chapter 9

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

This chapter discusses how to *design a relational database schema* based on a conceptual schema design. Figure 3.1 presented a high-level view of the database design process. In this chapter we focus on the **logical database design** step of database design, which is also known as **data model mapping**. We present the procedures to create a relational schema from an entity–relationship (ER) or an enhanced ER (EER) schema. Our discussion relates the constructs of the ER and EER models, presented in Chapters 3 and 4, to the constructs of the relational model, presented in Chapters 5 through 8. Many computer-aided software engineering (CASE) tools are based on the ER or EER models, or other similar models, as we have discussed in Chapters 3 and 4. Many tools use ER or EER diagrams or variations to develop the schema graphically and collect information about the data types and constraints, then convert the ER/EER schema automatically into a relational database schema in the DDL of a specific relational DBMS. The design tools employ algorithms similar to the ones presented in this chapter.

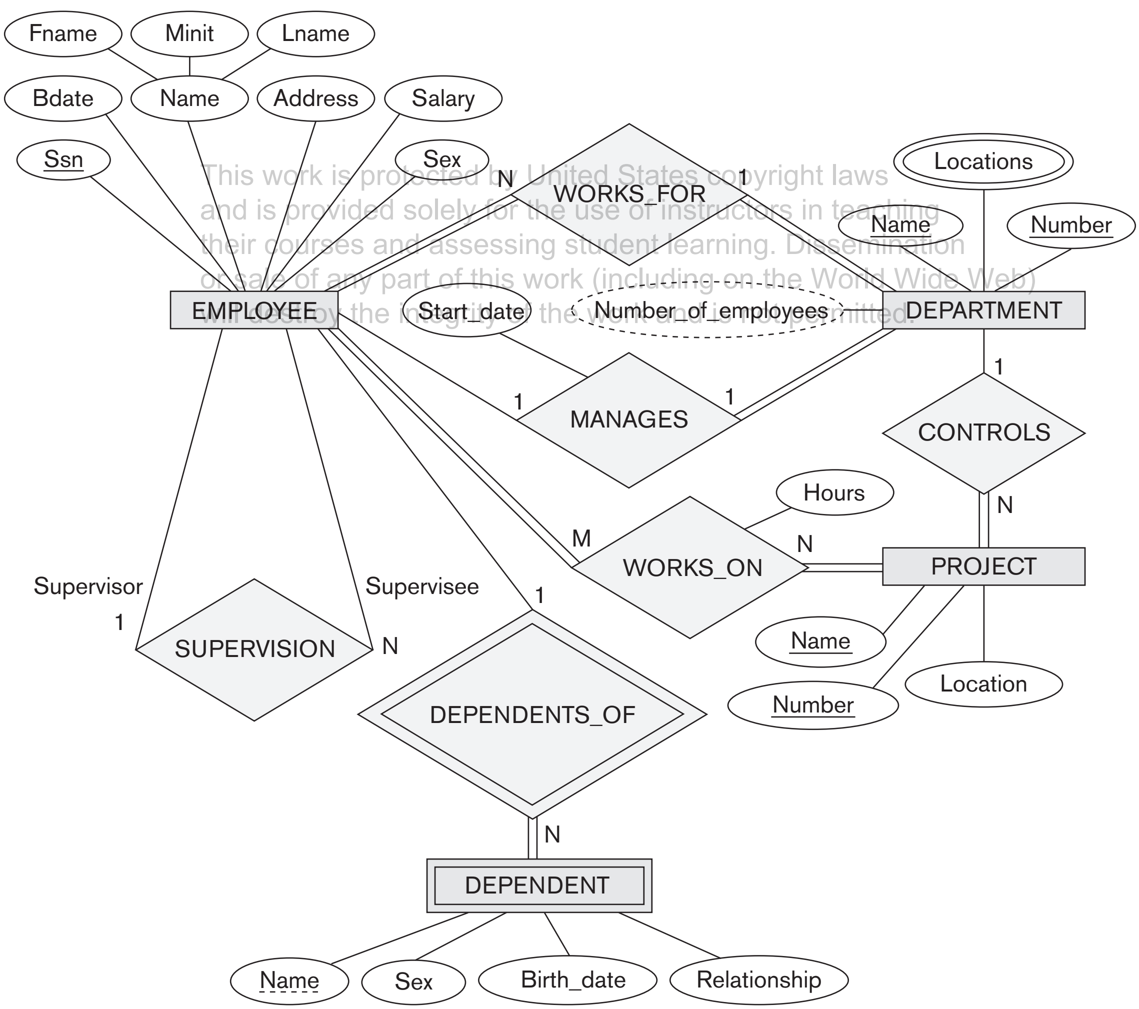
We outline a seven-step algorithm in Section 9.1 to convert the basic ER model constructs—entity types (strong and weak), binary relationships (with various structural constraints), *n*-ary relationships, and attributes (simple, composite, and multivalued)—into relations. Then, in Section 9.2, we continue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

9.1 Relational Database Design Using ER-to-Relational Mapping

9.1.1 ER-to-Relational Mapping Algorithm

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

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



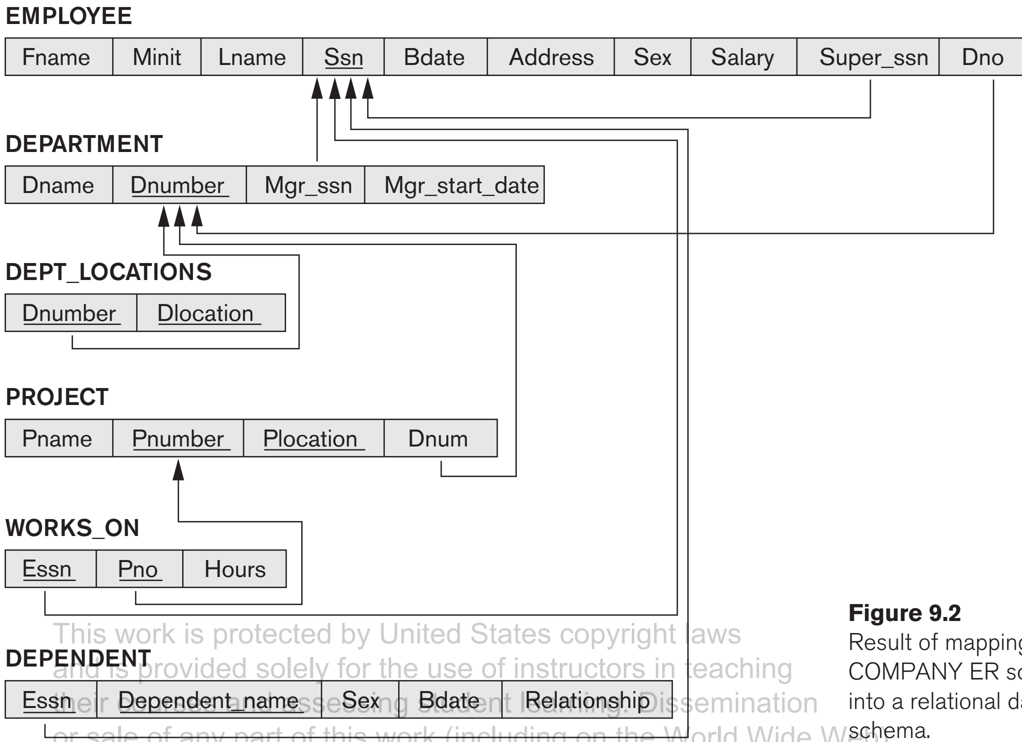


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

mapping steps. We assume that the mapping will create tables with simple single-valued attributes. The relational model constraints defined in Chapter 5, which include primary keys, unique keys (if any), and referential integrity constraints on the relations, will also be specified in the mapping results.

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

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

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

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

Step 2: Mapping of Weak Entity Types. For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R. In addition, include as foreign key attributes of R, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the identifying relationship type of W. The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any. If there is a weak entity type E_2 whose owner is also a weak entity type E_1, then E_1 should be mapped before E_2 to determine its primary key first.

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

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

(a) **EMPLOYEE**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber

PROJECT

Pname	Pnumber	Plocation

(b) **DEPENDENT**

Essn	Dependent_name	Sex	Bdate	Relationship

(c) **WORKS_ON**

Essn	Pno	Hours

(d) **DEPT_LOCATIONS**

Dnumber	Dlocation

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

It is common to choose the propagate (CASCADE) option for the referential triggered action (see Section 6.2) on the foreign key in the relation corresponding to the weak entity type, since a weak entity has an existence dependency on its owner entity. This can be used for both ON UPDATE and ON DELETE.

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

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

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

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

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

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

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

Step 4: Mapping of Binary 1:N Relationship Types. There are two possible approaches: (1) the foreign key approach and (2) the cross-reference or relationship relation approach. The first approach is generally preferred as it reduces the number of tables.

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

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

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

Step 5: Mapping of Binary M:N Relationship Types. In the traditional relational model with no multivalued attributes, the only option for M:N relationships is the **relationship relation (cross-reference) option**. For each binary M:N relationship type *R*, create a new relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types; their *combination* will form the primary key of *S*. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of *S*. Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations (as we did for

1:1 or 1:N relationship types) because of the M:N cardinality ratio; we must create a separate *relationship relation S*.

In our example, we map the M:N relationship type WORKS_ON from Figure 9.1 by creating the relation WORKS_ON in Figure 9.2. We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively (renaming is *not required;* it is a design choice). We also include an attribute Hours in WORKS_ON to represent the Hours attribute of the relationship type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {Essn, Pno}. This **relationship relation** is shown in Figure 9.3(c).

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

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

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

In our example, we create a relation DEPT_LOCATIONS (see Figure 9.3(d)). The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, whereas Dnumber—as foreign key—represents the primary key of the DEPARTMENT relation. The primary key of DEPT_LOCATIONS is the combination of {Dnumber, Dlocation}. A separate tuple will exist in DEPT_LOCATIONS for each location that a department has. It is important to note that in more recent versions of the relational model that allow array data types, the multivalued attribute can be mapped to an array attribute rather than requiring a separate table.

The propagate (CASCADE) option for the referential triggered action (see Section 6.2) should be specified on the foreign key in the relation *R* corresponding to the multivalued attribute for both ON UPDATE and ON DELETE. We should also note that the key of *R* when mapping a composite, multivalued attribute requires some analysis of the meaning of the component attributes. In some cases, when a multivalued attribute is composite, only some of the component attributes are required to be part of the key of *R*; these attributes are similar to a partial key of a weak entity type that corresponds to the multivalued attribute (see Section 3.5).

Figure 9.2 shows the COMPANY relational database schema obtained with steps 1 through 6, and Figure 5.6 shows a sample database state. Notice that we did not yet discuss the mapping of *n*-ary relationship types ($n > 2$) because none exist in Figure 9.1 ; these are mapped in a similar way to M:N relationship types by including the following additional step in the mapping algorithm.

Step 7: Mapping of *N*-ary Relationship Types. We use the **relationship relation option**. For each *n*-ary relationship type *R*, where $n > 2$, create a new relationship relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the *n*-ary relationship type (or simple components of composite attributes) as attributes of *S*. The primary key of *S* is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types *E* participating in *R* is 1, then the primary key of *S* should not include the foreign key attribute that references the relation *E′* corresponding to *E* (see the discussion in Section 3.9.2 concerning constraints on *n*-ary relationships).

Consider the ternary relationship type SUPPLY in Figure 3.17, which relates a SUPPLIER *s*, PART *p*, and PROJECT *j* whenever *s* is currently supplying *p* to *j*; this can be mapped to the relation SUPPLY shown in Figure 9.4, whose primary key is the combination of the three foreign keys {Sname, Part_no, Proj_name}.

9.1.2 Discussion and Summary of Mapping for ER Model Constructs

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

Figure 9.4
Mapping the *n*-ary relationship type SUPPLY from Figure 3.17(a).

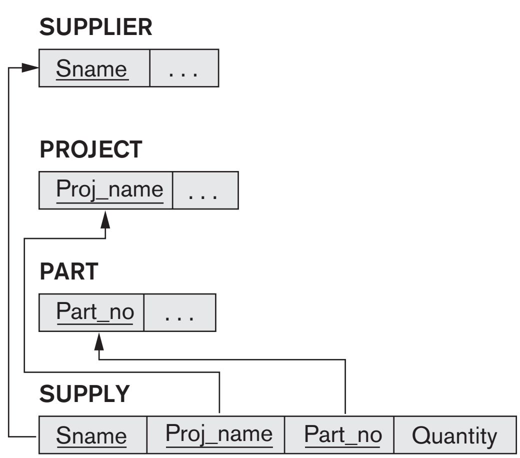


Table 9.1 Correspondence between ER and Relational Models

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

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

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

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

The basic relational algebra does not have a NEST or COMPRESS operation that would produce a set of tuples of the form {<'1', 'Houston'>, <'4', 'Stafford'>, <'5', {'Bellaire', 'Sugarland', 'Houston'}>} from the DEPT_LOCATIONS relation in Figure 3.6. This is a serious drawback of the basic normalized or *flat* version of the relational model. The object data model and object-relational systems (see Chapter 12) do allow multivalued attributes by using the array type for the attribute.

9.2 Mapping EER Model Constructs to Relations

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

9.2.1 Mapping of Specialization or Generalization

There are several options for mapping a number of subclasses that together form a specialization (or alternatively, that are generalized into a superclass), such as the {SECRETARY, TECHNICIAN, ENGINEER} subclasses of EMPLOYEE in Figure 4.4. The two main options are to map the whole specialization into a **single table**, or to map it into **multiple tables**. Within each option are variations that depend on the constraints on the specialization/generalization.

We can add a further step to our ER-to-relational mapping algorithm from Section 9.1.1, which has seven steps, to handle the mapping of specialization. Step 8, which follows, gives the most common options; other mappings are also possible. We discuss the conditions under which each option should be used. We use Attrs(R) to denote *the attributes of a relation* R, and PK(R) to denote the *primary key of* R. First we describe the mapping formally, then we illustrate it with examples.

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

- **Option 8A: Multiple relations—superclass and subclasses.** Create a relation L for C with attributes Attrs(L) = $\{k, a_1, \ldots, a_n\}$ and PK(L) = k. Create a relation L_i for each subclass S_i, $1 \le i \le m$, with the attributes Attrs(L_i) = $\{k\} \cup$ {attributes of S_i} and PK(L_i) = k. This option works for any specialization (total or partial, disjoint or overlapping).
- **Option 8B: Multiple relations—subclass relations only.** Create a relation L_i for each subclass S_i, $1 \le i \le m$, with the attributes Attrs(L_i) = {attributes of S_i} $\cup \{k, a_1, \ldots, a_n\}$ and PK(L_i) = k. This option only works for a specialization whose subclasses are *total* (every entity in the superclass must belong to (at least) one of the subclasses). Additionally, it is only recommended if the specialization has the *disjointedness constraint* (see Section 4.3.1). If the specialization is *overlapping*, the same entity may be duplicated in several relations.
- **Option 8C: Single relation with one type attribute.** Create a single relation L with attributes Attrs(L) = $\{k, a_1, \ldots, a_n\} \cup$ {attributes of S_1} $\cup \ldots \cup$ {attributes of S_m} $\cup \{t\}$ and PK(L) = k. The attribute t is called a **type** (or **discriminating**) attribute whose value indicates the subclass to which each tuple belongs, if any. This option works only for a specialization whose subclasses are *disjoint,* and has the potential for generating many NULL values if many specific (local) attributes exist in the subclasses.
- **Option 8D: Single relation with multiple type attributes.** Create a single relation schema L with attributes Attrs(L) = $\{k, a_1, \ldots, a_n\} \cup$ {attributes of S_1} $\cup \ldots \cup$ {attributes of S_m} $\cup \{t_1, t_2, \ldots, t_m\}$ and PK(L) = k. Each t_i, $1 \le i \le m$, is a **Boolean type attribute** indicating whether or not a tuple belongs to subclass S_i. This option is used for a specialization whose subclasses are *overlapping* (but will also work for a disjoint specialization).

Options 8A and 8B are the **multiple-relation options**, whereas options 8C and 8D are the **single-relation options**. Option 8A creates a relation L for the superclass C and its attributes, plus a relation L_i for each subclass S_i; each L_i includes the specific (local) attributes of S_i, plus the primary key of the superclass C, which is propagated to L_i and becomes its primary key. It also becomes a foreign key to the superclass relation. An EQUIJOIN operation on the primary key between any L_i and L produces all the specific and inherited attributes of the entities in S_i. This option is illustrated in Figure 9.5(a) for the EER schema in Figure 4.4. Option 8A works for any constraints on the specialization: disjoint or overlapping, total or partial. Notice that the constraint

$$\pi_{<k>}(L_i) \subseteq \pi_{<k>}(L)$$

must hold for each L_i. This specifies a foreign key from each L_i to L.

In option 8B, the EQUIJOIN operation between each subclass and the superclass is *built into* the schema and the superclass relation L is done away with, as illustrated in Figure 9.5(b) for the EER specialization in Figure 4.3(b). This option works well only when *both* the disjoint and total constraints hold. If the specialization is not total, an entity that does not belong to any of the subclasses S_i is lost. If the specialization is not disjoint, an entity belonging to more than one subclass will have its

(a) EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type

SECRETARY

Ssn	Typing_speed

TECHNICIAN

Ssn	Tgrade

ENGINEER

Ssn	Eng_type

(b) CAR

Vehicle_id	License_plate_no	Price	Max_speed	No_of_passengers

TRUCK

Vehicle_id	License_plate_no	Price	No_of_axles	Tonnage

(c) EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type	Typing_speed	Tgrade	Eng_type

(d) PART

Part_no	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price

Figure 9.5
Options for mapping specialization or generalization. (a) Mapping the EER schema in Figure 4.4 using option 8A. (b) Mapping the EER schema in Figure 4.3(b) using option 8B. (c) Mapping the EER schema in Figure 4.4 using option 8C. (d) Mapping Figure 4.5 using option 8D with Boolean type fields Mflag and Pflag.

inherited attributes from the superclass C stored redundantly in more than one table L_i. With option 8B, no relation holds all the entities in the superclass C; consequently, we must apply an OUTER UNION (or FULL OUTER JOIN) operation (see Section 6.4) to the L_i relations to retrieve all the entities in C. The result of the outer union will be similar to the relations under options 8C and 8D except that the type fields will be missing. Whenever we search for an arbitrary entity in C, we must search all the m relations L_i.

Options 8C and 8D create a single relation to represent the superclass C and all its subclasses. An entity that does not belong to some of the subclasses will have NULL values for the specific (local) attributes of these subclasses. These options are not recommended if many specific attributes are defined for the subclasses. If few local subclass attributes exist, however, these mappings are preferable to options 8A and 8B because they do away with the need to specify JOIN operations; therefore, they can yield a more efficient implementation for queries.

Option 8C is used to handle disjoint subclasses by including a single **type** (or **image** or **discriminating**) **attribute** t to indicate to which of the m subclasses each tuple belongs; hence, the domain of t could be $\{1, 2, \ldots, m\}$. If the specialization is partial, t can have NULL values in tuples that do not belong to any subclass. If the specialization is attribute-defined, that attribute itself serves the purpose of t and t is not needed; this option is illustrated in Figure 9.5(c) for the EER specialization in Figure 4.4.

Option 8D is designed to handle overlapping subclasses by including *m Boolean* **type** (or **flag**) fields, one for *each* subclass. It can also be used for disjoint subclasses.

Each type field t_i can have a domain {yes, no}, where a value of yes indicates that the tuple is a member of subclass S_i. If we use this option for the EER specialization in Figure 4.4, we would include three type attributes—Is_a_secretary, Is_a_engineer, and Is_a_technician—instead of the Job_type attribute in Figure 9.5(c). Figure 9.5(d) shows the mapping of the specialization from Figure 4.5 using option 8D.

For a multilevel specialization (or generalization) hierarchy or lattice, we do not have to follow the same mapping option for all the specializations. Instead, we can use one mapping option for part of the hierarchy or lattice and other options for other parts. Figure 9.6 shows one possible mapping into relations for the EER lattice in Figure 4.6. Here we used option 8A for PERSON/{EMPLOYEE, ALUMNUS, STUDENT}, and option 8C for EMPLOYEE/{STAFF, FACULTY, STUDENT_ASSISTANT} by including the type attribute Employee_type. We then used the single-table option 8D for STUDENT_ASSISTANT/{RESEARCH_ASSISTANT, TEACHING_ASSISTANT} by including the type attributes Ta_flag and Ra_flag in EMPLOYEE. We also used option 8D for STUDENT/STUDENT_ASSISTANT by including the type attributes Student_assist_flag in STUDENT, and for STUDENT/{GRADUATE_STUDENT, UNDERGRADUATE_STUDENT} by including the type attributes Grad_flag and Undergrad_flag in STUDENT. In Figure 9.6, all attributes whose names end with *type* or *flag* are type fields.

9.2.2 Mapping of Shared Subclasses (Multiple Inheritance)

A shared subclass, such as ENGINEERING_MANAGER in Figure 4.6, is a subclass of several superclasses, indicating multiple inheritance. These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category (union type) as we discussed in Section 4.4. We can apply any of the options discussed in step 8 to a shared subclass, subject to the restrictions discussed in step 8 of the mapping algorithm. In Figure 9.6, options 8C and 8D are used for the shared subclass STUDENT_ASSISTANT. Option 8C is used in the EMPLOYEE relation (Employee_type attribute) and option 8D is used in the STUDENT relation (Student_assist_flag attribute).

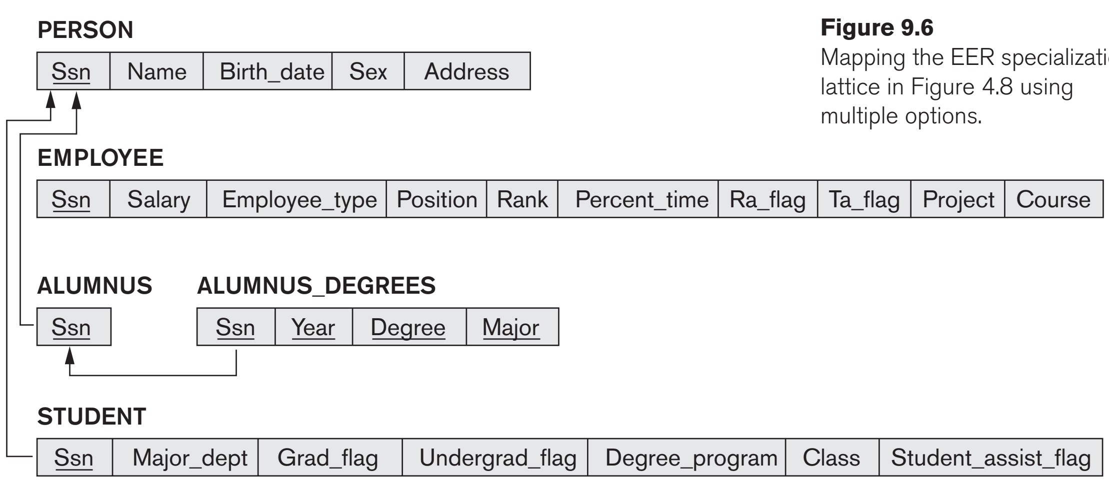


Figure 9.6
Mapping the EER specialization lattice in Figure 4.8 using multiple options.

9.2.3 Mapping of Categories (Union Types)

We add another step to the mapping procedure—step 9—to handle categories. A category (or union type) is a subclass of the *union* of two or more superclasses that can have different keys because they can be of different entity types (see Section 4.4). An example is the OWNER category shown in Figure 4.8, which is a subset of the union of three entity types PERSON, BANK, and COMPANY. The other category in that figure, REGISTERED_VEHICLE, has two superclasses that have the same key attribute.

Step 9: Mapping of Union Types (Categories). For mapping a category whose defining superclasses have different keys, it is customary to specify a new key attribute, called a **surrogate key**, when creating a relation to correspond to the union type. The keys of the defining classes are different, so we cannot use any one of them exclusively to identify all entities in the relation. In our example in Figure 4.8, we create a relation OWNER to correspond to the OWNER category, as illustrated in Figure 9.7, and include any attributes of the category in this relation. The primary key of the OWNER relation is the surrogate key, which we called Owner_id. We also

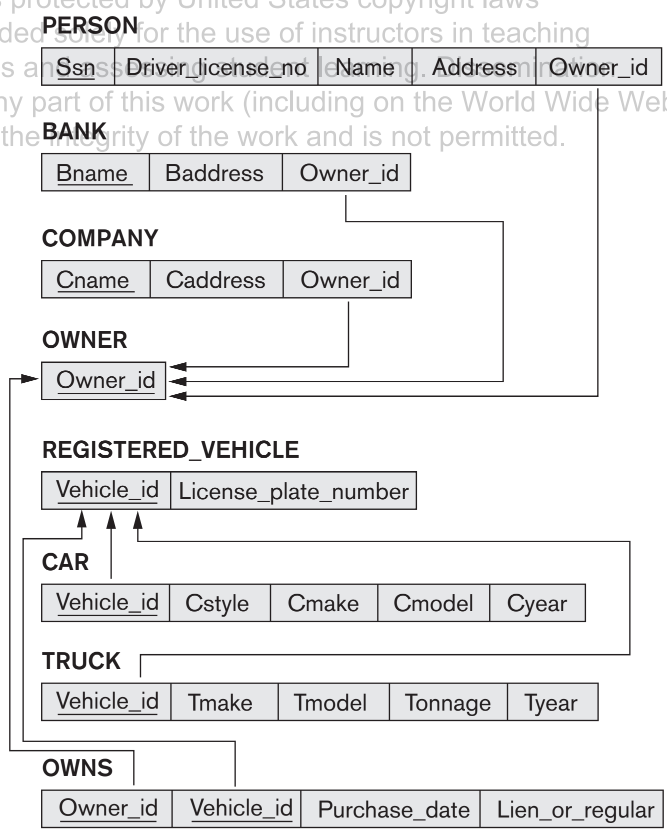


Figure 9.7
Mapping the EER categories (union types) in Figure 4.8 to relations.

include the surrogate key attribute Owner_id as foreign key in each relation corresponding to a superclass of the category, to specify the correspondence in values between the surrogate key and the original key of each superclass. Notice that if a particular PERSON (or BANK or COMPANY) entity is not a member of OWNER, it would have a NULL value for its Owner_id attribute in its corresponding tuple in the PERSON (or BANK or COMPANY) relation, and it would not have a tuple in the OWNER relation. It is also recommended to add a type attribute (not shown in Figure 9.7) to the OWNER relation to indicate the particular entity type to which each tuple belongs (PERSON or BANK or COMPANY).

For a category whose superclasses have the same key, such as VEHICLE in Figure 4.8, there is no need for a surrogate key. The mapping of the REGISTERED_VEHICLE category, which illustrates this case, is also shown in Figure 9.7.

9.3 Summary

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

Review Questions

9.1. **(a)** Discuss the correspondences between the ER model constructs and the relational model constructs. Show how each ER model construct can be mapped to the relational model and discuss any alternative mappings. (b) Discuss the options for mapping EER model constructs to relations, and the conditions under which each option could be used.

Exercises

9.2. Map the UNIVERSITY database schema shown in Figure 3.20 into a relational database schema.

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

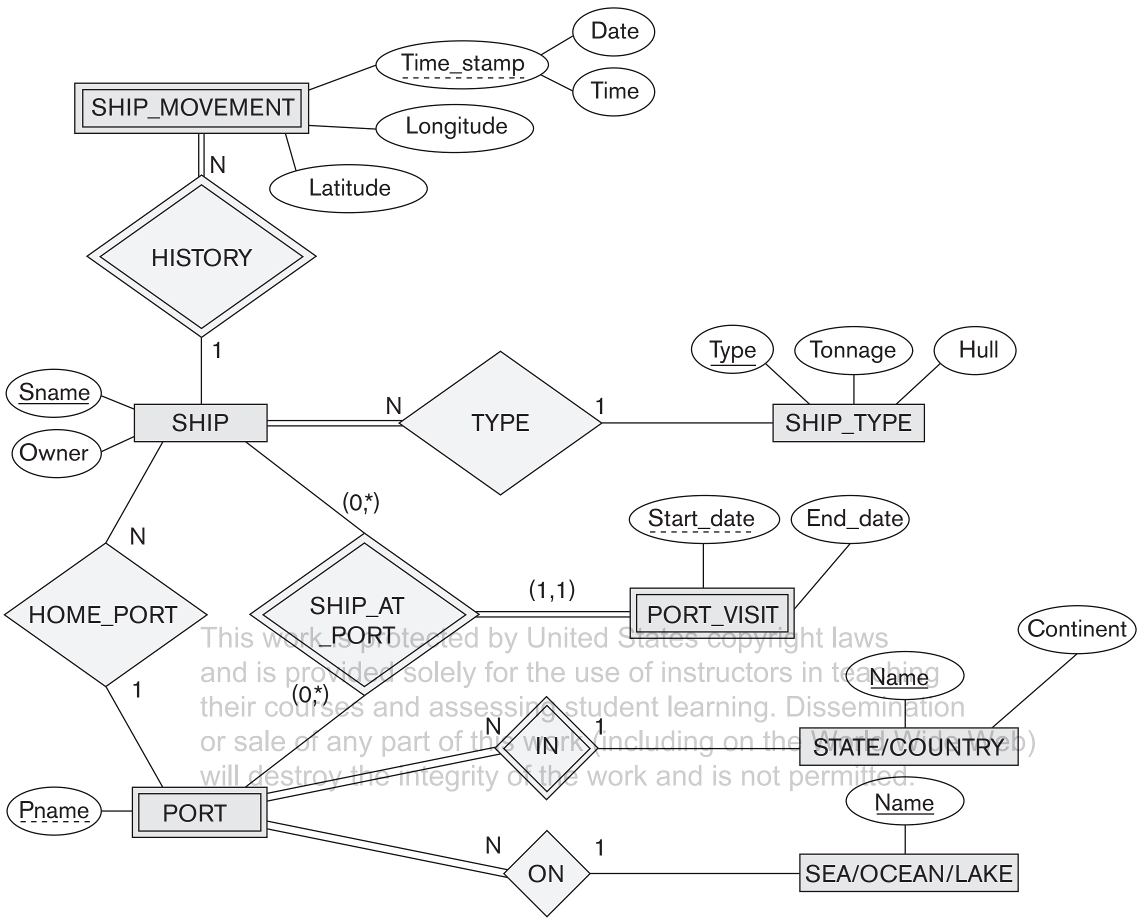


Figure 9.8
An ER schema for a SHIP_TRACKING database.

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

9.5. Map the BANK ER schema of Exercise 3.23 (shown in Figure 3.21) into a relational schema. Specify all primary keys and foreign keys. Repeat for the AIRLINE schema (Figure 3.20) of Exercise 3.19 and for the other schemas for Exercises 3.16 through 3.24.

9.6. Map the EER diagrams in Figures 4.9 and 4.12 into relational schemas. Justify your choice of mapping options.

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

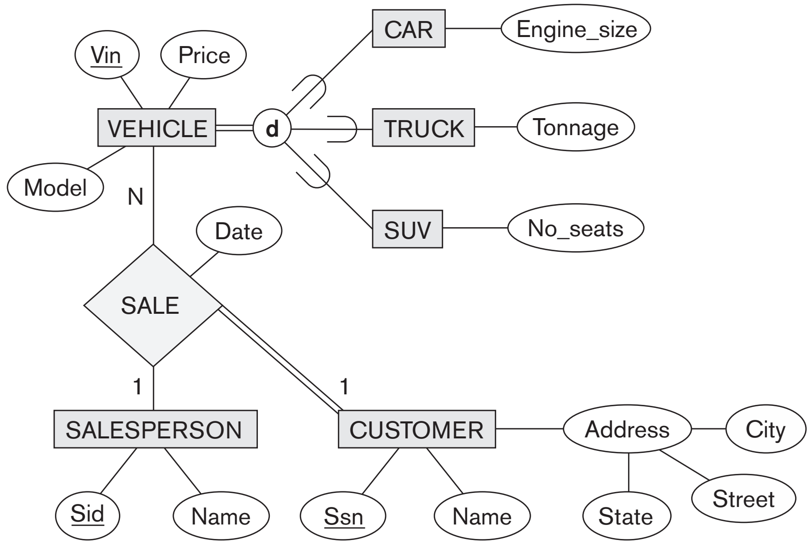


Figure 9.9
EER diagram for a car dealer.

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

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

9.9. Using the attributes you provided for the EER diagram in Exercise 4.27, map the complete schema into a set of relations. Choose an appropriate option out of 8A thru 8D from Section 9.2.1 in doing the mapping of generalizations and defend your choice.

Laboratory Exercises

9.10. Consider the ER design for the UNIVERSITY database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 3.31. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

9.11. Consider the ER design for the MAIL_ORDER database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 3.32. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

9.12. Consider the ER design for the CONFERENCE_REVIEW database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 3.34. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

9.13. Consider the EER design for the GRADE_BOOK database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 4.28. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

9.14. Consider the EER design for the ONLINE_AUCTION database that was modeled using a tool like ERwin or Rational Rose in Laboratory Exercise 4.29. Using the SQL schema generation feature of the modeling tool, generate the SQL schema for an Oracle database.

Selected Bibliography

The original ER-to-relational mapping algorithm was described in Chen's classic paper (Chen, 1976). Batini et al. (1992) discuss a variety of mapping algorithms from ER and EER models to legacy models and vice versa.