6.3 Mapping a Conceptual ER Model to a Relational Model

There exists a plethora of database modeling tools that allow the database designer to draw an (E)ER model and automatically generate a relational data model from it. If the correct translation rules are applied, the resulting relational model will automatically be normalized. Therefore, although the translation can be automated, it is useful to study these rules in detail. They provide us with valuable insights into the intricacies of good database design and the consequences of certain design decisions, by linking relational concepts to their (E)ER counterparts. In this section, we discuss how to map a conceptual ER model to a relational model. After that, we move on to mapping EER constructs.

6.3.1 Mapping Entity Types

The first step is to map each entity type into a relation. Simple attribute types can be directly mapped. A composite attribute type needs to be decomposed into its component attribute types. One of the key attribute types of the entity type can be set as the primary key of the relation.

You can see this illustrated in Figure 6.14. We have two entity types: EMPLOYEE and PROJECT. We create relations for both:

EMPLOYEE(SSN, address, first name, last name)
PROJECT(PNR, pname, pduration)

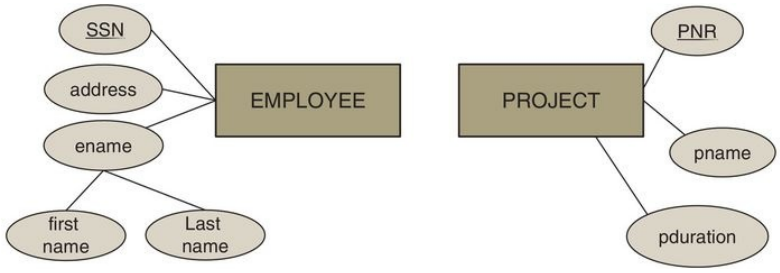


Figure 6.14 Mapping entity types to relations.

The EMPLOYEE entity type has three attribute types: SSN, which is the key attribute type; address, which is considered as an atomic attribute type; and ename, which is a composite attribute type consisting of first name and last name. The PROJECT entity type also has three attribute types: PNR, which is the key attribute type; pname; and pduration. You can see that both key attribute types SSN and PNR have been mapped to the primary keys of both relations. Also, note that the ename composite attribute type has been decomposed into first name and last name in the relation EMPLOYEE.

6.3.2 Mapping Relationship Types

Once we have mapped the entity types, we can continue with the relationship types. The mapping depends upon the degree and cardinalities, as we illustrate in what follows.

6.3.2.1 Mapping a Binary 1:1 Relationship type

For a binary 1:1 relationship type, we create two relations – one for each entity type participating in the relationship type. The connection can be made by including a foreign key in one of the relations to the primary key of the other. In case of existence dependency, we put the foreign key in the existence-dependent relation and declare it as NOT NULL. The attribute types of the 1:1 relationship type can then be added to the relation with the foreign key.

Let's consider the MANAGES relationship type between EMPLOYEE and DEPARTMENT, as depicted in Figure 6.15.

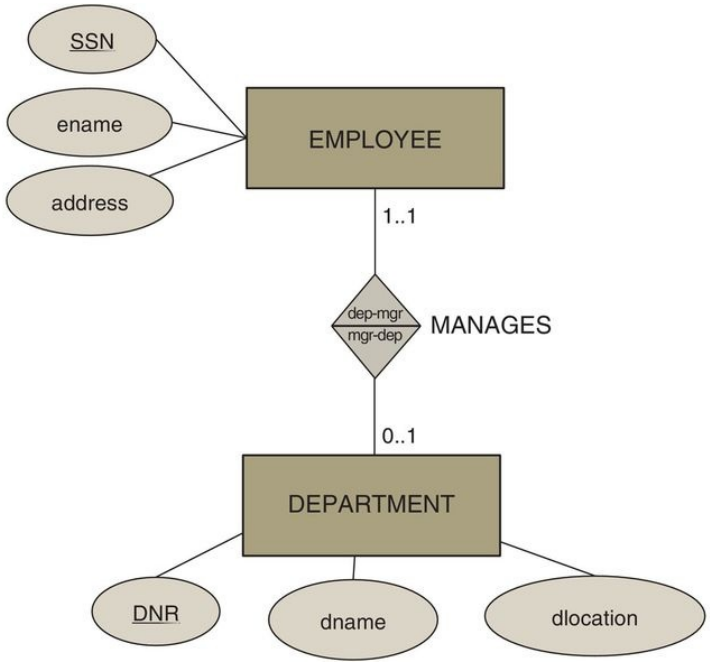


Figure 6.15 Mapping 1:1 ER relationship types to the relational model.

Remember, an employee manages either zero or one department, whereas a department is managed by exactly one employee, which means DEPARTMENT is existence-dependent on EMPLOYEE. We create relations for both entity types and add the corresponding attribute types as follows:

EMPLOYEE(SSN, ename, address)
DEPARTMENT(DNR, dname, dlocation)

The question now is: How do we map the relationship type? One option would be to add a foreign key DNR to the EMPLOYEE relation, which refers to the primary key DNR in DEPARTMENT as follows:

EMPLOYEE(SSN, ename, address, *DNR*)
DEPARTMENT(DNR, dname, dlocation)

This foreign key can be NULL, since not every employee manages a department. Let's now find out how many of the four cardinalities of the relationship type are correctly modeled, using Figure 6.16.

EMPLOYEE(SSN, ename, address, *DNR*)

511	John Smith	14 Avenue of the Americas, New York	001
289	Paul Barker	208 Market Street, San Francisco	003
356	Emma Lucas	432 Wacker Drive, Chicago	NULL
412	Michael Johnson	1134 Pennsylvania Avenue, Washington	NULL
564	Sarah Adams	812 Collins Avenue, Miami	001

DEPARTMENT(DNR, dname, dlocation)

001	Marketing	3th floor
002	Call center	2nd floor
003	Finance	basement
004	ICT	1st floor

Figure 6.16 Example tuples for mapping a 1:1 relationship type.

We start from DEPARTMENT. Can a department have zero managers? Yes, this is the case for department number 2, the Call Center, which has no manager assigned as its department number 002 does not appear in the DNR column of the EMPLOYEE table. Also, the ICT department has no manager. Can a department have more than one manager? Yes, this is the case for department number 001, marketing, which has two managers: employee 511, John Smith, and employee 564, Sarah Adams. Can an employee manage zero departments? Yes, this is the case for Emma Lucas and Michael Johnson. Can an employee manage more than one department? No, since the EMPLOYEE relation is normalized and the foreign key DNR should thus be single-valued, as required by the first normal form. To summarize, out of the four cardinalities, only two are supported. Moreover, this option generates a lot of NULL values for the DNR foreign key, as there are typically many employees who are not managing any department.

Another option would be to include SSN as a foreign key in DEPARTMENT, referring to SSN in EMPLOYEE:

EMPLOYEE(SSN, ename, address)
DEPARTMENT(DNR, dname, dlocation, *SSN*)

This foreign key should be declared as NOT NULL, since every department should have exactly one manager. Let’s now also look at the other cardinalities, using Figure 6.17.

EMPLOYEE(SSN, ename, address)

511	John Smith	14 Avenue of the Americas, New York
289	Paul Barker	208 Market Street, San Francisco
356	Emma Lucas	432 Wacker Drive, Chicago

DEPARTMENT(DNR, dname, dlocation, *SSN*)

001	Marketing	3th floor	511
002	Call center	2nd floor	511
003	Finance	basement	289
004	ICT	1st floor	511

Figure 6.17 Example tuples for mapping a 1:1 relationship type.

Can you have employees that manage zero departments? Yes, this is the case for Emma Lucas since her SSN 356 does not appear in the SSN column of the DEPARTMENT table. Can we make sure that an employee manages at most one department? In fact, we cannot! As you can see, John Smith manages three departments. Hence, out of the four cardinalities, three are supported. This option is to be preferred above the previous one, although it is not perfect. The semantics lost in the mapping should be documented and followed up using application code.

6.3.2.2 Mapping a Binary 1:N Relationship Type

Binary 1:N relationship types can be mapped by including a foreign key in the relation corresponding to the participating entity type at the N-side of the relationship type (e.g., the EMPLOYEE relation in Figure 6.18). The foreign key refers to the primary key of the relation corresponding to the entity type at the 1-side of the relationship type (e.g., the DEPARTMENT relation in Figure 6.18). Depending upon the minimum cardinality, the foreign key can be declared as NOT NULL or NULL ALLOWED. The attribute types (e.g., starting date in

Figure 6.18) of the 1:N relationship type can be added to the relation corresponding to the participating entity type.

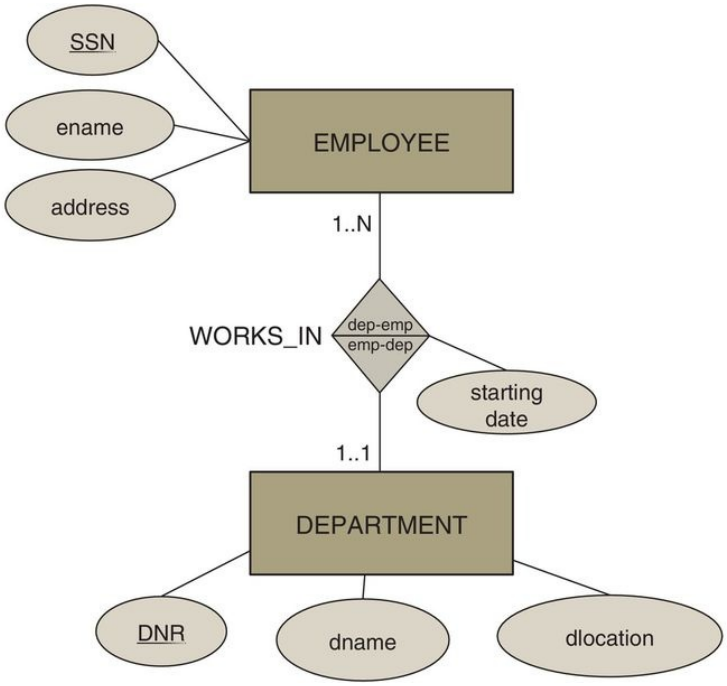


Figure 6.18 Mapping 1:N ER relationship types to the relational model.

The WORKS_IN relationship type is an example of a 1:N relationship type. An employee works in exactly one department, whereas a department can have one to N employees working in it. The attribute type starting date represents the date at which an employee started working in a department. As with 1:1 relationships, we first create the relations EMPLOYEE and DEPARTMENT for both entity types:

EMPLOYEE(SSN, ename, address)
DEPARTMENT(DNR, dname, dlocation)

We can again explore two options to establish the relationship type in the relational model. Since a department can have multiple employees, we cannot add a foreign key to it as this would create a multi-valued attribute type, which is

not tolerated in the relational model. That's why we add DNR as a foreign key to the EMPLOYEE relation.

EMPLOYEE(SSN, ename, address, starting date, *DNR*)
DEPARTMENT(DNR, dname, dlocation)

Since the minimum cardinality is one, this foreign key is defined as NOT NULL, ensuring that an employee works in exactly one department. What about the other cardinalities? We can find out using Figure 6.19.

EMPLOYEE(SSN, ename, address, starting date, *DNR*)

511	John Smith	14 Avenue of the Americas, New York	01/01/2000	001
289	Paul Barker	208 Market Street, San Francisco	01/01/1998	001
356	Emma Lucas	432 Wacker Drive, Chicago	01/01/2010	002

DEPARTMENT(DNR, dname, dlocation)

001	Marketing	3th floor
002	Call center	2nd floor
003	Finance	basement
004	ICT	1st floor

Figure 6.19 Example tuples for mapping a 1:N relationship type.

Can a department have more than one employee? Yes, this is the case for the marketing department, which has two employees – John Smith and Paul Barker. Can we guarantee that every department has at least one employee? In fact, we cannot. The finance and ICT departments have no employees. Out of the four cardinalities, three are supported. Note that the attribute type starting date has also been added to the EMPLOYEE relation.

6.3.2.3 Mapping a Binary M:N Relationship Type

M:N relationship types are mapped by introducing a new relation R. The primary key of R is a combination of foreign keys referring to the primary keys

of the relations corresponding to the participating entity types. The attribute types of the M:N relationship type can also be added to R.

The WORKS_ON relationship type shown in Figure 6.20 is an example of an M:N relationship type. An employee works on zero to N projects, whereas a project is being worked on by zero to M employees. We start by creating relations for both entity types. We cannot add a foreign key to the EMPLOYEE relation as this would give us a multi-valued attribute type since an employee can work on multiple projects. Likewise, we cannot add a foreign key to the project relation, as a project is being worked on by multiple employees. In other words, we need to create a new relation to map the ER relationship type WORKS_ON:

EMPLOYEE(<u>SSN</u>, ename, address)
PROJECT(<u>PNR</u>, pname, pduration)
WORKS_ON(<u>*SSN*</u>, <u>*PNR*</u>, hours)

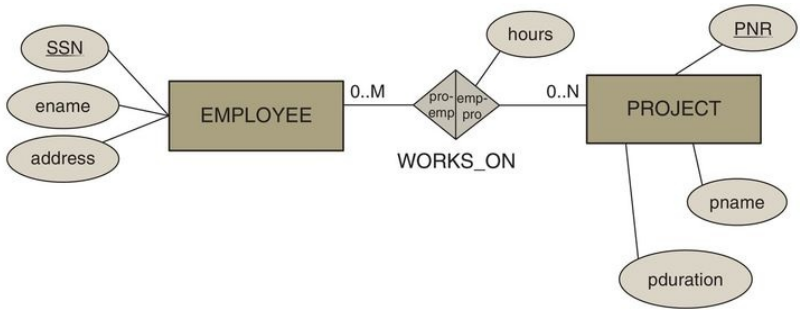


Figure 6.20 Mapping M:N ER relationship types to the relational model.

The WORKS_ON relation has two foreign keys, SSN and PNR, which together make up the primary key and, therefore, cannot be NULL (entity integrity constraint). The hours attribute type is also added to this relation.

Figure 6.21 shows some example tuples of the EMPLOYEE, PROJECT, and WORKS_ON relations.

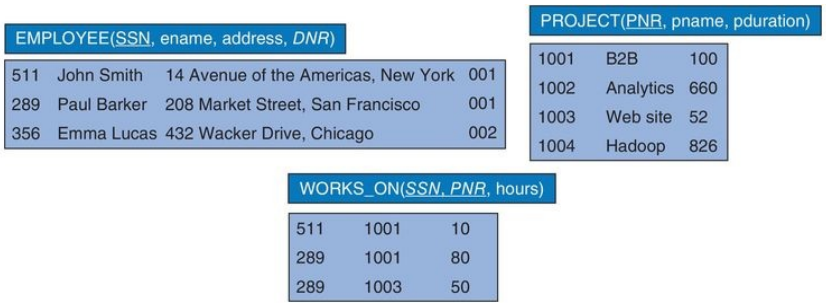


Figure 6.21 Example tuples for mapping an M:N relationship type.

All four cardinalities are successfully modeled. Emma Lucas does not work on any projects, whereas Paul Barker works on two projects. Projects 1002 and 1004 have no employees assigned, whereas project 1001 has two employees assigned.

Now let's see what happens if we change the assumptions as follows: an employee works on at least one project and a project is being worked on by at least one employee. In other words, the minimum cardinalities change to 1 on both sides. Essentially, the solution remains the same and you can see that none of the minimum cardinalities are supported since Emma Lucas is not working on any projects and projects 1002 and 1004 have no employees assigned. Out of the four cardinalities, only two are supported! This will require close follow-up during application development to make sure these missing cardinalities are enforced by the applications instead of the data model.

6.3.2.4 Mapping Unary Relationship Types

Unary or recursive relationship types can be mapped depending upon the cardinality. A recursive 1:1 or 1:N relationship type can be implemented by adding a foreign key referring to the primary key of the same relation. For an N:M recursive relationship type, a new relation R needs to be created with two NOT NULL foreign keys referring to the original relation. It is recommended to

use role names to clarify the meaning of the foreign keys. Let's illustrate this with some examples.

Figure 6.22 shows a 1:1 unary relationship type modeling the supervision relationships between employees. It can be implemented in the relational model by adding a foreign key – supervisor – to the EMPLOYEE relation, which refers to its primary key – SSN – as follows:

EMPLOYEE(SSN, ename, address, *supervisor*)

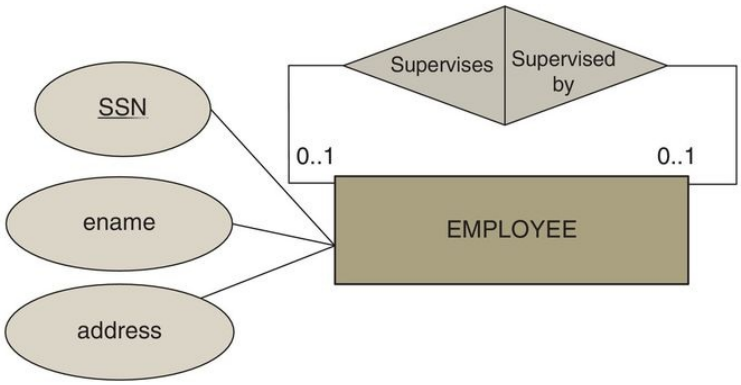


Figure 6.22 Mapping unary relationship types to the relational model.

The foreign key can be NULL since, according to the ER model, it is possible that an employee is supervised by zero other employees. Since the foreign key cannot be multi-valued, an employee cannot be supervised by more than one other employee. This is illustrated in Figure 6.23.

EMPLOYEE(SSN, ename, address, *supervisor*)

511	John Smith	14 Avenue of the Americas, New York	289
289	Paul Barker	208 Market Street, San Francisco	412
356	Emma Lucas	432 Wacker Drive, Chicago	289
412	Dan Kelly	668 Strip, Las Vegas	NULL

Figure 6.23 Example tuples for mapping a unary 1:1 relationship type.

Some employees do not supervise other employees, like Emma Lucas. However, some employees supervise more than one other employee, like Paul Barker who supervises both John Smith and Emma Lucas. To summarize, out of the four cardinalities, three are supported by our model.

Let's now change one assumption as follows: an employee can supervise at least zero, at most N other employees. The relational model stays the same, with supervisor as the foreign key referring to SSN:

EMPLOYEE(SSN, ename, address, *supervisor*)

In this case, all four cardinalities can be perfectly captured by our relational model.

Let's now set both maximum cardinalities to N and M, respectively. In other words, an employee can supervise zero to N employees, whereas an employee can be supervised by zero to M employees. We can no longer add a foreign key to the EMPLOYEE relation as this would result in a multi-valued attribute type. Hence, we need to create a new relation, SUPERVISION, with two foreign keys, Supervisor and Supervisee, which both refer to SSN in EMPLOYEE:

EMPLOYEE(SSN, ename, address)
SUPERVISION(*Supervisor*, *Supervisee*)

Since both foreign keys make up the primary key, they cannot be NULL.

All four cardinalities are perfectly supported (see Figure 6.24). Emma Lucas and John Smith are not supervising anyone, and Dan Kelly is not being supervised by anyone (both minimum cardinalities = 0). Paul Barker and Dan Kelly supervise two employees each (maximum cardinality N) and John Smith is being supervised by both Paul Barker and Dan Kelly (maximum cardinality M).

Note, however, that if one or both minimum cardinalities had been 1, then the relational model would have essentially stayed the same such that it could not accommodate this. Hence, these minimum cardinalities would again have to be enforced by the application programs, which is not an efficient solution.

EMPLOYEE(SSN, ename, address)

511	John Smith	14 Avenue of the Americas, New York
289	Paul Barker	208 Market Street, San Francisco
356	Emma Lucas	432 Wacker Drive, Chicago
412	Dan Kelly	668 Strip, Las Vegas

SUPERVISION(*Supervisor*, *Supervisee*)

289	511
289	356
412	289
412	511

Figure 6.24 Example tuples for mapping a unary N:M relationship type.

6.3.2.5 Mapping n-ary Relationship Types

To map an *n*-ary relationship type, we first create relations for each participating entity type. We then also define one additional relation R to represent the *n*-ary relationship type and add foreign keys referring to the primary keys of each of the relations corresponding to the participating entity types. The primary key of R is the combination of all foreign keys which are all NOT NULL. Any attribute type of the *n*-ary relationship can also be added to R. Let's illustrate this with an example.

The relationship type BOOKING (Figure 6.25) is a ternary relationship type between TOURIST, BOOKING, and TRAVEL AGENCY. It has one attribute type: price. The relational model has relations for each of the three entity types together with a relation BOOKING for the relationship type:

TOURIST(TNR, …)
TRAVEL_AGENCY(ANR, …)
HOTEL(HNR, …)
BOOKING(*TNR*, *ANR*, *HNR*, price)

The primary key of the BOOKING relation is the combination of the three foreign keys, as illustrated. It also includes the price attribute. All six cardinalities are perfectly represented in the relational model.

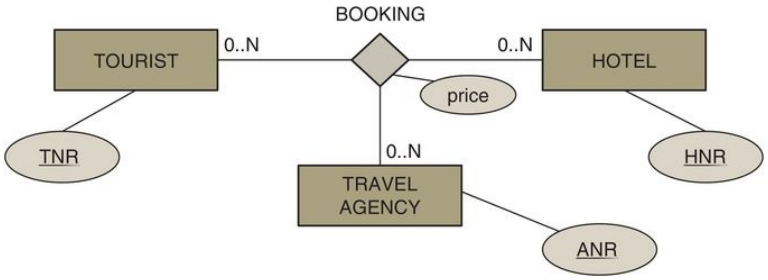


Figure 6.25 Mapping *n*-ary relationship types to the relational model.

The relationship type OFFERS (Figure 6.26) is a ternary relationship type between Instructor, Course, and Semester. An instructor offers a course during zero to N semesters. During a semester, a course should be offered by at least one and at most N instructors. During a semester, an instructor can offer zero to N courses. As with the previous example, the relational model has one relation per entity type and one relation for the relationship type:

INSTRUCTOR(INR, …)
COURSE(CNR, …)
SEMESTER(SEM-YEAR, …)
OFFERS(*INR*,*CNR*,*SEM-YEAR*)

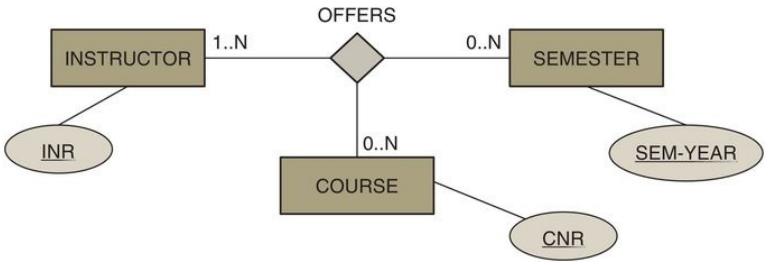


Figure 6.26 Mapping *n*-ary relationship types to the relational model.

Let’s have a look at the cardinalities. Figure 6.27 shows some example tuples. Note that course number 110, Analytics, is not offered during any semester. Some other courses are not offered in all semesters. Hence, the minimum cardinality of 1, stating that during a semester a course should be offered by at least one instructor, cannot be guaranteed by the relational model.

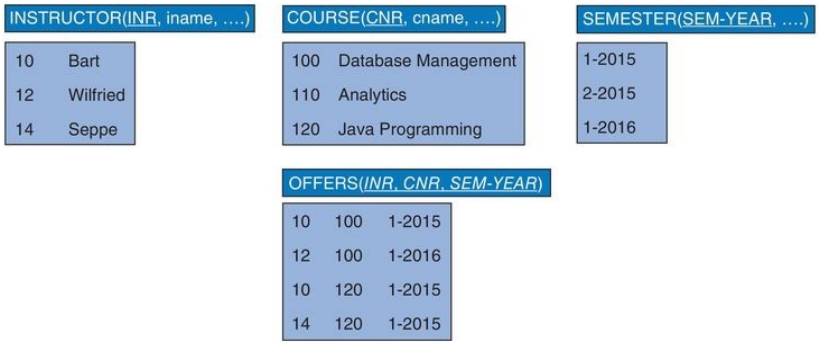


Figure 6.27 Example tuples for mapping an *n*-ary relationship type.

6.3.3 Mapping Multi-Valued Attribute Types

For each multi-valued attribute type, we create a new relation R. We put the multi-valued attribute type in R together with a foreign key referring to the primary key of the original relation. Multi-valued composite attribute types are again decomposed into their components. The primary key can then be set based upon the assumptions.

Let's say we have a multi-valued attribute type phone number (Figure 6.28). An employee can have multiple phones. We create a new relation EMP-PHONE:

EMPLOYEE(SSN, ename, address)
EMP-PHONE(PhoneNr, *SSN*)

It has two attribute types: PhoneNr and SSN. The latter is a foreign key referring to the EMPLOYEE relation. If we assume that each phone number is assigned to only one employee, then the attribute type PhoneNr suffices as a primary key of the relation EMP-PHONE.

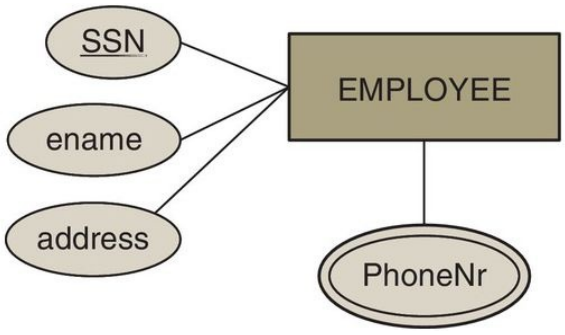


Figure 6.28 Mapping multi-valued attribute types to the relational model.

Let's now change the assumption such that a phone number can be shared by multiple employees. Hence, PhoneNr is no longer appropriate as a primary

key of the relation. Also, SSN cannot be assigned as a primary key since an employee can have multiple phone numbers. Hence, the primary key becomes the combination of both PhoneNr and SSN:

EMPLOYEE(SSN, ename, address)
EMP-PHONE(PhoneNr, *SSN*)

Some example tuples are depicted in Figure 6.29, where you can see that tuples 1 and 2 of the EMP-PHONE relation have the same value for PhoneNr, whereas tuples 2 and 3 have the same value for SSN. This example illustrates how the business specifics can help define the primary key of a relation.

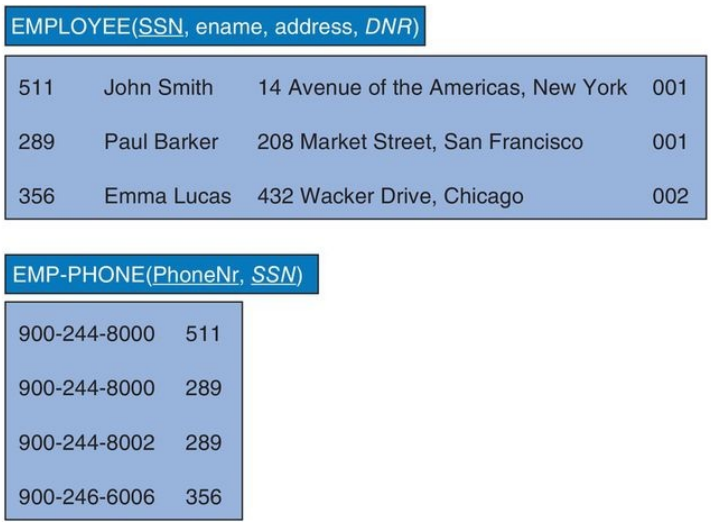


Figure 6.29 Example tuples for mapping a multi-valued attribute type.

6.3.4 Mapping Weak Entity Types

Remember, a weak entity type is an entity type that cannot produce its own key attribute type and is existence-dependent on an owner entity type. It should be mapped into a relation R with all its corresponding attribute types. Next, a foreign key must be added referring to the primary key of the relation corresponding to the owner entity type. Because of the existence dependency, the foreign key is declared as NOT NULL. The primary key of R is then the combination of the partial key and the foreign key.

Figure 6.30 illustrates our earlier example. Room is a weak entity type and needs to borrow HNR from Hotel to define a key attribute type, which is the combination of RNR and HNR.

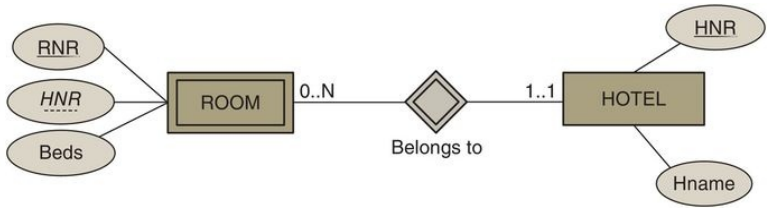


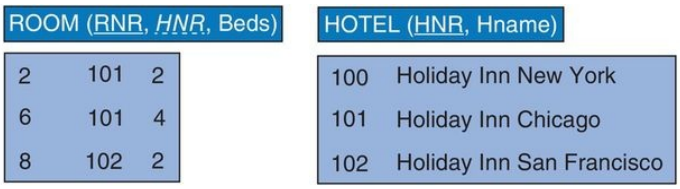
Figure 6.30 Mapping weak entity types to the relational model.

We can map both entity types to the relational model as follows:

Hotel (HNR, Hname)
Room (RNR, *HNR*, beds)

Room has a foreign key, HNR, which is declared as NOT NULL and refers to Hotel. Its primary key is the combination of RNR and HNR.

Some example tuples are depicted in Figure 6.31. All four cardinalities are nicely supported by the relational model.



ROOM (RNR, *HNR*, Beds)		
2	101	2
6	101	4
8	102	2

HOTEL (HNR, Hname)	
100	Holiday Inn New York
101	Holiday Inn Chicago
102	Holiday Inn San Francisco

Figure 6.31 Example tuples for mapping a weak entity type.

6.3.5 Putting it All Together

Up to now we have extensively discussed how to map the ER model to a relational model. Table 6.4 summarizes how the key concepts of both models are related.

Table 6.4 Mapping an ER model to a relational model

ER model	Relational model
Entity type	Relation
Weak entity type	Foreign key
1:1 or 1:N relationship type	Foreign key
M:N relationship type	New relation with two foreign keys
n-ary relationship type	New relation with *n* foreign keys
Simple attribute type	Attribute type
Composite attribute type	Component attribute types
Multi-valued attribute type	Relation and foreign key
Key attribute type	Primary or alternative key

Here you can see the resulting relational model for our employee administration ER model as discussed in Chapter 3:

- EMPLOYEE (SSN, ename, streetaddress, city, sex, dateofbirth, MNR, DNR)
  - – MNR foreign key refers to SSN in EMPLOYEE, NULL ALLOWED
  - – DNR foreign key refers to DNR in DEPARTMENT, NOT NULL
- DEPARTMENT (DNR, dname, dlocation, MGNR)
  - – MGNR: foreign key refers to SSN in EMPLOYEE, NOT NULL
- PROJECT (PNR, pname, pduration, DNR)
  - – DNR: foreign key refers to DNR in DEPARTMENT, NOT NULL
- WORKS_ON (SSN, PNR, HOURS)
  - – SSN foreign key refers to SSN in EMPLOYEE, NOT NULL
  - – PNR foreign key refers to PNR in PROJECT, NOT NULL

Let’s briefly discuss it. The primary key of the EMPLOYEE relation is SSN. It has two foreign keys: MNR, which refers to SSN and implements the recursive SUPERVISED BY relationship type; and DNR, which refers to DEPARTMENT and implements the WORKS_IN relationship type. The former is NULL ALLOWED, whereas the latter is not. The primary key of DEPARTMENT is DNR. It has one foreign key MGNR which refers to SSN in EMPLOYEE and implements the MANAGES relationship type. It cannot be NULL. The primary key of PROJECT is PNR. The foreign key DNR refers to DEPARTMENT. It implements the IN CHARGE OF relationship type and cannot be NULL. The WORKS_ON relation is needed to implement the M:N relationship type between EMPLOYEE and PROJECT. Its primary key is made

up of two foreign keys referring to EMPLOYEE and PROJECT respectively. It also includes the relationship type attribute HOURS representing how many hours an employee worked on a project.

Our relational model is not a perfect mapping of our ER model. Some of the cardinalities have not been perfectly translated. More specifically, we cannot guarantee that a department has at minimum one employee (not counting the manager). Another example is that the same employee can be a manager of multiple departments. Some of the earlier-mentioned shortcomings for the ER model still apply here. We cannot guarantee that a manager of a department also works in the department. We also cannot enforce that employees should work on projects assigned to departments to which they belong.

Retention Questions

- Illustrate how an ER entity type can be mapped to the relational model.
- Illustrate how ER relationship types with varying degrees and cardinalities can be mapped to the relational model. Discuss the loss of semantics where appropriate.

6.4 Mapping a Conceptual EER Model to a Relational Model

The EER model builds upon the ER model by introducing additional modeling constructs such as specialization, categorization, and aggregation (see Chapter 3). In this section we discuss how these can be mapped to the relational model.

6.4.1 Mapping an EER Specialization

EER specializations can be mapped in various ways. A first option is to create a relation for the superclass and each subclass, and link them with foreign keys. An alternative is to create a relation for each subclass and none for the superclass. Finally, we can create one relation with all attribute types of the superclass and subclasses and add a special attribute type. Let's explore these options in more detail, with some examples.

Figure 6.32 shows an EER specialization of ARTIST into SINGER and ACTOR. The specialization is partial since not all artists are either a singer or an actor. It also has overlap, since some singers can also be actors. An artist has an artist number and an artist name. A singer has a music style.

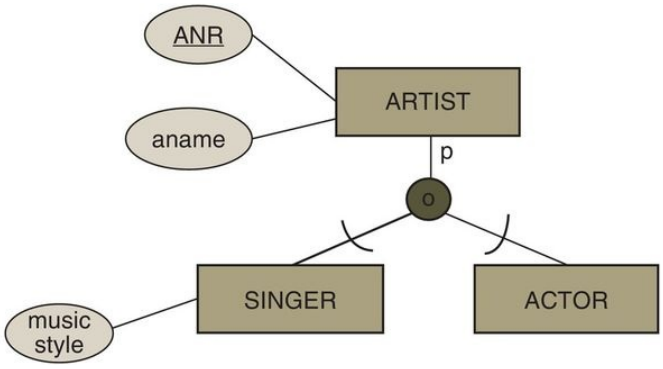


Figure 6.32 Example EER specialization with superclass ARTIST and subclasses SINGER and ACTOR.

In our first option (option 1), we create three relations: one for the superclass and two for the subclasses:

ARTIST(ANR, aname, …)
SINGER(*ANR*, music style, …)
ACTOR(*ANR*, …)

We add a foreign key ANR to each subclass relation that refers to the superclass relation. These foreign keys then also serve as primary keys.

Figure 6.33 illustrates option 1 with some example tuples. This solution works well if the specialization is partial. Not all artists are included in the subclass relations. For example, Claude Monet is only included in the superclass relation and not referred to in any of the subclass relations. In the case that the specialization had been total instead of partial, then we could not have enforced it with this solution. The overlap characteristic is also nicely modeled. You can see that Madonna is referenced both in the SINGER and ACTOR relations. If the specialization had been disjoint instead of overlap, then again we could not have enforced it with this solution.

Figure 6.33 Example tuples for option 1.

Let’s now change the specialization to total instead of partial. In other words, we assume all artists are either singers or actors. Option 1 would not work well for this since there could be ARTIST tuples which are not referenced in either the SINGER or ACTOR relation. In this case, a better option (option 2) to map this EER specialization only creates relations for the subclasses as follows:

SINGER(ANR, aname, music style, …)
ACTOR(ANR, aname, …)

The attribute types of the superclass have been added to each of the subclass relations.

Figure 6.34 illustrates some example tuples for option 2. This solution only works for a total specialization. The overlap characteristic can also be supported. You can see that Madonna is included in both relations. Note, however, that this creates redundancy. If we would also store her biography, picture, etc., then this information needs to be added to both relations, which is not very efficient from a storage perspective. This approach cannot enforce a specialization to be disjoint since the tuples in both relations can overlap.

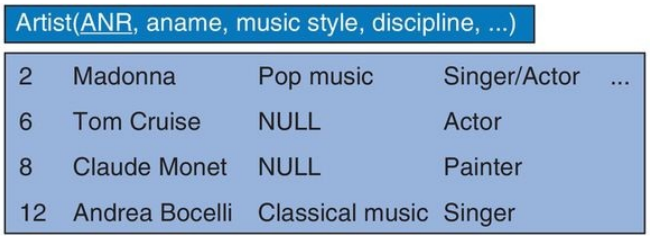
Figure 6.34 Example tuples for option 2.

Another option (option 3) is to store all superclass and subclass information into one relation:

ARTIST(ANR, aname, music style, …, discipline)

An attribute type discipline is then added to the relation to indicate the subclass.

Figure 6.35 shows some example tuples for option 3. The values that can be assigned to the attribute type discipline depend upon the characteristics of the specialization. Hence, all specialization options are supported. Note that this approach can generate a lot of NULL values for the subclass-specific attribute types (music style in our case).



Artist(ANR, aname, music style, discipline, ...)

2	Madonna	Pop music	Singer/Actor	...
6	Tom Cruise	NULL	Actor	
8	Claude Monet	NULL	Painter	
12	Andrea Bocelli	Classical music	Singer	

Figure 6.35 Example tuples for option 3.

In a specialization lattice, a subclass can have more than one superclass as you can see illustrated in Figure 6.36. A PhD student is both an employee and a student. This can be implemented in the relational model by defining three relations: EMPLOYEE, STUDENT, and PHD-STUDENT:

EMPLOYEE(SSN, …)
STUDENT(SNR, …)
PHD-STUDENT(*SSN*, *SNR*, …)

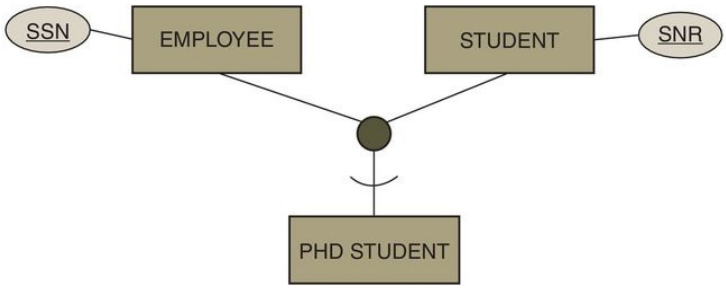


Figure 6.36 Example of an EER specialization lattice.

The primary key of the latter is a combination of two foreign keys referring to EMPLOYEE and STUDENT, respectively. This solution does not support total specialization, since we cannot enforce that all employee and student tuples are referenced in the PHD-STUDENT relation.

6.4.2 Mapping an EER Categorization

Another extension provided by the EER model is the concept of a categorization. As shown in Figure 6.37, the category subclass is a subset of the union of the entities of the superclasses.

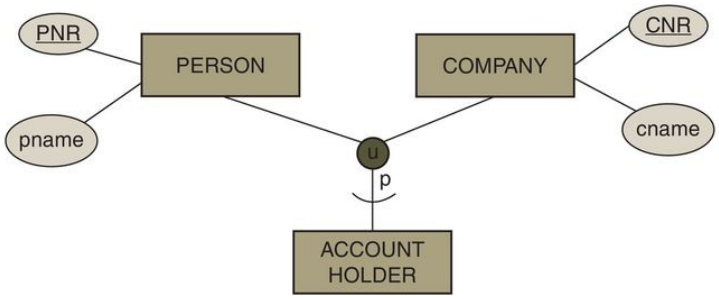


Figure 6.37 Example of an EER categorization with superclasses PERSON and COMPANY, and category ACCOUNT HOLDER.

Therefore, an account holder can be either a person or a company. This can be implemented in the relational model by creating a new relation ACCOUNT-HOLDER that corresponds to the category and adding the corresponding attribute types to it as follows:

PERSON(PNR, …, *CustNo*)
COMPANY(CNR, …, *CustNo*)
ACCOUNT-HOLDER(CustNo, …)

We then define a new primary key attribute, CustNo, also called a surrogate key, for the relation that corresponds to the category. This surrogate key is then added as a foreign key to each relation corresponding to a superclass of the category. This foreign key is declared as NOT NULL for a total categorization and NULL ALLOWED for a partial categorization. In the case that the

superclasses happen to share the same key attribute type, this one can be used and there is no need to define a surrogate key.

This is illustrated in Figure 6.38. In our case the categorization is partial since Wilfried and Microsoft are not account holders, hence the NULL values. This solution is not perfect: we cannot guarantee that the tuples of the category relation are a subset of the union of the tuples of the superclasses. As an example, customer number 12 in the ACCOUNT-HOLDER relation does not appear in either the PERSON or the COMPANY relation. Moreover, we cannot avoid that a tuple in the PERSON relation and a tuple in the COMPANY relation would have the same value for CustNo, which means they would refer to the same ACCOUNT-HOLDER tuple. In that case, this account holder would be a person and a company at the same time, which is incorrect as well.

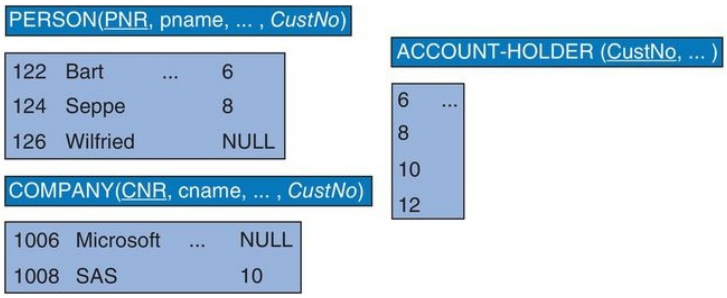


Figure 6.38 Example tuples for mapping a categorization.

6.4.3 Mapping an EER Aggregation

Aggregation is the third extension provided by the EER model. In Figure 6.39 we have aggregated the two entity types CONSULTANT and PROJECT and their relationship type into an aggregate called PARTICIPATION. This aggregate has an attribute type date and participates in a 1:M relationship type with the entity type CONTRACT.

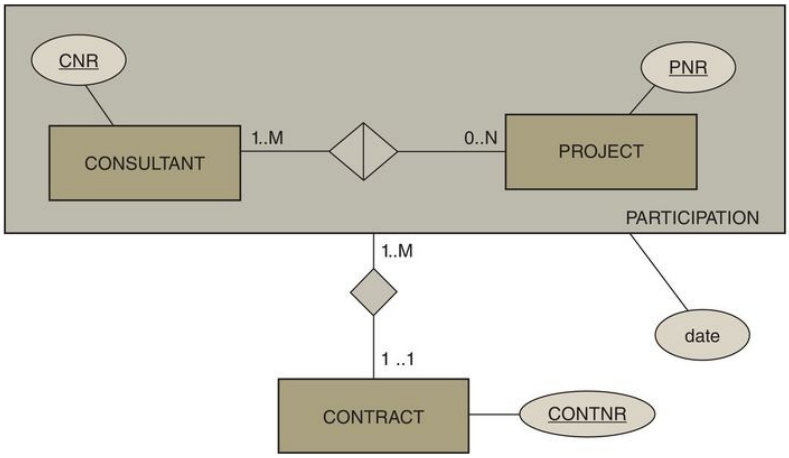


Figure 6.39 Example of an EER aggregation.

This can be implemented in the relational model by creating four relations: CONSULTANT, PROJECT, PARTICIPATION, and CONTRACT:

CONSULTANT(CNR, …)
PROJECT(PNR, …)
PARTICIPATION(*CNR*, *PNR*, *CONTNR*, date)
CONTRACT(CONTNR, …)

The PARTICIPATION relation models the aggregation. Its primary key is a combination of two foreign keys referring to the CONSULTANT and PROJECT relations. It includes a NOT NULL foreign key to the CONTRACT relation to model the relationship type. It also includes the attribute type date.

Retention Questions

- Discuss how the EER concepts of specialization, categorization, and aggregation can be mapped to the relational model. Illustrate with examples and clarify what semantics may get lost in the mapping.

Summary

In this chapter we have discussed the relational model as one of the most popular data models used in the industry today. After formally introducing its basic building blocks, we elaborated on different types of keys. Next, we reviewed various relational constraints that ensure the data in the relational database have the desired properties. Normalization was extensively covered. First, we illustrated the need to guarantee no redundancy or anomalies in the data model. Functional dependencies and prime attribute types were introduced as important concepts during the normalization procedure, which brings the data model into the first normal form, second normal form, third normal form, Boyce–Codd normal form, and fourth normal form. We concluded by discussing how both ER and EER conceptual data models can be mapped to a logical relational model. We extensively discussed the semantics that get lost during the mapping by using plenty of examples. In the next chapter, we zoom into Structured Query Language (SQL), which is the DDL and DML of choice for relational databases.

Scenario Conclusion

Following the mapping procedure outlined in this chapter, the EER conceptual data model for Sober can be mapped to the following logical relational model (primary keys are underlined; foreign keys are in italics):

- **CAR** (CAR-NR, CARTYPE)
- **SOBER CAR** (*S-CAR-NR*)

    - FOREIGN KEY S-CAR-NR refers to CAR-NR in CAR; NULL NOT ALLOWED
- **OTHER CAR** (*O-CAR-NR*, *O-CUST-NR*)
    - FOREIGN KEY O-CAR-NR refers to CAR-NR in CAR; NULL NOT ALLOWED
    - FOREIGN KEY O-CUST-NR refers to CUST-NR in CUSTOMER; NULL NOT ALLOWED
- **ACCIDENT** (ACC-NR, ACC-DATE-TIME, ACC-LOCATION)
- **INVOLVED** (*I-CAR-NR*, *I-ACC-NR*, DAMAGE AMOUNT)
    - FOREIGN KEY I-CAR-NR refers to CAR-NR in CAR; NULL NOT ALLOWED
    - FOREIGN KEY I-ACC-NR refers to ACC-NR in ACCIDENT; NULL NOT ALLOWED
- **RIDE** (RIDE-NR, PICKUP-DATE-TIME, DROPOFF-DATE-TIME, DURATION, PICKUP-LOC, DROPOFF-LOC, DISTANCE, FEE, *R-CAR-NR*)
    - FOREIGN KEY R-CAR-NR refers to CAR-NR in CAR; NULL NOT ALLOWED
- **RIDE HAILING** (*H-RIDE-NR*, PASSENGERS, WAIT-TIME, REQUEST-TYPE, *H-CUST-NR*)
    - FOREIGN KEY H-RIDE-NR refers to RIDE-NR in RIDE; NULL NOT ALLOWED
    - FOREIGN KEY H-CUST-NR refers to CUST-NR in CUSTOMER; NULL NOT ALLOWED

- **RIDE SHARING** (*S-RIDE-NR*)
    - FOREIGN KEY S-RIDE-NR refers to RIDE-NR in RIDE; NULL NOT ALLOWED
- **CUSTOMER** (CUST-NR, CUST-NAME)
- **BOOK** (*B-CUST-NR, B-S-RIDE-NR*)
    - FOREIGN KEY B-CUST-NR refers to CUST-NR in CUSTOMER; NULL NOT ALLOWED
    - FOREIGN KEY B-S-RIDE-NR refers to S-RIDE-NR in RIDE SHARING; NULL NOT ALLOWED

The relational model has ten relations. Both EER specializations (RIDE into RIDE HAILING and RIDE SHARING; CAR into SOBER CAR and OTHER CAR) have been mapped using option 1; a separate relation was introduced for the superclass and for each of the subclasses. The reason we chose option 1 is that both superclasses participated in relationship types: RIDE with CAR and CAR with ACCIDENT. Although both specializations are total and disjoint in the EER model, this cannot be enforced in the relational model. Hence, it is possible to have a tuple in the CAR relation which is not referenced in either the SOBER CAR or OTHER CAR relation, which makes the specialization partial. Likewise, it is perfectly possible to have the same CAR referenced both in the SOBER CAR and OTHER CAR relations, which makes the specialization overlap. The four EER cardinalities of the OPERATED BY, LEAD CUSTOMER, and OWNS relationship types can be perfectly mapped to the relational model. The minimum cardinalities of 1 of the EER relationship types BOOK and INVOLVED cannot be enforced in the relational model. For example, it is perfectly

possible to define an accident, by adding a new tuple to ACCIDENT, without any car involved.

Key Terms List

alternative keys

Boyce–Codd normal form (BCNF)

candidate key

deletion anomaly

domain

first normal form (1 NF)

foreign key

fourth normal form (4 NF)

full functional dependency

functional dependency

insertion anomaly

multi-valued dependency

normalization

primary key

prime attribute type

relation

relational model

second normal form (2 NF)

superkey

third normal form (3 NF)

transitive dependency

trivial functional dependency

tuple

update anomaly

Review Questions

6.1. Consider the following (normalized) relational model (primary keys are underlined, foreign keys are in italics).

EMPLOYEE(SSN, ENAME, EADDRESS, SEX, DATE_OF_BIRTH, *SUPERVISOR*, *DNR*)

SUPERVISOR: *foreign key refers to* SSN in EMPLOYEE, *NULL value allowed*

DNR: *foreign key refers to* DNR in DEPARTMENT, *NULL value not allowed*

DEPARTMENT(DNR, DNAME, DLOCATION, *MGNR*)

MGNR: *foreign key refers to* SSN in EMPLOYEE, *NULL value not allowed*

PROJECT(PNR, PNAME, PDURATION, *DNR*)