2.2 THE ENTITY-RELATIONSHIP MODEL

The purpose of the entity-relationship model is to allow the description of the conceptual scheme of an enterprise to be written down without the attention to efficiency or physical database design that is expected in most other models. It is normally assumed that the "entity-relationship diagram" thus constructed will be turned later into a conceptual scheme in one of the other models, e.g., the relational model, upon which real database systems are built. The transformation from entity-relationship diagram to, say, a network is fairly straightforward using constructions we shall give in this chapter, but obtaining the conceptual scheme that offers the most efficiency can be quite difficult and requires an understanding of design issues in the target model.

Entities

The term "entity" defies a formal definition, much as the terms "point" and "line" in geometry are defined only implicitly by axioms that give their properties. Suffice it to say an *entity* is a thing that exists and is distinguishable; that is, we can tell one entity from another. For example, each person is an entity, and each automobile is an entity. We could regard each ant as an entity if we had a way to distinguish one from another (say paint little numbers on their backs).

The notion of distinguishability of entities is very close to object identity, which we discussed in Section 1.5. For this reason, the entity-relationship model is generally regarded as an object-oriented model.

Entity Sets

A group consisting of all "similar" entities forms an *entity set*. Examples of entity sets are

1. All persons.
2. All red-haired persons.
3. All automobiles.

Notice from examples (1) and (2), persons and red-haired persons, that the term "similar entities" is not precisely defined, and one can establish an infinite number of different properties by which to define an entity set. One of the key steps in selecting a scheme for the real world, as it pertains to a particular database, is choosing the entity sets. As we shall see below, it is necessary to characterize all members of an entity set by a collection of characteristics called "attributes." Thus, "similarity" at least requires that a set of characteristics common to all members of the entity set can be found.

The notion of "entity set" is a scheme-level notion, in the terminology of Section 1.2. The corresponding instance-level notion is the current subset of all

members of a given entity set that are present in the database.

Example 2.1: The California Department of Motor Vehicles may design its database scheme to have an entity set AUTOMOBILES. In the current instance of that entity set are all the automobiles presently registered in California, not all automobiles in the world or all the automobiles that could ever exist. □

Attributes and Keys

Entity sets have properties, called *attributes*, which associate with each entity in the set a value from a *domain* of values for that attribute. Usually, the domain for an attribute will be a set of integers, real numbers, or character strings, but we do not rule out other types of values. For example, the entity set of persons may be declared to have attributes such as name (a character string), height (a real number), and so on.

The selection of relevant attributes for entity sets is another critical step in the design of a conceptual database scheme. An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a *key* for that entity set. In principle, each entity set has a key, since we hypothesized that each entity is distinguishable from all others. But if we do not choose, for an entity set, a collection of attributes that includes a key, then we shall not be able to distinguish one entity in the set from another. Often an arbitrary serial number is supplied as an attribute to serve as a key.

Example 2.2: An entity set that included only U.S. nationals could use the single attribute "Social Security number" as a key. However, suppose we wished to identify uniquely members of an entity set including citizens of many countries. We could not be sure that two countries do not use the same identification numbers, so an appropriate key would be the pair of attributes ID_NO and COUNTRY. □

Isa Hierarchies

We say A **isa** B, read "A is a B," if entity set B is a generalization of entity set A, or equivalently, A is a special kind of B. The primary purpose of declaring **isa** relationships between entity sets A and B is so A can inherit the attributes of B, but also have some additional attributes that don't make sense for those members of B that are not also members of A. Technically, each entity a in set A is related to exactly one entity b in set B, such that a and b are really the same entity. No b in B can be so related to two different members of A, but some members of B can be related to no member of A. The key attributes for entity set A are actually attributes of entity set B, and the values of those attributes for an entity a in A are taken from the corresponding b in B.

Example 2.3: A corporation might well have an entity set EMPLOYEES, with attributes such as ID_NO, NAME, and SALARY. If the corporation were

a baseball team, certain of the employees, the players, would have other important attributes, like BATTING_AVG or HOME_RUNS, that the other employees would not have. The most sensible way to design this scheme is to have another entity set PLAYERS, with the relationship PLAYERS **isa** EMPLOYEES. Attributes like NAME, belonging to EMPLOYEES, are inherited by PLAYERS, but only PLAYERS have attributes like BATTING_AVG. □

Relationships

A *relationship* among entity sets is an ordered list of entity sets. A particular entity set may appear more than once on the list. This list of entity sets is the scheme-level notion of a relationship. If there is a relationship $\mathcal{R}$ among entity sets $E_1, E_2, \ldots, E_k$, then the current instance of $\mathcal{R}$ is a set of k-tuples. We call such a set a *relationship set*. Each k-tuple $(e_1, e_2, \ldots, e_k)$ in relationship set $\mathcal{R}$ implies that entities $e_1, e_2, \ldots, e_k$, where e_1 is in set E_1, e_2 is in set E_2, and so on, stand in relationship $\mathcal{R}$ to each other as a group. The most common case, by far, is where $k = 2$, but lists of three or more entity sets are sometimes related.

Example 2.4: Suppose we have an entity set PERSONS and we have a relationship MOTHER_OF, whose list of entity sets is PERSONS, PERSONS. The relationship set for relationship MOTHER_OF consists of all and only those pairs (p_1, p_2) such that person p_2 is the mother of person p_1.

An alternative way of representing this information is to postulate the existence of entity set MOTHERS and relationship MOTHERS **isa** PERSONS. This arrangement would be more appropriate if the database stored values for attributes of mothers that it did not store for persons in general. Then the relationship MOTHER_OF would be the list of entity sets

PERSONS, MOTHERS

To get information about a person's mother as a person, we would compose (in the sense of ordinary set-theoretic relations) the relationships MOTHER_OF and **isa**. □

Borrowed Key Attributes

We mentioned in connection with **isa** relationships that if A **isa** B, then the key for A would naturally be the key attributes of B, and those attributes would not appear as attributes of entity set A. Thus, in Example 2.3, the key for entity set PLAYERS would most naturally be the attribute ID_NO of EMPLOYEES. Then a player would be uniquely identified by the ID_NO of the employee that is him.

There are times when we want the key for one entity set A to be attributes of another entity set B to which A is connected by a relationship $\mathcal{R}$ other than

isa. It is only necessary that $\mathcal{R}$ provide, for each entity a in A, a unique b in B to which a is related. For instance, we assumed in Example 2.2 that individuals had attribute COUNTRY that, with ID_NO, formed a key for individuals. It is just as likely that the database design would include countries as another entity set, and there would be a relationship CITIZEN_OF relating individuals to countries. Then the key for individuals would be their ID_NO and the name of the country to which they were related by CITIZEN_OF.

Note that when "borrowing" key attributes, it is essential that the relationship to be followed leads to a unique entity in the target entity set. That must be the case when following an **isa** relationship, but it need not be the case in general; e.g., can individuals be citizens of two countries? Shortly, we shall investigate the matter of functionality of relationships, which tells us when an entity of one entity set is related, via a particular relationship, to a unique member of another entity set.

Entity-Relationship Diagrams

It is useful to summarize the information in a design using *entity-relationship diagrams*, where:

1. Rectangles represent entity sets.
2. Circles represent attributes. They are linked to their entity sets by (undirected) edges. Sometimes, attributes that are part of the key for their entity set will be underlined. As a special case regarding attributes, we sometimes identify an entity set having only one attribute with the attribute itself, calling the entity set by the name of the attribute. In that case, the entity set appears as a circle attached to whatever relationships the entity set is involved in, rather than as a rectangle.
3. Diamonds represent relationships. They are linked to their constituent entity sets by edges, which can be undirected edges or directed edges (arcs); the use of arcs is discussed later when we consider functionality of relationships. The order of entity sets in the list for the relationship can be indicated by numbering edges, although the order is irrelevant unless the same entity set appears more than once on a list.

Example 2.5: Figure 2.1(a) shows a simple entity-relationship diagram, with three entity sets, EMPS, DEPTS, and MANAGERS. The first two are related by relationship ASSIGNED_TO and the second and third are related by MANAGES. For the present, we should ignore the arrows on some of the edges connecting the relationship diamonds to the entity-set rectangles. We show three attributes, NAME, PHONE, and SALARY, for EMPS; NAME is taken to be the key.[1] Departments have attributes NAME (of the department) and

[1] While we shall often imagine that "names" of entities can serve as keys for their entity

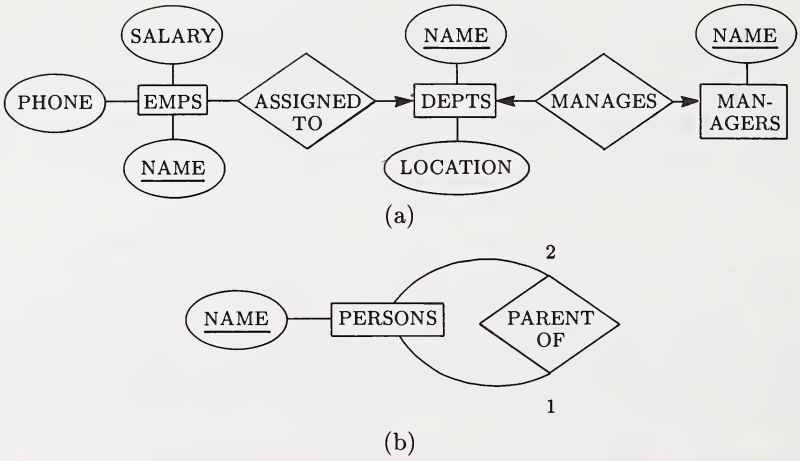

Figure 2.1 Examples of entity-relationship diagrams.

LOCATION, while MANAGERS has only the attribute name.[2]

In Figure 2.1(b) we see an entity set PERSONS and we see a relationship PARENT_OF between PERSONS and PERSONS. We also notice two edges from PARENT_OF to PERSONS; the first represents the child and the second the parent. That is, the current value of the PARENT_OF relationship set is the set of pairs (p_1, p_2) such that p_2 is known to be a parent of p_1. □

Functionality of Relationships

To model the real world adequately, it is often necessary to classify relationships according to how many entities from one entity set can be associated with how many entities of another entity set. The simplest and rarest form of relationship on two sets is *one-to-one*, meaning that for each entity in either set there is at most one associated member of the other set. For example, the relationship MANAGES between DEPTS and MANAGERS, in Figure 2.1(a), might be declared a one-to-one relationship. If so, then in the database we never can find more than one manager for a department, nor can one person manage two or more departments. It is possible that some department has no manager at the

---

sets, it is very likely that two entities with the same name can exist. Surely there could be two employees with the same name. In practice, many entity sets, such as EMPS, would be given artificial keys, such as an employee ID number. However, we shall frequently pretend that "names" are unique to simplify the set of necessary attributes.

[2] An alternative formulation of this structure would give MANAGERS no attributes and have an isa relationship from MANAGERS to EMPLOYEES. We see this treatment of employees and managers in Example 2.6.

moment, or even that someone listed in the database as a manager currently has no department to manage.

Note that the one-to-oneness of this relationship is an assumption about the real world that the database designer could choose to make or not to make. It is just as possible to assume that the same person could head two departments, or even that a department could have two managers. However, if one head for one department is the rule in this organization, then it may be possible to take advantage of the fact that MANAGES is one-to-one, when designing the physical database.

Many-One Relationships

In a *many-one* relationship, one entity in set E_2 is associated with zero or more entities in set E_1, but each entity in E_1 is associated with at most one entity in E_2. This relationship is said to be many-one *from* E_1 *to* E_2. That is, the relationship is a (partial) function from E_1 to E_2. For example, the relationship between EMPS and DEPTS in Figure 2.1(a) may well be many-one from EMPS to DEPTS, meaning that every employee is assigned to at most one department. It is possible that some employees, such as the company president, are assigned to no department.

The concept of a many-one relationship generalizes to relationships among more than two entity sets. If there is a relationship $\mathcal{R}$ among entity sets $E_1, E_2, \ldots, E_k$, and given entities for all sets but E_i, there is at most one related entity of set E_i, then we say $\mathcal{R}$ is many-one from $E_1, \ldots, E_{i-1}, E_{i+1}, \ldots, E_k$ to E_i.

Many-Many Relationships

We also encounter *many-many* relationships, where there are no restrictions on the sets of k-tuples of entities that may appear in a relationship set. For example, the relationship PARENT_OF in Figure 2.1 is many-many, because we expect to find two parents for each child, and a given individual may have any number of children. The relationship of enrollment between courses and students, mentioned in Section 2.1, is another example of a many-many relationship, because typically, many students take each course and many courses are taken by each student.

While many-many relationships appear frequently in practice, we have to be careful how these relationships are expressed in the conceptual scheme of the actual database.[3] Many data models do not allow direct expression of many-many relationships, instead requiring that they be decomposed into several

[3] Recall that the entity-relationship design is not the conceptual scheme, but rather a sketch of one, and we need to translate from entities and relationships into the data model of the DBMS that is used.

many-one relationships by techniques we shall cover later in this chapter. As we indicated in the previous section, the reason is that no efficient data structures are available in these models for implementing many-many relationships. The relational model permits direct expression of many-many relationships, but the problem of efficient implementation is merely pushed down to the physical level.

Indicating Functionality in Entity-Relationship Diagrams

In entity-relationship diagrams we use arcs, that is, edges with a direction indicated by an arrow, to indicate when a relationship is many-one or one-one. In the simplest case, a many-one relationship $\mathcal{R}$ from A to B, we place an arc from the diamond for $\mathcal{R}$ to the rectangle for B. As an example, we may suppose that employees are assigned to at most one department, which explains the arrow from ASSIGNED_TO to DEPTS in Figure 2.1(a). More generally, if $\mathcal{R}$ involves three or more entity sets and is many-one to some entity set A, we draw an arc to A and undirected edges to the other sets. More complicated mappings that are many-one to two or more entity sets will not be represented by an edge convention.

If $\mathcal{R}$ is one-one between A and B, we draw arrows from $\mathcal{R}$ to both A and B. For example, we may suppose that managers may manage only one department, and departments may have only one manager. That justifies the arcs from MANAGES to both DEPTS and MANAGERS in Figure 2.1(a). As an exception, if A **isa** B, we draw an arc only to B.

Example 2.6: Let us introduce the example of a database that will reappear at many points throughout the book. In the town of Yuppie Valley, a small supermarket, the Yuppie Valley Culinary Boutique (YVCB) has purchased a microcomputer and is about to design a database system that will hold the information the store needs to conduct its business. After due consideration, the database administrator for the system, Sally Hacker, a Sophomore at Calvin Klein Senior High School in Yuppie Valley, who works in the store every Thursday afternoon, developed the entity-relationship diagram shown in Figure 2.2. We shall consider the reasoning that lies behind this diagram in the following paragraphs.

One important aspect of the YVCB business is dealing with suppliers. Thus, Sally decided that the database should have an entity set SUPPLIERS. In our example, we'll use only two attributes, SNAME, the key, and SADDR.[4] In practice, there would probably be several more attributes stored about suppliers, e.g., their phone number.

---

[4] Several entity sets will have an attribute that could logically be called "NAME." There is nothing preventing us from giving them all an attribute NAME, but we shall distinguish the different kinds of names by using attributes SNAME (supplier name), CNAME (customer name), and so on.

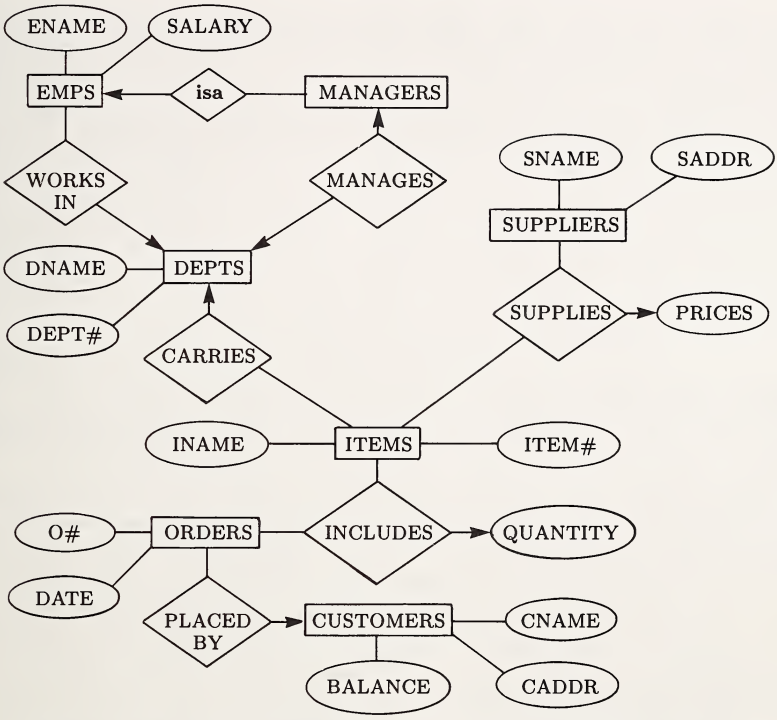

Figure 2.2 Entity-relationship diagram for the YVCB database.

One important fact about suppliers that cannot be stored conveniently as an attribute is the set of items that each supplies. Thus, Sally specified an entity set ITEMS, with two attributes, INAME and ITEM#, either of which can serve as the key. To connect items and suppliers there is a many-many relationship SUPPLIES, with the intent that each item is related to all the suppliers that can supply the item, and each supplier is related to the items it can supply. However, a third entity set, which we call PRICES, is involved in the relationship. Each supplier sets a price for each item it can supply, so we prefer to see the SUPPLIES relationship as a ternary one among ITEMS, SUPPLIERS, and PRICES, with the intent that if the relationship set for SUPPLIES contains the triple (i, s, p), then supplier s is willing to sell item i at price p.

If we look at Figure 2.2 we see a circle around entity set PRICES, rather than the customary rectangle. The reason is that PRICES presumably has only one attribute, the price itself. Thus, our exception to the way entity sets are represented applies, and we draw PRICES as if it were an attribute of the

relationship SUPPLIES. That arrangement makes some sense if we view SUPPLIES as representing item–supplier pairs, and the price as telling something about that pair.

Notice also that SUPPLIES has an arc to PRICES, reminding us that this relationship is many-one from ITEMS and SUPPLIERS to PRICES, i.e., given a supplier and an item, there is a unique price at which the supplier will sell the item. Also observe that we cannot normally break SUPPLIES into two or three binary relationships. For example, if we had one relationship between SUPPLIERS and ITEMS and another between SUPPLIERS and PRICES, then a supplier would be compelled to sell all items it sold at the same price; that is, we could not figure out which price goes with which item.

The YVCB is organized into departments, each of which has a manager and some employees. The attributes of entity set DEPTS are DNAME and DEPT#. Each department is responsible for selling some of the items, and store policy requires that each item be sold by only one department. Thus, there is a many-one relationship CARRIES from ITEMS to DEPTS.

The employees are represented by entity set EMPS, and there is a many-one relationship WORKS_IN from EMPS to DEPTS, reflecting the policy that employees are never assigned to two or more departments. The managers of departments are represented by another entity set MANAGERS. There is a one-to-one relationship MANAGES between MANAGERS and DEPTS; the one-to-one-ness reflects the assumption that in the YVCB there will never be more than one manager for a department, nor more than one department managed by one individual. Finally, since managers are employees, we have an **isa** relationship from MANAGERS to EMPS. To access the salary or name of a manager, we follow the **isa** relationship to the employee entity that the manager is, and find that information in the attributes SALARY and ENAME of EMPS.

Now, let us consider the bottom part of Figure 2.2. There we see another important entity set of the enterprise, the customers. The attributes of the entity set CUSTOMERS are CNAME, CADDR, and BALANCE. The first is the key, the customer's name. The second is the customer's address, and the third is the balance on the customer's charge account.

Customers place orders for food items, which are delivered by the YVCB. An order consists of a list of items and quantities placed by one customer. Attributes of the entity set ORDERS are O# (Order number) and DATE, but the actual content of the order is represented by a relationship INCLUDES among ORDERS, ITEMS, and QUANTITY. The latter is a trivial entity set whose entities are the integers. Since a quantity has only its value as an attribute, we show it as a circle attached to the relationship INCLUDES. That relationship is many-one from ITEMS and ORDERS to QUANTITY, since each order can have only one quantity of any given item. Finally, the many-one relationship PLACED_BY from ORDERS to CUSTOMERS tells who placed each order. □