Solutions to Selected Exercises

1 Solutions to Chapter 2 Exercises

2-1. We have only one entity, called **Person** (everything else is represented as attributes), with the following attributes: Id (the identifier); Name, which is composed of one or more given names and one or more family names; One or more aliases, an address (composed of street, city, state,zip); date-of-birth; and age, which can be calculated from the date of birth.

This example illustrates most of the different kinds of attributes, plus the fact that the representation of a given attribute depends on the situation; here name is NOT divided into first,middle,last.

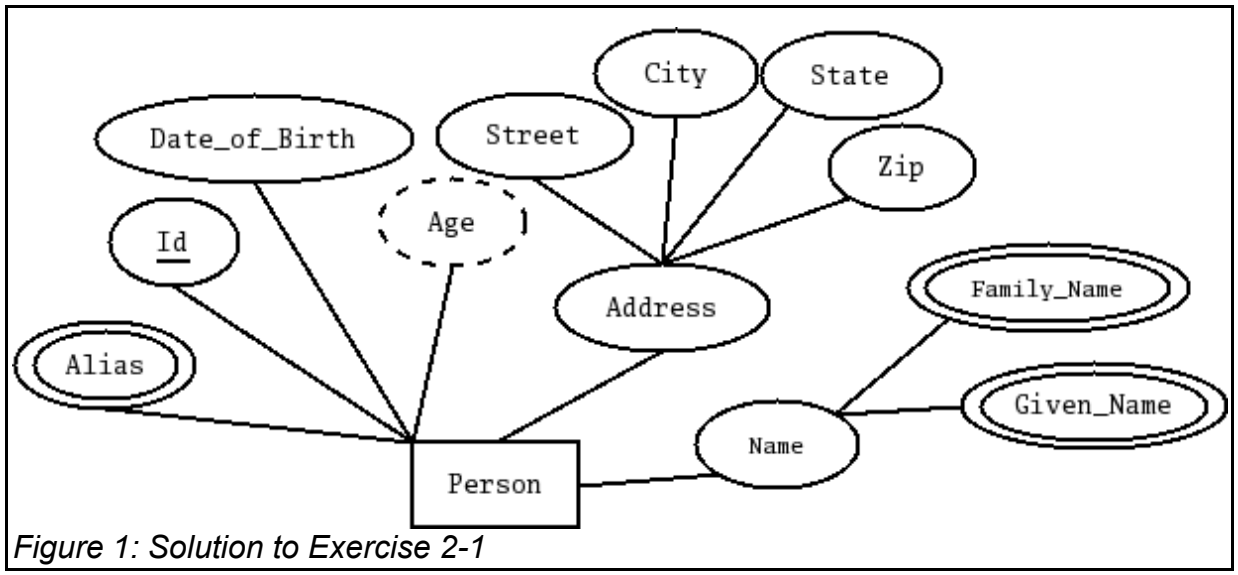


Figure 1: Solution to Exercise 2-1

2-2. We want to model a CD as an entity. The only entity is the CD and everything else is modeled as attributes.

- A CD has a number, which is its identifier.
- A CD has a title.
- A CD has zero or more performers; for each performer we keep their first and last name.
- A CD has zero or more songs. For each song we keep its title and the name of its author (with name divided into first and last)

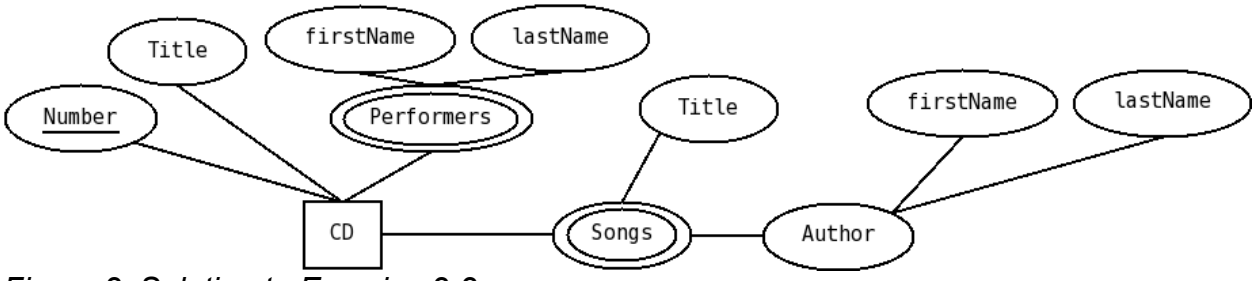


Figure 2: Solution to Exercise 2-2

2-3. Now model a CD, but with Person and Song as entities. So the requirements are as follows:

- A CD has a number, which is its identifier, and a title.
- A person has an id, and a name, divided in first and last.
- A song has an id, a title and a length.
- We keep track of which person is a song's author. A person can author many songs and a song has exactly one author.
- We keep track of which people perform on a CD. Zero or more people perform on a CD and people can perform on zero or more CDs.
- We keep track of which songs are included on a CD. One or more songs are included on a CD, and a song is included in one or more CDs.

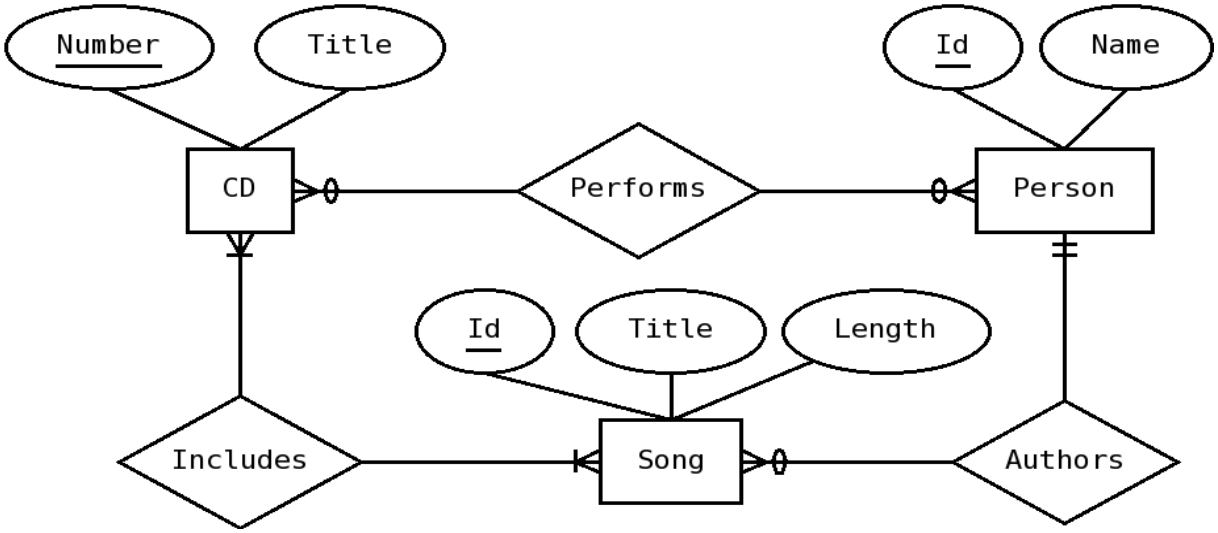


Figure 3: Solution to Exercise 2-3

2-4. We want to keep track of **courses**, **sections** and **professors**. A course has a course number (identifier), and a title. A section has a crn (identifier), semester, and year. A section belongs to exactly one course, and a course may have zero or more sections. A Professor has ssn (identifier) and name. A professor teaches one or more sections (a section is taught by exactly one professor). A professor is qualified to teach one or more courses (a course may have one or more qualified professors). We also want to keep track of the date a professor became qualified to teach the course.

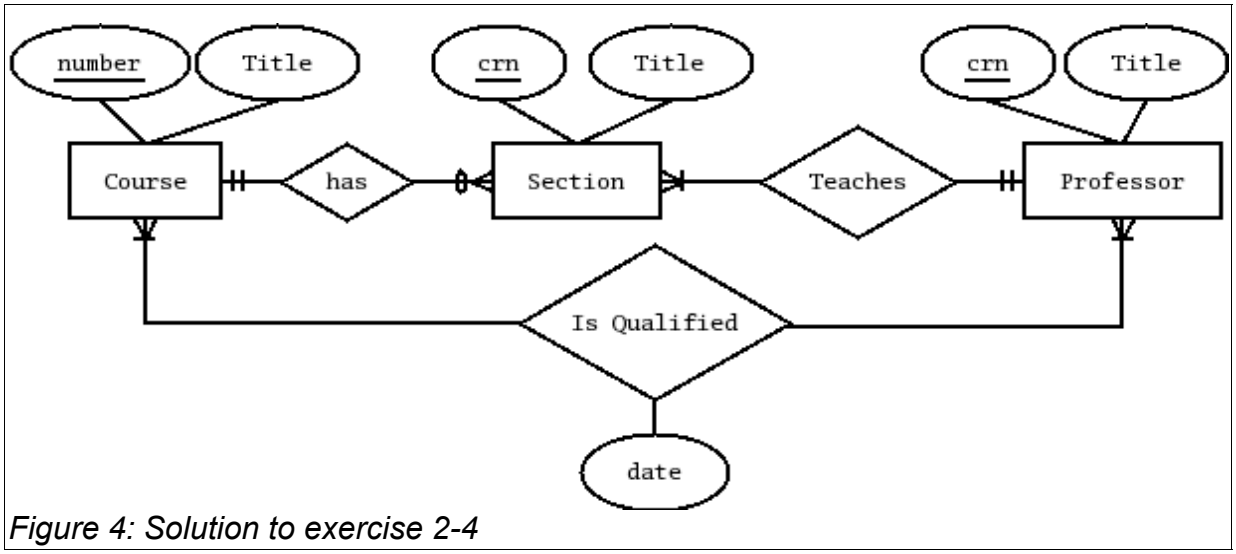


Figure 4: Solution to exercise 2-4

2-5. Answer to 2-5 not provided.

2-6. Answer to 2-6 not provided.

2-7. We have two kinds of entities: **Products** and **Categories**, both with attributes id (the identifier), and name. Each product belongs to zero or more categories and each category can have zero or more products. Categories are organized in a hierarchy; each category belongs to zero or one categories, and may have zero or more sub-categories.

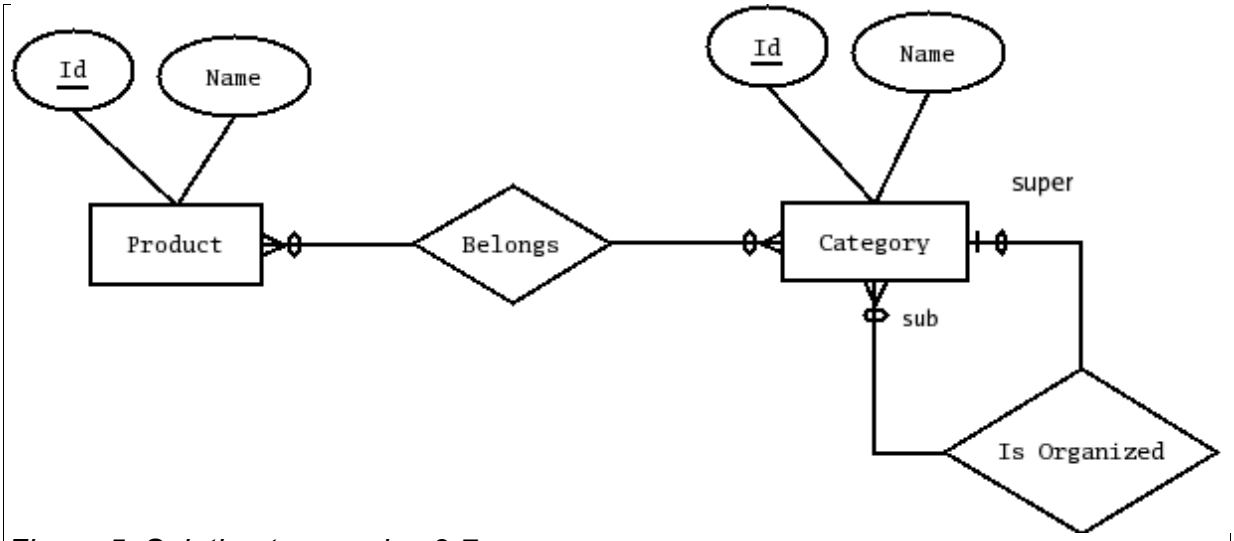


Figure 5: Solution to exercise 2-7

2-8. We want to model **course** and **program** information for a university. For each course we keep its id, its title, the number of lecture hours, the number of lab hours, the number of credit hours (which is always equal to the number of lecture hours plus one-half of the number of lab hours). Courses may have other courses as prerequisites. We also want to keep information about programs of study. For each program we want to keep its id and its title. We also want to keep track of which courses are required for a program of study, and which courses are allowed as electives for a program of study.

This one, could be easily modeled in two different ways. The first solution uses two different relationships between program and course; **requires**, which represents the fact that a course is required for a program and uses as elective, which represents the fact that a program uses a course as an elective. Notice the cardinalities for the relationships weren't really specified in the requirements, but zero-or-more sounds reasonable.

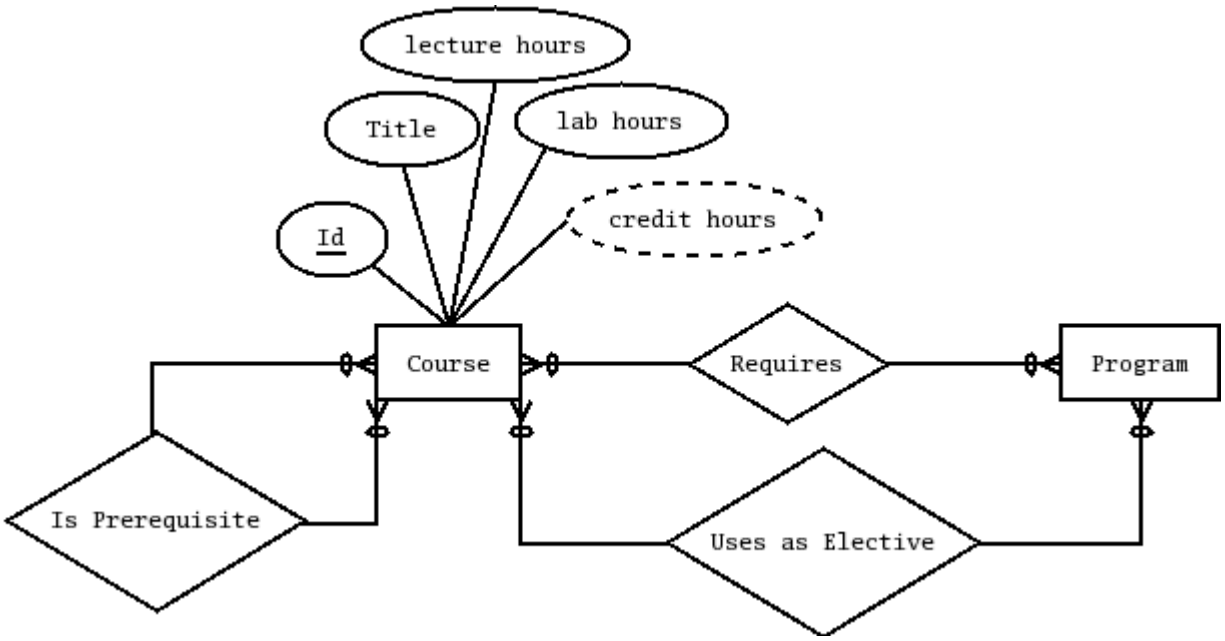


Figure 6: Solution to Exercise 2-8, with two different relationships between program and course

The second solution uses only one relationship, with an attribute to distinguish between required and elective courses. Notice this would convey less information if the cardinalities for requires and 'Uses As Elective' were different.

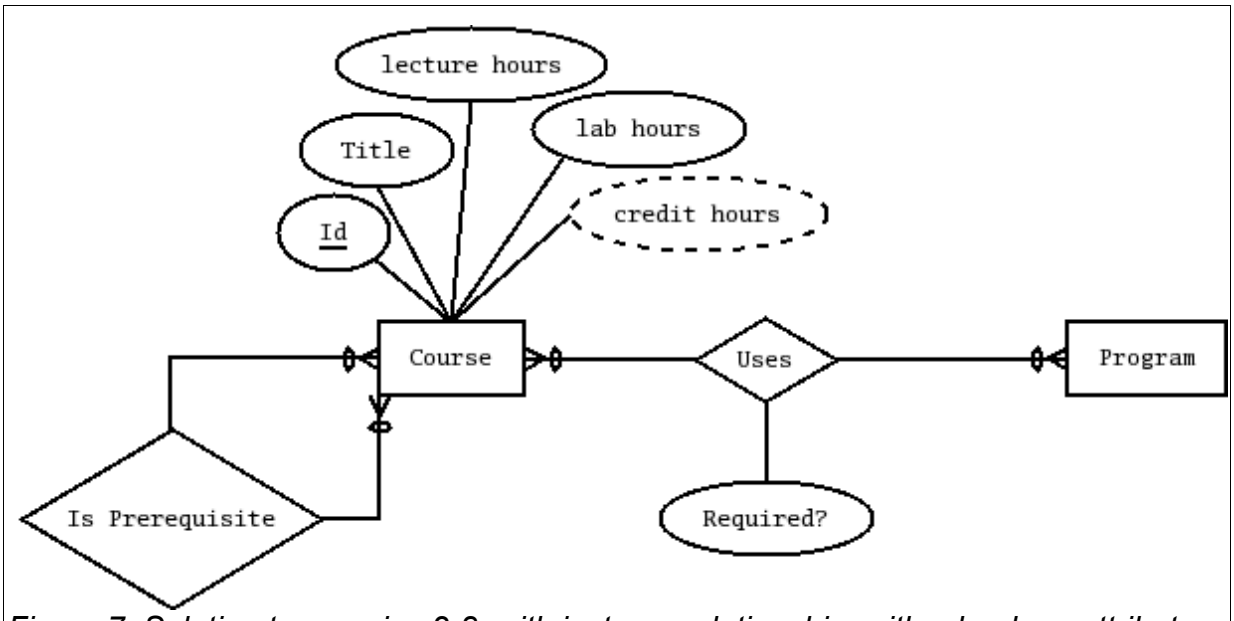


Figure 7: Solution to exercise 2-8, with just one relationship, with a boolean attribute of the relationship indicating whether the program uses the course as a requirement or an elective

2 Solutions to Chapter 7 Exercises

1. We have a relation with fields A,B,C,D,E,F and its primary key is {A,B,C}; mark whether the following dependencies would be *partial*, *transitive*, or *full (OK)*.

Dependency	Status (partial, transitive, full)
A -> F	Partial (A is part of the PK, but not the full PK)
A,B -> E,F	Partial (A is part of the PK, but not the full PK)
E -> F	Transitive (E is not part of the PK)
A,B,C->E	Full (A,B,C is the PK)

2. No solution provided for exercise 2.
3.
4. For the following relation, and functional dependencies, indicate the normal form it is in (3 points). If it is not in 3NF, decompose it into 3NF relations.

Team

SportId	SportName	SchoolId	SchoolName	TeamName

Other Dependencies:

a. SportId -> SportName
b. SchoolId -> SchoolName

We can see that both of this dependencies are partial, because their **determinants** (SportId and SchoolId respectively) are pieces of the primary key. So, since there are partial dependencies, the relational schema is NOT even in 2NF (second normal form), which means it is in 1NF (first normal form)

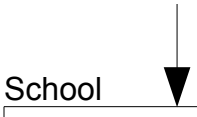
Now, to transform this schema to 3NF, we need to eliminate the offending dependencies. Lets start with the first one, SportId -> SportName. We need to:

1. Create a new table, containing both attributes with the determinant (SportId) as the primary key (we can call this table School)
2. Eliminate the determined attributes (SportName) from the original table
3. Add a foreign key constraint from the original table to this new table

The resulting schema looks as follows:

Team

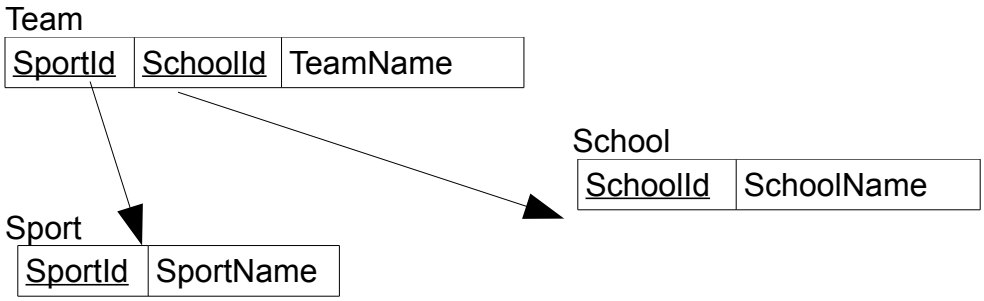
SportId	~~SportName~~	SchoolId	SchoolName	TeamName



School

SchoolId	SchoolName

Now, this still leaves us with one partial dependency, SchoolId -> SchoolName; after a similar transformation (and completely deleting the unneede attributes from the original table, rather than just striking through), we end up with the following relational schema diagram.


Team
SportId
SchoolId
TeamName
School
SchoolId
SchoolName
Sport
SportId
SportName