 a relationship *Led-by* among two players and a team. The intention is that this relationship set consists of triples (player1, player2, team) such that player 1 played on the team at a time when some other player 2 was the team captain.

a) Draw the modification to the E/R diagram.

b) Replace your ternary relationship with a new entity set and binary relationships.

! c) Are your new binary relationships the same as any of the previously existing relationships? Note that we assume the two players are different, i.e., the team captain is not self-led.

Exercise 4.1.5: Modify Exercise 4.1.3 to record for each player the history of teams on which they have played, including the start date and ending date (if they were traded) for each such team.

! **Exercise 4.1.6:** Design a genealogy database with one entity set: *People*. The information to record about persons includes their name (an attribute), their mother, father, and children.

! **Exercise 4.1.7:** Modify your "people" database design of Exercise 4.1.6 to include the following special types of people:

1. Females.

2. Males.

3. People who are parents.

You may wish to distinguish certain other kinds of people as well, so relationships connect appropriate subclasses of people.

Exercise 4.1.8: An alternative way to represent the information of Exercise 4.1.6 is to have a ternary relationship *Family* with the intent that in the relationship set for *Family*, triple (person, mother, father) is a person, their mother, and their father; all three are in the *People* entity set, of course.

a) Draw this diagram, placing arrows on edges where appropriate.

b) Replace the ternary relationship *Family* by an entity set and binary relationships. Again place arrows to indicate the multiplicity of relationships.

Exercise 4.1.9: Design a database suitable for a university registrar. This database should include information about students, departments, professors, courses, which students are enrolled in which courses, which professors are teaching which courses, student grades, TA's for a course (TA's are students), which courses a department offers, and any other information you deem appropriate. Note that this question is more free-form than the questions above, and you need to make some decisions about multiplicities of relationships, appropriate types, and even what information needs to be represented.

! **Exercise 4.1.10:** Informally, we can say that two E/R diagrams "have the same information" if, given a real-world situation, the instances of these two diagrams that reflect this situation can be computed from one another. Consider the E/R diagram of Fig. 4.6. This four-way relationship can be decomposed into a three-way relationship and a binary relationship by taking advantage of the fact that for each movie, there is a unique studio that produces that movie. Give an E/R diagram without a four-way relationship that has the same information as Fig. 4.6.

4.2 Design Principles

We have yet to learn many of the details of the E/R model, but we have enough to begin study of the crucial issue of what constitutes a good design and what should be avoided. In this section, we offer some useful design principles.

4.2.1 Faithfulness

First and foremost, the design should be faithful to the specifications of the application. That is, entity sets and their attributes should reflect reality. You can't attach an attribute *number-of-cylinders* to *Stars*, although that attribute

would make sense for an entity set *Automobiles*. Whatever relationships are asserted should make sense given what we know about the part of the real world being modeled.

Example 4.12: If we define a relationship *Stars-in* between *Stars* and *Movies*, it should be a many-many relationship. The reason is that an observation of the real world tells us that stars can appear in more than one movie, and movies can have more than one star. It is incorrect to declare the relationship *Stars-in* to be many-one in either direction or to be one-one. □

Example 4.13: On the other hand, sometimes it is less obvious what the real world requires us to do in our E/R design. Consider, for instance, entity sets *Courses* and *Instructors*, with a relationship *Teaches* between them. Is *Teaches* many-one from *Courses* to *Instructors*? The answer lies in the policy and intentions of the organization creating the database. It is possible that the school has a policy that there can be only one instructor for any course. Even if several instructors may "team-teach" a course, the school may require that exactly one of them be listed in the database as the instructor responsible for the course. In either of these cases, we would make *Teaches* a many-one relationship from *Courses* to *Instructors*.

Alternatively, the school may use teams of instructors regularly and wish its database to allow several instructors to be associated with a course. Or, the intent of the *Teaches* relationship may not be to reflect the current teacher of a course, but rather those who have ever taught the course, or those who are capable of teaching the course; we cannot tell simply from the name of the relationship. In either of these cases, it would be proper to make *Teaches* be many-many. □

4.2.2 Avoiding Redundancy

We should be careful to say everything once only. The problems we discussed in Section 3.3 regarding redundancy and anomalies are typical of problems that can arise in E/R designs. However, in the E/R model, there are several new mechanisms whereby redundancy and other anomalies can arise.

For instance, we have used a relationship *Owns* between movies and studios. We might also choose to have an attribute *studioName* of entity set *Movies*. While there is nothing illegal about doing so, it is dangerous for several reasons.

1. Doing so leads to repetition of a fact, with the result that extra space is required to represent the data, once we convert the E/R design to a relational (or other type of) concrete implementation.

2. There is an update-anomaly potential, since we might change the relationship but not the attribute, or vice-versa.

We shall say more about avoiding anomalies in Sections 4.2.4 and 4.2.5.

4.2.3 Simplicity Counts

Avoid introducing more elements into your design than is absolutely necessary.

Example 4.14: Suppose that instead of a relationship between *Movies* and *Studios* we postulated the existence of "movie-holdings," the ownership of a single movie. We might then create another entity set *Holdings*. A one-one relationship *Represents* could be established between each movie and the unique holding that represents the movie. A many-one relationship from *Holdings* to *Studios* completes the picture shown in Fig. 4.11.

Figure 4.11: A poor design with an unnecessary entity set

Technically, the structure of Fig. 4.11 truly represents the real world, since it is possible to go from a movie to its unique owning studio via *Holdings*. However, *Holdings* serves no useful purpose, and we are better off without it. It makes programs that use the movie-studio relationship more complicated, wastes space, and encourages errors. □

4.2.4 Choosing the Right Relationships

Entity sets can be connected in various ways by relationships. However, adding to our design every possible relationship is not often a good idea. Doing so can lead to redundancy, update anomalies, and deletion anomalies, where the connected pairs or sets of entities for one relationship can be deduced from one or more other relationships. We shall illustrate the problem and what to do about it with two examples. In the first example, several relationships could represent the same information; in the second, one relationship could be deduced from several others.

Example 4.15: Let us review Fig. 4.7, where we connected movies, stars, and studios with a three-way relationship *Contracts*. We omitted from that figure the two binary relationships *Stars-in* and *Owns* from Fig. 4.2. Do we also need these relationships, between *Movies* and *Stars*, and between *Movies* and *Studios*, respectively? The answer is: "we don't know; it depends on our assumptions regarding the three relationships in question."

It might be possible to deduce the relationship *Stars-in* from *Contracts*. If a star can appear in a movie only if there is a contract involving that star, that movie, and the owning studio for the movie, then there truly is no need for relationship *Stars-in*. We could figure out all the star-movie pairs by looking at the star-movie-studio triples in the relationship set for *Contracts* and taking only the star and movie components, i.e., projecting *Contracts* onto *Stars-in*.

However, if a star can work on a movie without there being a contract — or what is more likely, without there being a contract that we know about in our database — then there could be star-movie pairs in *Stars-in* that are not part of star-movie-studio triples in *Contracts*. In that case, we need to retain the *Stars-in* relationship.

A similar observation applies to relationship *Owns*. If for every movie, there is at least one contract involving that movie, its owning studio, and some star for that movie, then we can dispense with *Owns*. However, if there is the possibility that a studio owns a movie, yet has no stars under contract for that movie, or no such contract is known to our database, then we must retain *Owns*.

In summary, we cannot tell you whether a given relationship will be redundant. You must find out from those who wish the database implemented what to expect. Only then can you make a rational decision about whether or not to include relationships such as *Stars-in* or *Owns*. □

Example 4.16: Now, consider Fig. 4.2 again. In this diagram, there is no relationship between stars and studios. Yet we can use the two relationships *Stars-in* and *Owns* to build a connection by the process of composing those two relationships. That is, a star is connected to some movies by *Stars-in*, and those movies are connected to studios by *Owns*. Thus, we could say that a star is connected to the studios that own movies in which the star has appeared.

Would it make sense to have a relationship *Works-for*, as suggested in Fig. 4.12, between *Stars* and *Studios* too? Again, we cannot tell without knowing more. First, what would the meaning of this relationship be? If it is to mean "the star appeared in at least one movie of this studio," then probably there is no good reason to include it in the diagram. We could deduce this information from *Stars-in* and *Owns* instead.

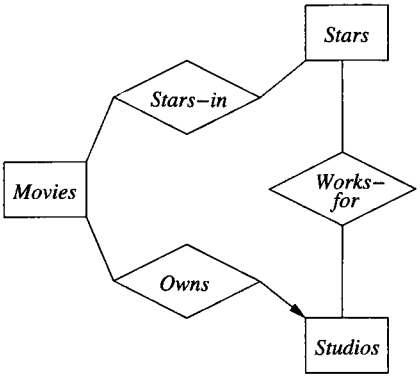

Figure 4.12: Adding a relationship between *Stars* and *Studios*

However, perhaps we have other information about stars working for studios that is not implied by the connection through a movie. In that case, a

relationship connecting stars directly to studios might be useful and would not be redundant. Alternatively, we might use a relationship between stars and studios to mean something entirely different. For example, it might represent the fact that the star is under contract to the studio, in a manner unrelated to any movie. As we suggested in Example 4.7, it is possible for a star to be under contract to one studio and yet work on a movie owned by another studio. In this case, the information found in the new *Works-for* relation would be independent of the *Stars-in* and *Owns* relationships, and would surely be nonredundant. □

4.2.5 Picking the Right Kind of Element

Sometimes we have options regarding the type of design element used to represent a real-world concept. Many of these choices are between using attributes and using entity set/relationship combinations. In general, an attribute is simpler to implement than either an entity set or a relationship. However, making everything an attribute will usually get us into trouble.

Example 4.17: Let us consider a specific problem. In Fig. 4.2, were we wise to make studios an entity set? Should we instead have made the name and address of the studio be attributes of movies and eliminated the *Studio* entity set? One problem with doing so is that we repeat the address of the studio for each movie. We can also have an update anomaly if we change the address for one movie but not another with the same studio, and we can have a deletion anomaly if we delete the last movie owned by a given studio.

On the other hand, if we did not record addresses of studios, then there is no harm in making the studio name an attribute of movies. We have no anomalies in this case. Saying the name of a studio for each movie is not true redundancy, since we must represent the owner of each movie somehow, and saying the name of the studio is a reasonable way to do so. □

We can abstract what we have observed in Example 4.17 to give the conditions under which we prefer to use an attribute instead of an entity set. Suppose E is an entity set. Here are conditions that E must obey in order for us to replace E by an attribute or attributes of several other entity sets.

1. All relationships in which E is involved must have arrows entering E. That is, E must be the "one" in many-one relationships, or its generalization for the case of multiway relationships.

2. If E has more than one attribute, then no attribute depends on the other attributes, the way *address* depends on *name* for *Studios*. That is, the only key for E is all its attributes.

3. No relationship involves E more than once.

If these conditions are met, then we can replace entity set E as follows:

a) If there is a many-one relationship R from some entity set F to E, then remove R and make the attributes of E be attributes of F, suitably renamed if they conflict with attribute names for F. In effect, each F-entity takes, as attributes, the name of the unique, related E-entity.[2] For instance, *Movies* entities could take their studio name as an attribute, should we dispense with studio addresses.

b) If there is a multiway relationship R with an arrow to E, make the attributes of E be attributes of R and delete the arc from R to E. An example of this transformation is replacing Fig. 4.8, where there is an entity set *Salaries* with a number as its lone attribute, by its original diagram in Fig. 4.7.

Example 4.18: Let us consider a point where there is a tradeoff between using a multiway relationship and using a connecting entity set with several binary relationships. We saw a four-way relationship *Contracts* among a star, a movie, and two studios in Fig. 4.6. In Fig. 4.9, we mechanically converted it to an entity set *Contracts*. Does it matter which we choose?

As the problem was stated, either is appropriate. However, should we change the problem just slightly, then we are almost forced to choose a connecting entity set. Let us suppose that contracts involve one star, one movie, but any set of studios. This situation is more complex than the one in Fig. 4.6, where we had two studios playing two roles. In this case, we can have any number of studios involved, perhaps one to do production, one for special effects, one for distribution, and so on. Thus, we cannot assign roles for studios.

It appears that a relationship set for the relationship *Contracts* must contain triples of the form (star, movie, set-of-studios), and the relationship *Contracts* itself involves not only the usual *Stars* and *Movies* entity sets, but a new entity set whose entities are *sets of* studios. While this approach is possible, it seems unnatural to think of sets of studios as basic entities, and we do not recommend it.

A better approach is to think of contracts as an entity set. As in Fig. 4.9, a contract entity connects a star, a movie and a set of studios, but now there must be no limit on the number of studios. Thus, the relationship between contracts and studios is many-many, rather than many-one as it would be if contracts were a true "connecting" entity set. Figure 4.13 sketches the E/R diagram. Note that a contract is related to a single star and to a single movie, but to any number of studios. □

4.2.6 Exercises for Section 4.2

Exercise 4.2.1: In Fig. 4.14 is an E/R diagram for a bank database involving customers and accounts. Since customers may have several accounts, and

[2] In a situation where an F-entity is not related to any E-entity, the new attributes of F would be given special "null" values to indicate the absence of a related E-entity. A similar arrangement would be used for the new attributes of R in case (b).

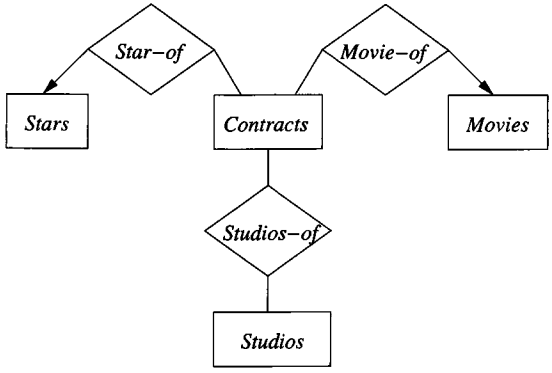

Figure 4.13: Contracts connecting a star, a movie, and a set of studios

accounts may be held jointly by several customers, we associate with each customer an "account set," and accounts are members of one or more account sets. Assuming the meaning of the various relationships and attributes are as expected given their names, criticize the design. What design rules are violated? Why? What modifications would you suggest?

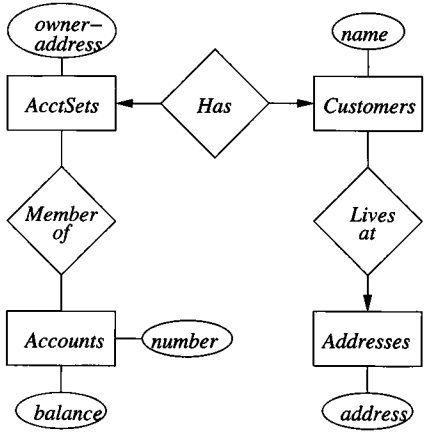

Figure 4.14: A poor design for a bank database

Exercise 4.2.2: Under what circumstances (regarding the unseen attributes of *Studios* and *Presidents*) would you recommend combining the two entity sets and relationship in Fig. 4.3 into a single entity set and attributes?

Exercise 4.2.3: Suppose we delete the attribute *address* from *Studios* in Fig. 4.7. Show how we could then replace an entity set by an attribute. Where

would that attribute appear?

Exercise 4.2.4: Give choices of attributes for the following entity sets in Fig. 4.13 that will allow the entity set to be replaced by an attribute:

a) *Stars.*

b) *Movies.*

! c) *Studios.*

!! **Exercise 4.2.5:** In this and following exercises we shall consider two design options in the E/R model for describing births. At a birth, there is one baby (twins would be represented by two births), one mother, any number of nurses, and any number of doctors. Suppose, therefore, that we have entity sets *Babies*, *Mothers*, *Nurses*, and *Doctors*. Suppose we also use a relationship *Births*, which connects these four entity sets, as suggested in Fig. 4.15. Note that a tuple of the relationship set for *Births* has the form (baby, mother, nurse, doctor). If there is more than one nurse and/or doctor attending a birth, then there will be several tuples with the same baby and mother, one for each combination of nurse and doctor.

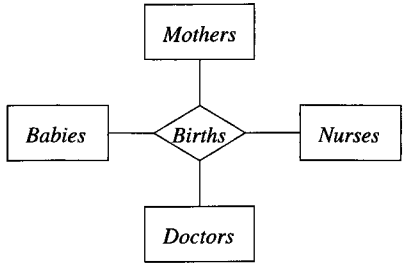

Figure 4.15: Representing births by a multiway relationship

There are certain assumptions that we might wish to incorporate into our design. For each, tell how to add arrows or other elements to the E/R diagram in order to express the assumption.

a) For every baby, there is a unique mother.

b) For every combination of a baby, nurse, and doctor, there is a unique mother.

c) For every combination of a baby and a mother there is a unique doctor.

! **Exercise 4.2.6:** Another approach to the problem of Exercise 4.2.5 is to connect the four entity sets *Babies*, *Mothers*, *Nurses*, and *Doctors* by an entity set

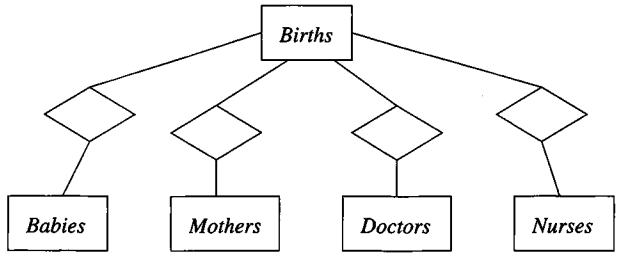

Figure 4.16: Representing births by an entity set

Births, with four relationships, one between *Births* and each of the other entity sets, as suggested in Fig. 4.16. Use arrows (indicating that certain of these relationships are many-one) to represent the following conditions:

a) Every baby is the result of a unique birth, and every birth is of a unique baby.

b) In addition to (a), every baby has a unique mother.

c) In addition to (a) and (b), for every birth there is a unique doctor.

In each case, what design flaws do you see?

!! **Exercise 4.2.7:** Suppose we change our viewpoint to allow a birth to involve more than one baby born to one mother. How would you represent the fact that every baby still has a unique mother using the approaches of Exercises 4.2.5 and 4.2.6?

4.3 Constraints in the E/R Model

The E/R model has several ways to express the common kinds of constraints on the data that will populate the database being designed. Like the relational model, there is a way to express the idea that an attribute or attributes are a key for an entity set. We have already seen how an arrow connecting a relationship to an entity set serves as a "functional dependency." There is also a way to express a referential-integrity constraint, where an entity in one set is required to have an entity in another set to which it is related.

4.3.1 Keys in the E/R Model

A *key* for an entity set E is a set K of one or more attributes such that, given any two distinct entities e_1 and e_2 in E, e_1 and e_2 cannot have identical values for each of the attributes in the key K. If K consists of more than one attribute, then it is possible for e_1 and e_2 to agree in some of these attributes, but never in all attributes. Some important points to remember are:

- Every entity set must have a key, although in some cases — isa-hierarchies and "weak" entity sets (see Section 4.4), the key actually belongs to another entity set.

- There can be more than one possible key for an entity set. However, it is customary to pick one key as the "primary key," and to act as if that were the only key.

- When an entity set is involved in an isa-hierarchy, we require that the root entity set have all the attributes needed for a key, and that the key for each entity is found from its component in the root entity set, regardless of how many entity sets in the hierarchy have components for the entity.

In our running movies example, we have used *title* and *year* as the key for *Movies*, counting on the observation that it is unlikely that two movies with the same title would be released in one year. We also decided that it was safe to use *name* as a key for *MovieStar*, believing that no real star would ever want to use the name of another star.

4.3.2 Representing Keys in the E/R Model

In our E/R-diagram notation, we underline the attributes belonging to a key for an entity set. For example, Fig. 4.17 reproduces our E/R diagram for movies, stars, and studios from Fig. 4.2, but with key attributes underlined. Attribute *name* is the key for *Stars*. Likewise, *Studios* has a key consisting of only its own attribute *name*.

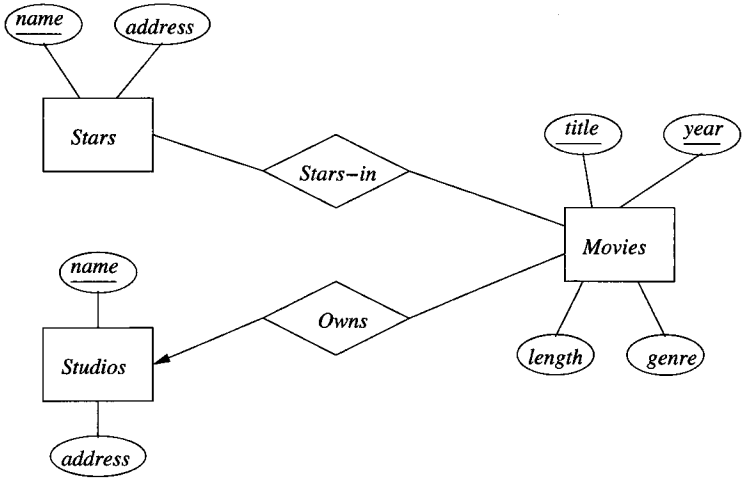

Figure 4.17: E/R diagram; keys are indicated by underlines

The attributes *title* and *year* together form the key for *Movies*. Note that when several attributes are underlined, as in Fig. 4.17, then they are each members of the key. There is no notation for representing the situation where there are several keys for an entity set; we underline only the primary key. You should also be aware that in some unusual situations, the attributes forming the key for an entity set do not all belong to the entity set itself. We shall defer this matter, called "weak entity sets," until Section 4.4.

4.3.3 Referential Integrity

Recall our discussion of referential-integrity constraints in Section 2.5.2. These constraints say that a value appearing in one context must also appear in another. For example, let us consider the many-one relationship *Owns* from *Movies* to *Studios* in Fig. 4.2. The many-one requirement simply says that no movie can be owned by more than one studio. It does *not* say that a movie must surely be owned by a studio, or that the owning studio must be present in the *Studios* entity set, as stored in our database. An appropriate referential integrity constraint on relationship *Owns* is that for each movie, the owning studio (the entity "referenced" by the relationship for this movie) must exist in our database.

The arrow notation in E/R diagrams is able to indicate whether a relationship is expected to support referential integrity in one or more directions. Suppose R is a relationship from entity set E to entity set F. A rounded arrowhead pointing to F indicates not only that the relationship is many-one from E to F, but that the entity of set F related to a given entity of set E is required to exist. The same idea applies when R is a relationship among more than two entity sets.

Example 4.19: Figure 4.18 shows some appropriate referential integrity constraints among the entity sets *Movies*, *Studios*, and *Presidents*. These entity sets and relationships were first introduced in Figs. 4.2 and 4.3. We see a rounded arrow entering *Studios* from relationship *Owns*. That arrow expresses the referential integrity constraint that every movie must be owned by one studio, and this studio is present in the *Studios* entity set.

Figure 4.18: E/R diagram showing referential integrity constraints

Similarly, we see a rounded arrow entering *Studios* from *Runs*. That arrow expresses the referential integrity constraint that every president runs a studio that exists in the *Studios* entity set.

Note that the arrow to *Presidents* from *Runs* remains a pointed arrow. That choice reflects a reasonable assumption about the relationship between studios

and their presidents. If a studio ceases to exist, its president can no longer be called a president, so we would expect the president of the studio to be deleted from the entity set *Presidents*. Hence there is a rounded arrow to *Studios*. On the other hand, if a president were fired or resigned, the studio would continue to exist. Thus, we place an ordinary, pointed arrow to *Presidents*, indicating that each studio has at most one president, but might have no president at some time. □

4.3.4 Degree Constraints

In the E/R model, we can attach a bounding number to the edges that connect a relationship to an entity set, indicating limits on the number of entities that can be connected to any one entity of the related entity set. For example, we could choose to place a constraint on the degree of a relationship, such as that a movie entity cannot be connected by relationship *Stars-in* to more than 10 star entities.

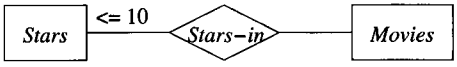

Figure 4.19: Representing a constraint on the number of stars per movie

Figure 4.19 shows how we can represent this constraint. As another example, we can think of the arrow as a synonym for the constraint "≤ 1," and we can think of the rounded arrow of Fig. 4.18 as standing for the constraint "$= 1$."

4.3.5 Exercises for Section 4.3

Exercise 4.3.1: For your E/R diagrams of:

a) Exercise 4.1.1.

b) Exercise 4.1.3.

c) Exercise 4.1.6.

(*i*) Select and specify keys, and (*ii*) Indicate appropriate referential integrity constraints.

! **Exercise 4.3.2:** We may think of relationships in the E/R model as having keys, just as entity sets do. Let R be a relationship among the entity sets $E_1, E_2, \ldots, E_n$. Then a *key* for R is a set K of attributes chosen from the attributes of $E_1, E_2, \ldots, E_n$ such that if $(e_1, e_2, \ldots, e_n)$ and $(f_1, f_2, \ldots, f_n)$ are two different tuples in the relationship set for R, then it is not possible that these tuples agree in all the attributes of K. Now, suppose $n = 2$; that is, R is a binary relationship. Also, for each i, let K_i be a set of attributes that is a key for entity set E_i. In terms of E_1 and E_2, give a smallest possible key for R under the assumption that:

a) R is many-many.

b) R is many-one from E_1 to E_2.

c) R is many-one from E_2 to E_1.

d) R is one-one.

!! **Exercise 4.3.3:** Consider again the problem of Exercise 4.3.2, but with n allowed to be any number, not just 2. Using only the information about which arcs from R to the E_i's have arrows, show how to find a smallest possible key K for R in terms of the K_i's.

4.4 Weak Entity Sets

It is possible for an entity set's key to be composed of attributes, some or all of which belong to another entity set. Such an entity set is called a *weak entity set.*

4.4.1 Causes of Weak Entity Sets

There are two principal reasons we need weak entity sets. First, sometimes entity sets fall into a hierarchy based on classifications unrelated to the "isa hierarchy" of Section 4.1.11. If entities of set E are subunits of entities in set F, then it is possible that the names of E-entities are not unique until we take into account the name of the F-entity to which the E entity is subordinate. Several examples will illustrate the problem.

Example 4.20: A movie studio might have several film crews. The crews might be designated by a given studio as crew 1, crew 2, and so on. However, other studios might use the same designations for crews, so the attribute *number* is not a key for crews. Rather, to name a crew uniquely, we need to give both the name of the studio to which it belongs and the number of the crew. The situation is suggested by Fig. 4.20. The double-rectangle indicates a weak entity set, and the double-diamond indicates a many-one relationship that helps provide the key for the weak entity set. The notation will be explained further in Section 4.4.3. The key for weak entity set *Crews* is its own *number* attribute and the *name* attribute of the unique studio to which the crew is related by the many-one *Unit-of* relationship. □

Example 4.21: A species is designated by its genus and species names. For example, humans are of the species *Homo sapiens*; *Homo* is the genus name and *sapiens* the species name. In general, a genus consists of several species, each of which has a name beginning with the genus name and continuing with the species name. Unfortunately, species names, by themselves, are not unique.

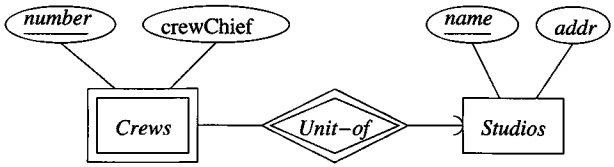

Figure 4.20: A weak entity set for crews, and its connections

Two or more genera may have species with the same species name. Thus, to designate a species uniquely we need both the species name and the name of the genus to which the species is related by the *Belongs-to* relationship, as suggested in Fig. 4.21. *Species* is a weak entity set whose key comes partially from its genus. □

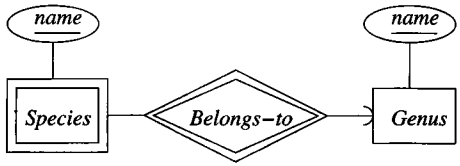

Figure 4.21: Another weak entity set, for species

The second common source of weak entity sets is the connecting entity sets that we introduced in Section 4.1.10 as a way to eliminate a multiway relationship.[3] These entity sets often have no attributes of their own. Their key is formed from the attributes that are the key attributes for the entity sets they connect.

Example 4.22: In Fig. 4.22 we see a connecting entity set *Contracts* that replaces the ternary relationship *Contracts* of Example 4.5. *Contracts* has an attribute *salary*, but this attribute does not contribute to the key. Rather, the key for a contract consists of the name of the studio and the star involved, plus the title and year of the movie involved. □

4.4.2 Requirements for Weak Entity Sets

We cannot obtain key attributes for a weak entity set indiscriminately. Rather, if E is a weak entity set then its key consists of:

1. Zero or more of its own attributes, and

[3]Remember that there is no particular requirement in the E/R model that multiway relationships be eliminated, although this requirement exists in some other database design models.

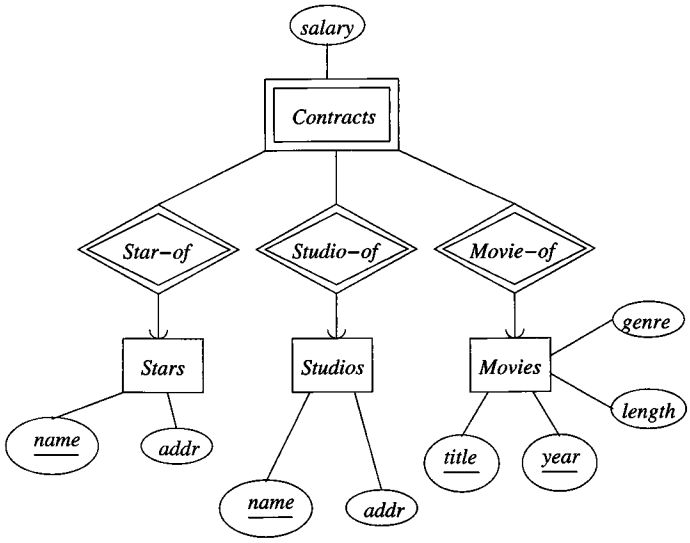

Figure 4.22: Connecting entity sets are weak

2. Key attributes from entity sets that are reached by certain many-one relationships from E to other entity sets. These many-one relationships are called *supporting relationships* for E, and the entity sets reached from E are *supporting entity sets.*

In order for R, a many-one relationship from E to some entity set F, to be a supporting relationship for E, the following conditions must be obeyed:

a) R must be a binary, many-one relationship[4] from E to F.

b) R must have referential integrity from E to F. That is, for every E-entity, there must be exactly one existing F-entity related to it by R. Put another way, a rounded arrow from R to F must be justified.

c) The attributes that F supplies for the key of E must be key attributes of F.

d) However, if F is itself weak, then some or all of the key attributes of F supplied to E will be key attributes of one or more entity sets G to which F is connected by a supporting relationship. Recursively, if G is weak, some key attributes of G will be supplied from elsewhere, and so on.

[4]Remember that a one-one relationship is a special case of a many-one relationship. When we say a relationship must be many-one, we always include one-one relationships as well.

e) If there are several different supporting relationships from E to the same entity set F, then each relationship is used to supply a copy of the key attributes of F to help form the key of E. Note that an entity e from E may be related to different entities in F through different supporting relationships from E. Thus, the keys of several different entities from F may appear in the key values identifying a particular entity e from E.

The intuitive reason why these conditions are needed is as follows. Consider an entity in a weak entity set, say a crew in Example 4.20. Each crew is unique, abstractly. In principle we can tell one crew from another, even if they have the same number but belong to different studios. It is only the data about crews that makes it hard to distinguish crews, because the number alone is not sufficient. The only way we can associate additional information with a crew is if there is some deterministic process leading to additional values that make the designation of a crew unique. But the only unique values associated with an abstract crew entity are:

1. Values of attributes of the *Crews* entity set, and
2. Values obtained by following a relationship from a crew entity to a unique entity of some other entity set, where that other entity has a unique associated value of some kind. That is, the relationship followed must be many-one to the other entity set F, and the associated value must be part of a key for F.

4.4.3 Weak Entity Set Notation

We shall adopt the following conventions to indicate that an entity set is weak and to declare its key attributes.

1. If an entity set is weak, it will be shown as a rectangle with a double border. Examples of this convention are *Crews* in Fig. 4.20 and *Contracts* in Fig. 4.22.
2. Its supporting many-one relationships will be shown as diamonds with a double border. Examples of this convention are *Unit-of* in Fig. 4.20 and all three relationships in Fig. 4.22.
3. If an entity set supplies any attributes for its own key, then those attributes will be underlined. An example is in Fig. 4.20, where the number of a crew participates in its own key, although it is not the complete key for *Crews*.

We can summarize these conventions with the following rule:

- Whenever we use an entity set E with a double border, it is weak. The key for E is whatever attributes of E are underlined plus the key attributes of those entity sets to which E is connected by many-one relationships with a double border.

We should remember that the double-diamond is used only for supporting relationships. It is possible for there to be many-one relationships from a weak entity set that are not supporting relationships, and therefore do not get a double diamond.

Example 4.23: In Fig. 4.22, the relationship *Studio-of* need not be a supporting relationship for *Contracts*. The reason is that each movie has a unique owning studio, determined by the (not shown) many-one relationship from *Movies* to *Studios*. Thus, if we are told the name of a star and a movie, there is at most one contract with any studio for the work of that star in that movie. In terms of our notation, it would be appropriate to use an ordinary single diamond, rather than the double diamond, for *Studio-of* in Fig. 4.22. □

4.4.4 Exercises for Section 4.4

Exercise 4.4.1: One way to represent students and the grades they get in courses is to use entity sets corresponding to students, to courses, and to "enrollments." Enrollment entities form a "connecting" entity set between students and courses and can be used to represent not only the fact that a student is taking a certain course, but the grade of the student in the course. Draw an E/R diagram for this situation, indicating weak entity sets and the keys for the entity sets. Is the grade part of the key for enrollments?

Exercise 4.4.2: Modify your solution to Exercise 4.4.1 so that we can record grades of the student for each of several assignments within a course. Again, indicate weak entity sets and keys.

Exercise 4.4.3: For your E/R diagrams of Exercise 4.2.6(a)–(c), indicate weak entity sets, supporting relationships, and keys.

Exercise 4.4.4: Draw E/R diagrams for the following situations involving weak entity sets. In each case indicate keys for entity sets.

a) Entity sets *Courses* and *Departments*. A course is given by a unique department, but its only attribute is its number. Different departments can offer courses with the same number. Each department has a unique name.

! b) Entity sets *Leagues*, *Teams*, and *Players*. League names are unique. No league has two teams with the same name. No team has two players with the same number. However, there can be players with the same number on different teams, and there can be teams with the same name in different leagues.

4.5 From E/R Diagrams to Relational Designs

To a first approximation, converting an E/R design to a relational database schema is straightforward:

- Turn each entity set into a relation with the same set of attributes, and
- Replace a relationship by a relation whose attributes are the keys for the connected entity sets.

While these two rules cover much of the ground, there are also several special situations that we need to deal with, including:

1. Weak entity sets cannot be translated straightforwardly to relations.
2. "Isa" relationships and subclasses require careful treatment.
3. Sometimes, we do well to combine two relations, especially the relation for an entity set E and the relation that comes from a many-one relationship from E to some other entity set.

4.5.1 From Entity Sets to Relations

Let us first consider entity sets that are not weak. We shall take up the modifications needed to accommodate weak entity sets in Section 4.5.4. For each non-weak entity set, we shall create a relation of the same name and with the same set of attributes. This relation will not have any indication of the relationships in which the entity set participates; we'll handle relationships with separate relations, as discussed in Section 4.5.2.

Example 4.24: Consider the three entity sets *Movies*, *Stars* and *Studios* from Fig. 4.17, which we reproduce here as Fig. 4.23. The attributes for the *Movies* entity set are *title*, *year*, *length*, and *genre*. As a result, this relation `Movies` looks just like the relation `Movies` of Fig. 2.1 with which we began Section 2.2.

Next, consider the entity set *Stars* from Fig. 4.23. There are two attributes, *name* and *address*. Thus, we would expect the corresponding `Stars` relation to have schema `Stars(name, address)` and for

name	*address*
`Carrie Fisher`	`123 Maple St., Hollywood`
`Mark Hamill`	`456 Oak Rd., Brentwood`
`Harrison Ford`	`789 Palm Dr., Beverly Hills`

to be a typical instance. □

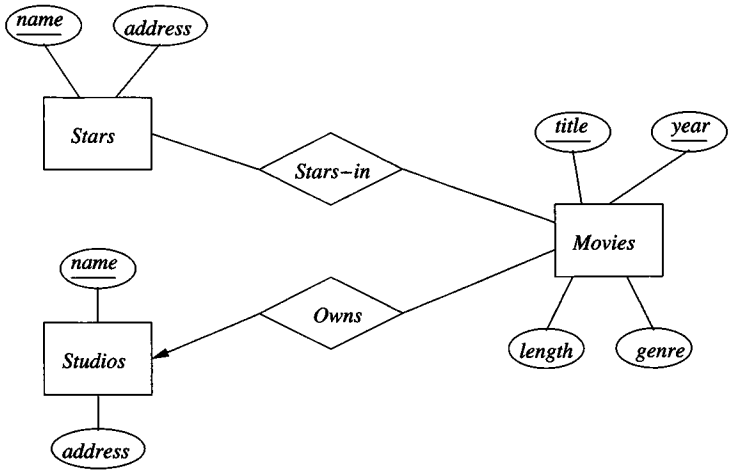

Figure 4.23: E/R diagram for the movie database

4.5.2 From E/R Relationships to Relations

Relationships in the E/R model are also represented by relations. The relation for a given relationship R has the following attributes:

1. For each entity set involved in relationship R, we take its key attribute or attributes as part of the schema of the relation for R.

2. If the relationship has attributes, then these are also attributes of relation R.

If one entity set is involved several times in a relationship, in different roles, then its key attributes each appear as many times as there are roles. We must rename the attributes to avoid name duplication. More generally, should the same attribute name appear twice or more among the attributes of R itself and the keys of the entity sets involved in relationship R, then we need to rename to avoid duplication.

Example 4.25: Consider the relationship *Owns* of Fig. 4.23. This relationship connects entity sets *Movies* and *Studios*. Thus, for the schema of relation `Owns` we use the key for *Movies*, which is *title* and *year*, and the key of *Studios*, which is *name*. That is, the schema for relation `Owns` is:

```
Owns(title, year, studioName)
```

A sample instance of this relation is:

title	*year*	*studioName*
Star Wars	1977	Fox
Gone With the Wind	1939	MGM
Wayne's World	1992	Paramount

We have chosen the attribute `studioName` for clarity; it corresponds to the attribute *name* of *Studios*. □

title	*year*	*starName*
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Gone With the Wind	1939	Vivien Leigh
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

Figure 4.24: A relation for relationship *Stars-In*

Example 4.26: Similarly, the relationship *Stars-In* of Fig. 4.23 can be transformed into a relation with the attributes `title` and `year` (the key for *Movies*) and attribute `starName`, which is the key for entity set *Stars*. Figure 4.24 shows a sample relation `Stars-In`. □

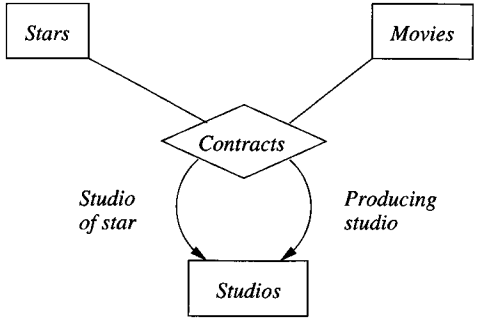

Figure 4.25: The relationship *Contracts*

Example 4.27: Multiway relationships are also easy to convert to relations. Consider the four-way relationship *Contracts* of Fig. 4.6, reproduced here as Fig. 4.25, involving a star, a movie, and two studios — the first holding the star's contract and the second contracting for that star's services in that movie. We represent this relationship by a relation `Contracts` whose schema consists of the attributes from the keys of the following four entity sets:

1. The key `starName` for the star.
2. The key consisting of attributes `title` and `year` for the movie.
3. The key `studioOfStar` indicating the name of the first studio; recall we assume the studio name is a key for the entity set `Studios`.
4. The key `producingStudio` indicating the name of the studio that will produce the movie using that star.

That is, the relation schema is:

```
Contracts(starName, title, year, studioOfStar, producingStudio)
```

Notice that we have been inventive in choosing attribute names for our relation schema, avoiding "name" for any attribute, since it would be unobvious whether that referred to a star's name or studio's name, and in the latter case, which studio role. Also, were there attributes attached to entity set *Contracts*, such as *salary*, these attributes would be added to the schema of relation `Contracts`. □

4.5.3 Combining Relations

Sometimes, the relations that we get from converting entity sets and relationships to relations are not the best possible choice of relations for the given data. One common situation occurs when there is an entity set E with a many-one relationship R from E to F. The relations from E and R will each have the key for E in their relation schema. In addition, the relation for E will have in its schema the attributes of E that are not in the key, and the relation for R will have the key attributes of F and any attributes of R itself. Because R is many-one, all these attributes are functionally determined by the key for E, and we can combine them into one relation with a schema consisting of:

1. All attributes of E.
2. The key attributes of F.
3. Any attributes belonging to relationship R.

For an entity e of E that is not related to any entity of F, the attributes of types (2) and (3) will have null values in the tuple for e.

Example 4.28: In our running movie example, *Owns* is a many-one relationship from *Movies* to *Studios*, which we converted to a relation in Example 4.25. The relation obtained from entity set *Movies* was discussed in Example 4.24. We can combine these relations by taking all their attributes and forming one relation schema. If we do, the relation looks like that in Fig. 4.26. □

title	*year*	*length*	*genre*	*studioName*
Star Wars	1977	124	sciFi	Fox
Gone With the Wind	1939	239	drama	MGM
Wayne's World	1992	95	comedy	Paramount

Figure 4.26: Combining relation Movies with relation Owns

Whether or not we choose to combine relations in this manner is a matter of judgement. However, there are some advantages to having all the attributes that are dependent on the key of entity set E together in one relation, even if there are a number of many-one relationships from E to other entity sets. For example, it is often more efficient to answer queries involving attributes of one relation than to answer queries involving attributes of several relations. In fact, some design systems based on the E/R model combine these relations automatically.

On the other hand, one might wonder if it made sense to combine the relation for E with the relation of a relationship R that involved E but was not many-one from E to some other entity set. Doing so is risky, because it often leads to redundancy, as the next example shows.

Example 4.29: To get a sense of what can go wrong, suppose we combined the relation of Fig. 4.26 with the relation that we get for the many-many relationship *Stars-in*; recall this relation was suggested by Fig. 4.24. Then the combined relation would look like Fig. 3.2, which we reproduce here as Fig. 4.27. As we discussed in Section 3.3.1, this relation has anomalies that we need to remove by the process of normalization. □

title	*year*	*length*	*genre*	*studioName*	*starName*
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Figure 4.27: The relation Movies with star information

4.5.4 Handling Weak Entity Sets

When a weak entity set appears in an E/R diagram, we need to do three things differently.

1. The relation for the weak entity set W itself must include not only the attributes of W but also the key attributes of the supporting entity sets. The supporting entity sets are easily recognized because they are reached by supporting (double-diamond) relationships from W.

2. The relation for any relationship in which the weak entity set W appears must use as a key for W all of its key attributes, including those of other entity sets that contribute to W's key.

3. However, a supporting relationship R, from the weak entity set W to a supporting entity set, need not be converted to a relation at all. The justification is that, as discussed in Section 4.5.3, the attributes of many-one relationship R's relation will either *be* attributes of the relation for W, or (in the case of attributes on R) can be added to the schema for W's relation.

Of course, when introducing additional attributes to build the key of a weak entity set, we must be careful not to use the same name twice. If necessary, we rename some or all of these attributes.

Example 4.30: Let us consider the weak entity set *Crews* from Fig. 4.20, which we reproduce here as Fig. 4.28. From this diagram we get three relations, whose schemas are:

```
Studios(name, addr)
Crews(number, studioName, crewChief)
Unit-of(number, studioName, name)
```

The first relation, `Studios`, is constructed in a straightforward manner from the entity set of the same name. The second, `Crews`, comes from the weak entity set *Crews*. The attributes of this relation are the key attributes of *Crews* and the one nonkey attribute of *Crews*, which is *crewChief*. We have chosen `studioName` as the attribute in relation `Crews` that corresponds to the attribute *name* in the entity set *Studios*.

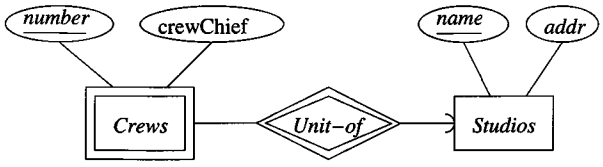

Figure 4.28: The crews example of a weak entity set

The third relation, `Unit-of`, comes from the relationship of the same name. As always, we represent an E/R relationship in the relational model by a relation whose schema has the key attributes of the related entity sets. In this case,

`Unit-of` has attributes `number` and `studioName`, the key for weak entity set *Crews*, and attribute `name`, the key for entity set *Studios*. However, notice that since *Unit-of* is a many-one relationship, the studio `studioName` is surely the same as the studio `name`.

For instance, suppose Disney crew #3 is one of the crews of the Disney studio. Then the relationship set for E/R relationship *Unit-of* includes the pair

(Disney-crew-#3, Disney)

This pair gives rise to the tuple

(3, Disney, Disney)

for the relation `Unit-of`.

Notice that, as must be the case, the components of this tuple for attributes `studioName` and `name` are identical. As a consequence, we can "merge" the attributes `studioName` and `name` of `Unit-of`, giving us the simpler schema:

```
Unit-of(number, name)
```

However, now we can dispense with the relation `Unit-of` altogether, since its attributes are now a subset of the attributes of relation `Crews`. □

The phenomenon observed in Example 4.30 — that a supporting relationship needs no relation — is universal for weak entity sets. The following is a modified rule for converting to relations entity sets that are weak.

- If W is a weak entity set, construct for W a relation whose schema consists of:
  1. All attributes of W.
  2. All attributes of supporting relationships for W.
  3. For each supporting relationship for W, say a many-one relationship from W to entity set E, all the *key* attributes of E.

  Rename attributes, if necessary, to avoid name conflicts.
- Do *not* construct a relation for any supporting relationship for W.

4.5.5 Exercises for Section 4.5

Exercise 4.5.1: Convert the E/R diagram of Fig. 4.29 to a relational database schema.

! **Exercise 4.5.2:** There is another E/R diagram that could describe the weak entity set *Bookings* in Fig. 4.29. Notice that a booking can be identified uniquely by the flight number, day of the flight, the row, and the seat; the customer is not then necessary to help identify the booking.

Relations With Subset Schemas

You might imagine from Example 4.30 that whenever one relation R has a set of attributes that is a subset of the attributes of another relation S, we can eliminate R. That is not exactly true. R might hold information that doesn't appear in S because the additional attributes of S do not allow us to extend a tuple from R to S.

For instance, the Internal Revenue Service tries to maintain a relation `People(name, ss#)` of potential taxpayers and their social-security numbers, even if the person had no income and did not file a tax return. They might also maintain a relation `TaxPayers(name, ss#, amount)` indicating the amount of tax paid by each person who filed a return in the current year. The schema of `People` is a subset of the schema of `TaxPayers`, yet there may be value in remembering the social-security number of those who are mentioned in `People` but not in `Taxpayers`.

In fact, even identical sets of attributes may have different semantics, so it is not possible to merge their tuples. An example would be two relations `Stars(name, addr)` and `Studios(name, addr)`. Although the schemas look alike, we cannot turn star tuples into studio tuples, or vice-versa.

On the other hand, when the two relations come from the weak-entity-set construction, then there can be no such additional value to the relation with the smaller set of attributes. The reason is that the tuples of the relation that comes from the supporting relationship correspond one-for-one with the tuples of the relation that comes from the weak entity set. Thus, we routinely eliminate the former relation.

a) Revise the diagram of Fig. 4.29 to reflect this new viewpoint.

b) Convert your diagram from (a) into relations. Do you get the same database schema as in Exercise 4.5.1?

Exercise 4.5.3: The E/R diagram of Fig. 4.30 represents ships. Ships are said to be *sisters* if they were designed from the same plans. Convert this diagram to a relational database schema.

Exercise 4.5.4: Convert the following E/R diagrams to relational database schemas.

a) Figure 4.22.

b) Your answer to Exercise 4.4.1.

c) Your answer to Exercise 4.4.4(a).

d) Your answer to Exercise 4.4.4(b).

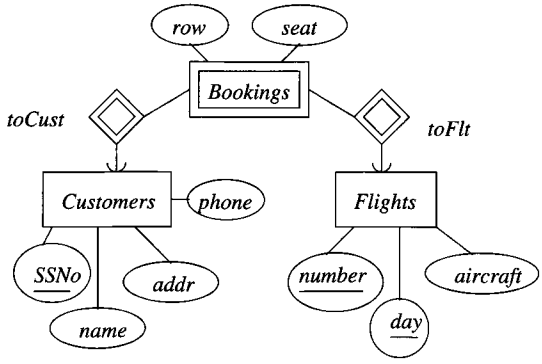

Figure 4.29: An E/R diagram about airlines

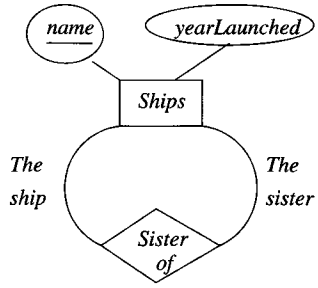

Figure 4.30: An E/R diagram about sister ships

4.6 Converting Subclass Structures to Relations

When we have an isa-hierarchy of entity sets, we are presented with several choices of strategy for conversion to relations. Recall we assume that:

- There is a root entity set for the hierarchy,
- This entity set has a key that serves to identify every entity represented by the hierarchy, and
- A given entity may have *components* that belong to the entity sets of any subtree of the hierarchy, as long as that subtree includes the root.

The principal conversion strategies are:

1. *Follow the E/R viewpoint.* For each entity set E in the hierarchy, create a relation that includes the key attributes from the root and any attributes belonging to E.

2. *Treat entities as objects belonging to a single class.* For each possible subtree that includes the root, create one relation, whose schema includes all the attributes of all the entity sets in the subtree.

3. *Use null values.* Create one relation with all the attributes of all the entity sets in the hierarchy. Each entity is represented by one tuple, and that tuple has a null value for whatever attributes the entity does not have.

We shall consider each approach in turn.

4.6.1 E/R-Style Conversion

Our first approach is to create a relation for each entity set, as usual. If the entity set E is not the root of the hierarchy, then the relation for E will include the key attributes at the root, to identify the entity represented by each tuple, plus all the attributes of E. In addition, if E is involved in a relationship, then we use these key attributes to identify entities of E in the relation corresponding to that relationship.

Note, however, that although we spoke of "isa" as a relationship, it is unlike other relationships, in that it connects components of a single entity, not distinct entities. Thus, we do not create a relation for "isa."

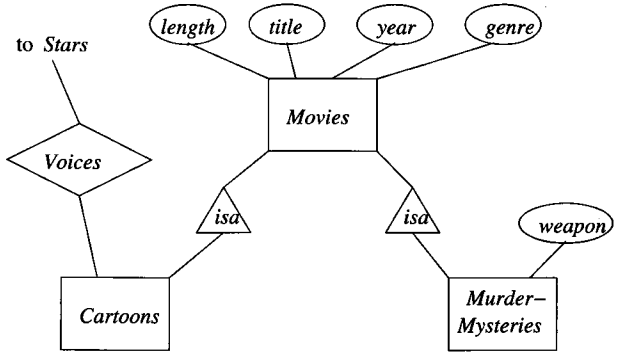

Figure 4.31: The movie hierarchy

Example 4.31: Consider the hierarchy of Fig. 4.10, which we reproduce here as Fig. 4.31. The relations needed to represent the entity sets in this hierarchy are:

1. `Movies(title, year, length, genre)`. This relation was discussed in Example 4.24, and every movie is represented by a tuple here.

2. `MurderMysteries(title, year, weapon)`. The first two attributes are the key for all movies, and the last is the lone attribute for the corresponding entity set. Those movies that are murder mysteries have a tuple here as well as in `Movies`.

3. `Cartoons(title, year)`. This relation is the set of cartoons. It has no attributes other than the key for movies, since the extra information about cartoons is contained in the relationship *Voices*. Movies that are cartoons have a tuple here as well as in `Movies`.

Note that the fourth kind of movie — those that are both cartoons and murder mysteries — have tuples in all three relations.

In addition, we shall need the relation `Voices(title, year, starName)` that corresponds to the relationship *Voices* between *Stars* and *Cartoons*. The last attribute is the key for *Stars* and the first two form the key for *Cartoons*.

For instance, the movie *Roger Rabbit* would have tuples in all four relations. Its basic information would be in `Movies`, the murder weapon would appear in `MurderMysteries`, and the stars that provided voices for the movie would appear in `Voices`.

Notice that the relation `Cartoons` has a schema that is a subset of the schema for the relation `Voices`. In many situations, we would be content to eliminate a relation such as `Cartoons`, since it appears not to contain any information beyond what is in `Voices`. However, there may be silent cartoons in our database. Those cartoons would have no voices, and we would therefore lose information should we eliminate relation `Cartoons`. □

4.6.2 An Object-Oriented Approach

An alternative strategy for converting isa-hierarchies to relations is to enumerate all the possible subtrees of the hierarchy. For each, create one relation that represents entities having components in exactly those subtrees. The schema for this relation has all the attributes of any entity set in the subtree. We refer to this approach as "object-oriented," since it is motivated by the assumption that entities are "objects" that belong to one and only one class.

Example 4.32: Consider the hierarchy of Fig. 4.31. There are four possible subtrees including the root:

1. *Movies* alone.
2. *Movies* and *Cartoons* only.
3. *Movies* and *Murder-Mysteries* only.
4. All three entity sets.

We must construct relations for all four "classes." Since only Murder-Mysteries contributes an attribute that is unique to its entities, there is actually some repetition, and these four relations are:

```
Movies(title, year, length, genre)
MoviesC(title, year, length, genre)
MoviesMM(title, year, length, genre, weapon)
MoviesCMM(title, year, length, genre, weapon)
```

If *Cartoons* had attributes unique to that entity set, then all four relations would have different sets of attributes. As that is not the case here, we could combine `Movies` with `MoviesC` (i.e., create one relation for non-murder-mysteries) and combine `MoviesMM` with `MoviesCMM` (i.e., create one relation for all murder mysteries), although doing so loses some information — which movies are cartoons.

We also need to consider how to handle the relationship *Voices* from *Cartoons* to *Stars*. If *Voices* were many-one from *Cartoons*, then we could add a `voice` attribute to `MoviesC` and `MoviesCMM`, which would represent the *Voices* relationship and would have the side-effect of making all four relations different. However, *Voices* is many-many, so we need to create a separate relation for this relationship. As always, its schema has the key attributes from the entity sets connected; in this case

```
Voices(title, year, starName)
```

would be an appropriate schema.

One might consider whether it was necessary to create two such relations, one connecting cartoons that are not murder mysteries to their voices, and the other for cartoons that *are* murder mysteries. However, there does not appear to be any benefit to doing so in this case. □

4.6.3 Using Null Values to Combine Relations

There is one more approach to representing information about a hierarchy of entity sets. If we are allowed to use `NULL` (the null value as in SQL) as a value in tuples, we can handle a hierarchy of entity sets with a single relation. This relation has all the attributes belonging to any entity set of the hierarchy. An entity is then represented by a single tuple. This tuple has `NULL` in each attribute that is not defined for that entity.

Example 4.33: If we applied this approach to the diagram of Fig. 4.31, we would create a single relation whose schema is:

```
Movie(title, year, length, genre, weapon)
```

Those movies that are not murder mysteries would have `NULL` in the `weapon` component of their tuple. It would also be necessary to have a relation `Voices` to connect those movies that are cartoons to the stars performing the voices, as in Example 4.32. □

4.6.4 Comparison of Approaches

Each of the three approaches, which we shall refer to as "straight-E/R," "object-oriented," and "nulls," respectively, have advantages and disadvantages. Here is a list of the principal issues.

1. It can be expensive to answer queries involving several relations, so we would prefer to find all the attributes we needed to answer a query in one relation. The nulls approach uses only one relation for all the attributes, so it has an advantage in this regard. The other two approaches have advantages for different kinds of queries. For instance:

   (a) A query like "what films of 2008 were longer than 150 minutes?" can be answered directly from the relation `Movies` in the straight-E/R approach of Example 4.31. However, in the object-oriented approach of Example 4.32, we need to examine `Movies`, `MoviesC`, `MoviesMM`, and `MoviesCMM`, since a long movie may be in any of these four relations.

   (b) On the other hand, a query like "what weapons were used in cartoons of over 150 minutes in length?" gives us trouble in the straight-E/R approach. We must access `Movies` to find those movies of over 150 minutes. We must access `Cartoons` to verify that a movie is a cartoon, and we must access `MurderMysteries` to find the murder weapon. In the object-oriented approach, we have only to access the relation `MoviesCMM`, where all the information we need will be found.

2. We would like not to use too many relations. Here again, the nulls method shines, since it requires only one relation. However, there is a difference between the other two methods, since in the straight-E/R approach, we use only one relation per entity set in the hierarchy. In the object-oriented approach, if we have a root and n children ($n + 1$ entity sets in all), then there are 2^n different classes of entities, and we need that many relations.

3. We would like to minimize space and avoid repeating information. Since the object-oriented method uses only one tuple per entity, and that tuple has components for only those attributes that make sense for the entity, this approach offers the minimum possible space usage. The nulls approach also has only one tuple per entity, but these tuples are "long"; i.e., they have components for all attributes, whether or not they are appropriate for a given entity. If there are many entity sets in the hierarchy, and there are many attributes among those entity sets, then a large fraction of the space could be wasted in the nulls approach. The straight-E/R method has several tuples for each entity, but only the key attributes are repeated. Thus, this method could use either more or less space than the nulls method.

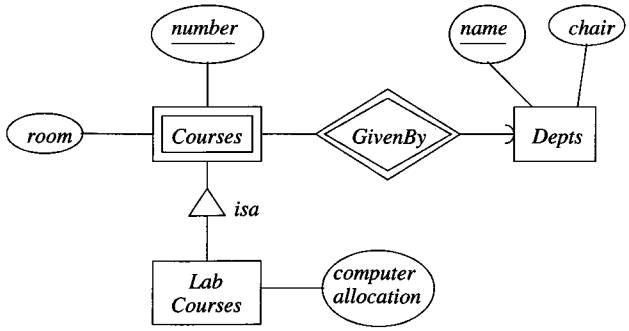

Figure 4.32: E/R diagram for Exercise 4.6.1

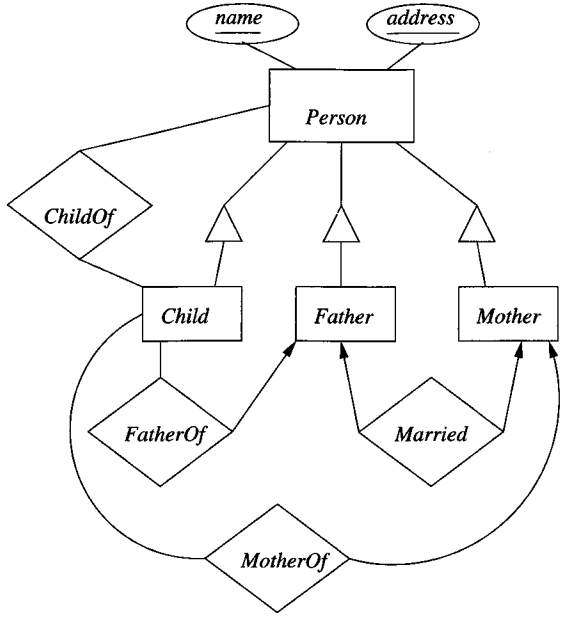

Figure 4.33: E/R diagram for Exercise 4.6.2

4.6.5 Exercises for Section 4.6

Exercise 4.6.1: Convert the E/R diagram of Fig. 4.32 to a relational database schema, using each of the following approaches:

a) The straight-E/R method.

b) The object-oriented method.

c) The nulls method.

! **Exercise 4.6.2:** Convert the E/R diagram of Fig. 4.33 to a relational database schema, using:

a) The straight-E/R method.

b) The object-oriented method.

c) The nulls method.

Exercise 4.6.3: Convert your E/R design from Exercise 4.1.7 to a relational database schema, using:

a) The straight-E/R method.

b) The object-oriented method.

c) The nulls method.

! **Exercise 4.6.4:** Suppose that we have an isa-hierarchy involving e entity sets. Each entity set has a attributes, and k of those at the root form the key for all these entity sets. Give formulas for (i) the minimum and maximum number of relations used, and (ii) the minimum and maximum number of components that the tuple(s) for a single entity have all together, when the method of conversion to relations is:

a) The straight-E/R method.

b) The object-oriented method.

c) The nulls method.

4.7 Unified Modeling Language

UML (*Unified Modeling Language*) was developed originally as a graphical notation for describing software designs in an object-oriented style. It has been extended, with some modifications, to be a popular notation for describing database designs, and it is this portion of UML that we shall study here. UML offers much the same capabilities as the E/R model, with the exception of multiway relationships. UML also offers the ability to treat entity sets as true classes, with methods as well as data. Figure 4.34 summarizes the common concepts, with different terminology, used by E/R and UML.