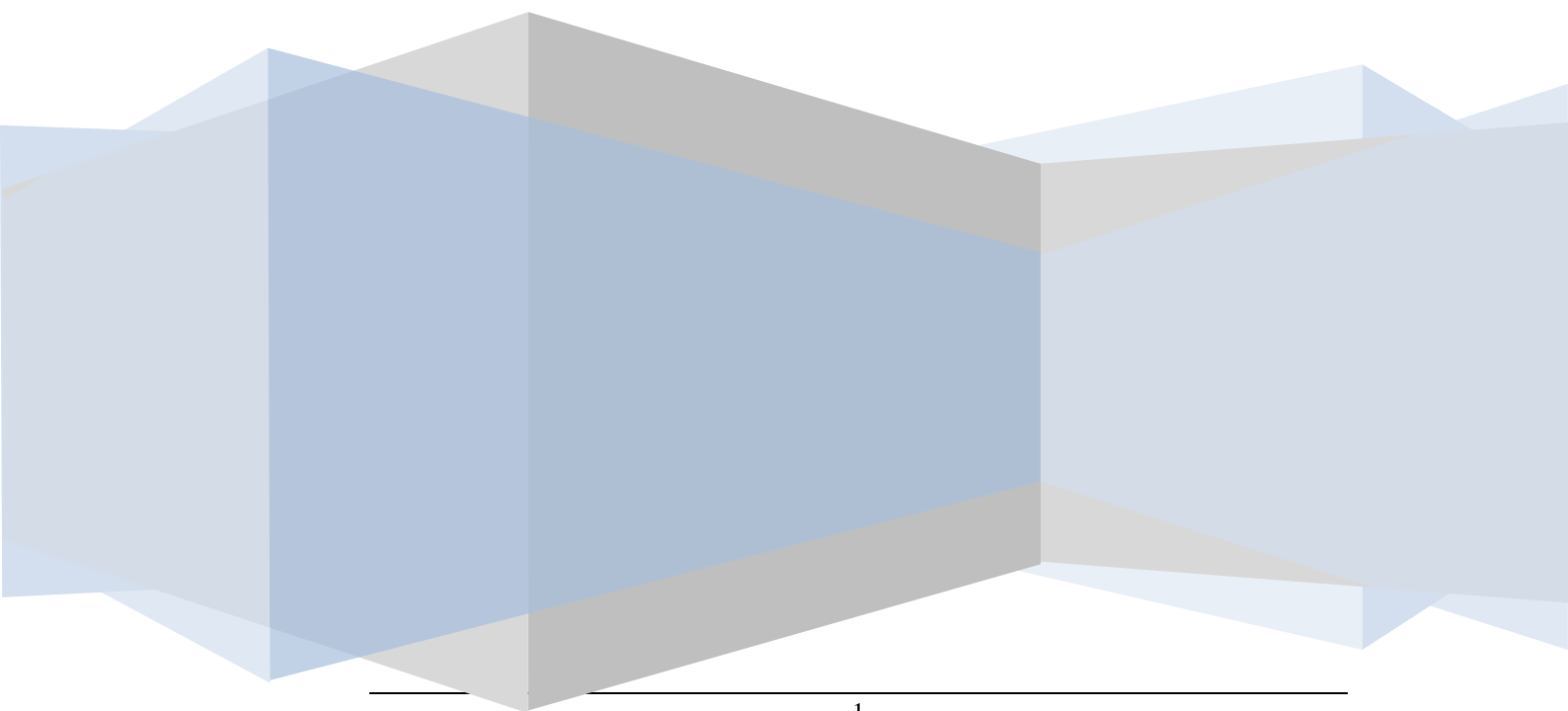


Inholland University of Applied Sciences

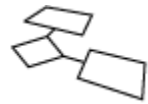
Data modeling 1

Reader





1	INTRODUCTION DATABASES	3
1.1	INTRODUCTION.....	3
1.2	DATABASES.....	3
1.2.1	<i>Why a database?</i>	3
1.2.2	<i>The structure of a database</i>	3
1.2.3	<i>Concepts when working with databases</i>	5
1.2.4	<i>Keys</i>	5
1.2.5	<i>Referential integrity</i>	6
2	THE ENTITY RELATIONSHIP DIAGRAM.....	7
2.1	INTRODUCTION.....	7
2.2	SYMBOLS OF AN ERD	7
2.2.1	<i>Entities</i>	7
2.2.2	<i>Relationships</i>	8
2.2.3	<i>Attributes</i>	8
2.3	CARDINALITY.....	9
2.3.1	<i>Functionality</i>	9
2.3.2	<i>Totality</i>	11
2.4	HISTORICAL AND UP-TO-DATE DATABASES.....	12
2.5	EXAMPLE OF AN ERD	12
2.6	SPECIAL RELATIONSHIPS.....	13
2.6.1	<i>Generalisation</i>	13
2.6.2	<i>Specialisation</i>	14
2.6.3	<i>Multidimensional Relationships</i>	15
2.6.4	<i>Relationships that connect an entity to itself</i>	15
3	FROM ERD TO RELATIONAL MODEL.....	17
3.1	CONDITIONS	17
3.2	CONVERTING RELATIONSHIPS	17
3.3	CONVERTING TOTALITY.....	20
4	DESIGNING AN ERD	25
4.1	INTRODUCTION.....	25
4.2	RELEVANCE.....	25
4.2.1	<i>Relevance of entities</i>	25
4.2.2	<i>Relevance of relationships</i>	26
4.3	AGAIN: ATTRIBUTES TO RELATIONSHIPS	27
4.4	ONCE AGAIN: N-TO-M RELATIONSHIPS	27
4.5	AGAIN: MULTIDIMENSIONAL RELATIONSHIPS	28
4.6	PITFALLS	30
4.6.1	<i>Entities that are actually attributes</i>	30
4.6.2	<i>Attributes that are actually entities</i>	30
4.6.3	<i>The relationship is_a</i>	31
4.6.4	<i>Attribute lists</i>	31
4.6.5	<i>Process data</i>	31
 ASSIGNMENTS	
	33
	ENTITIES, RELATIONSHIPS, AND ATTRIBUTES.....	33
	CARDINALITY	34
	COMPLETE ER DIAGRAMS	34
	SPECIAL RELATIONSHIPS	35
	TRANSLATING INTO RELATIONAL MODEL.....	37
	DESIGN ER DIAGRAMS	39
	ADDITIONAL ASSIGNMENTS	42



1

Introduction databases

1.1 Introduction

An organisation is a partnership of people that is founded with a certain purpose. This could, for example, be a sports club, a hospital, a articleion company or a school. To achieve the goal of the organisation, different resources are needed in addition to people. Think for example of money, machines, buildings, good infrastructure and knowledge. People are becoming increasingly aware that information is also an important tool in the pursuit of the organisational goals. The information needed must be in the right place at the right time for the organisation to function properly.

process and data-
oriented
approach

As information plays an increasingly important role, the need for data is growing enormously. We can look at data processing within an organisation in two ways. We can look at how the data is "flowing" through the organisation and for which activities they are being used; the *process-oriented* approach, or we can look at the way the data is stored; the *data-oriented* approach. In this reader we will focus on the data-oriented approach and designing how data is going to be stored.

1.2 Databases

1.2.1 Why a database?

file

The computer is increasingly used for the storage of data. When automation was still in its infancy, in the 1950s, data was stored separately in separate *files*. The sales department had separate files for customer data and article details, the purchasing department had its own supplier and article files etc. This way of storing data can lead to a number of problems. Because the same data is stored separately in multiple places, there is a risk that the data in one file will no longer match the data in another file. For example, a purchasing officer may remove an article from the article file and this change will not be made to the sales department's article file. In this situation the article could still be sold, while it actually has been removed from the catalogue.

database

In the end, this issue was resolved by centrally managing all data and making it available to all users. Instead of, for example, a separate article file for each department, a single article file was kept centrally, which all departments involved could use. Such a collection of centrally managed data is called a *database*.

1.2.2 The structure of a database

When building a database, we could choose to put all the data on one big pile and manage it that way. Then, a file of sales data could look like figure 1.1.

**SALES**

Customer name	Customer address	Date	Ordered article	Quantity
P.Okko	Parkway 19	12-06-2002	Sports bike 230Z	2
P.Okko	Parkway 19	12-06-2002	Tile AB2	1
A.Bangbuck	Fieldlane 2	20-06-2002	Tile AB2	4
A.Bangbuck	Fieldlane 2	20-12-2002	Tile AB2	1
P.Okko	Parkway 19	01-12-2002	Sports bike 230Z	36
...

► **Figure 1.1** An example of a sales data file

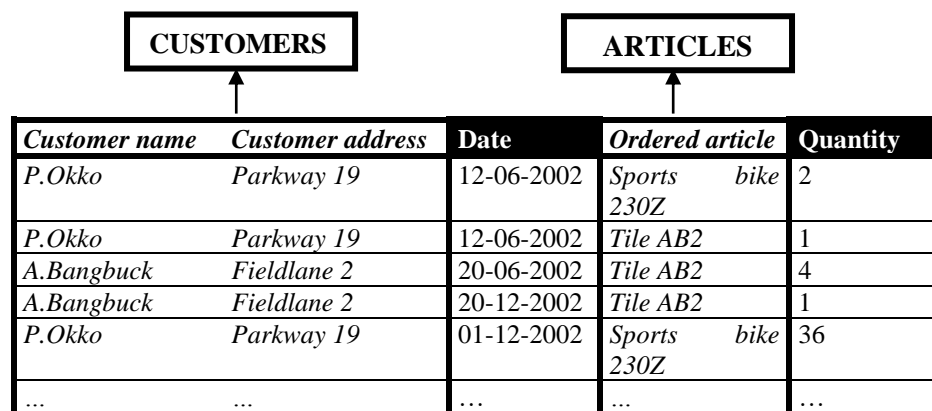
redundancy

Immediately noticeable is that the same data appears more than once in this sales file. For example, the SPORTS BIKE 230Z article occurs in this example twice, and the TILE AB2 article three times. The customer names and customer addresses are also repeated. The occurrence of the same data multiple times is what we call *redundancy*. Redundancy involves a number of problems.

Of course, storing the same data multiple times costs a lot of (disk) space. In addition, it does not improve the reliability of the data. Because the sales representative has to re-enter the customer's name and address when entering each sales order, chances are that a mistake will be made one time and P.OKKA is typed, for example. As a result, an article is accidentally sold to a customer who does not exist at all. Moreover, the file is difficult to maintain in this way. For example, suppose that customer P.OKKO moves from PARKWAY 19 to PERIODSQUARE 187B. In every place where this customer appears in the sales file (maybe a few thousand times), the customer address will now have to be changed, with of course a great chance of errors.

relational model

To avoid redundancy, before we start building, we will have to think carefully about how we are going to set up the database. Over time, several solutions have been devised to minimise redundancy. In this reader we will use the *relational model* that offers a solution to redundancy. In the relational model, the columns that contain data that is repeated over and over again become separate files. Figure 1.2 shows what this would mean for our example.



► **Figure 1.2** Splitting up the sales file into a sales, article and customer file

In addition to the existing sales file, two new files arise: the CUSTOMERS file and the ARTICLES file. In these new files, every customer and each article appears only once. All we have to do now is refer to the right customer and the right article from the SALES file. This is usually done by means of a number or code as shown in figure 1.3.



CUSTOMERS			ARTICLES	
Customer no.	Customer name	Customer address	Article code	Description
265	P.Okko	Park way 19	SF23	Sports bike 230Z
791	A.Bangbuck	Fieldlane 2	KAB2	Tile AB2
...

SALES			
Customer no.	Article code	Date	Quantity
265	SF23	12-06-2002	2
265	KAB2	12-06-2002	1
791	KAB2	20-06-2002	4
791	KAB2	20-12-2002	1
265	SF23	01-12-2002	36
...

► **Figure 1.3** Referencing from the sales file to the customer and article file by means of a Customer No and an Article Code

1.2.3 Concepts when working with databases

Table Until now, we've talked about files and columns. However, when working with databases, concepts such as *table*, *attribute* and *record* are often used. Figure 1.4 shows what is meant by these terms.

name of <i>table</i>	ARTICLES			
	Article code	Description	Location	Weight
Rows	SF23	Sports bike 230Z	02F3	25
	KAB2	Tile AB2	02G1	57
	KLG1	Clock 1G	08A2	34
	BKG8	Bookcase Pine 8i	01B1	19

	Columns			

Labels in the diagram: 'name of table' points to the table name 'ARTICLES'. 'Rows' points to the data rows. 'Columns' points to the column headers. 'Attribute' points to a column header. 'record' points to a data row.

► **Figure 1.4** Some important concepts when working with databases

For the above concepts, synonyms are often used in literature. Below is an overview of these synonyms and the meaning given to them in this reader:

Term	Synonym
Table	file, entity, entity type
record	row, tuple, line
Attribute	column type, field, field type, attribute type

1.2.4 Keys

If, as discussed in paragraph 1.2.2, a reference is made to another table from a given table, there should be no misunderstanding as to which record is meant in that other table. If we, for example, refer from the table SALES to a customer with customer number 265, we want to make sure that this number belongs to customer P.OKKO and that there are no other customers with this number. Inside the table CUSTOMERS each customer number must therefore be unique. Such an attribute that makes a record unique, we call a *key* (also called *primary key* or *unique identifier*).

Key

A key can consist of one or more attributes. For the CUSTOMER table, only the customer number is enough to ensure that each record is unique. But which attribute could serve as a key to SALES the table? No attribute in this table makes a record unique. One solution could be to make the attributes CUSTOMER NO,



composite key

ARTICLE CODE and DATE form the key together. The *composite key* that is created is unique to each record (if we assume that a customer can never buy the same article more than once on the same date).

Instead of a composite key, we could have chosen to add a new SALES NUMBER attribute to the SALES table as a key. Which solution can we best choose now? For reasons of efficiency, in practice the key is usually sought to consist of as few attributes as possible. In that sense, the solution with the SALES NUMBER would therefore be best (because the key consists of only one attribute instead of three).

1.2.5 Referential integrity

referential integrity

Inconsistency

When we enter customer number 3006 in the table SALES from the example while there is no customer with that number in the table CUSTOMERS, a problem arises. In such cases, the references between the tables, and therefore the data, are no longer reliable (integrity). The reliability of the references in a database is called *referential integrity*. Most modern database software automatically monitors the referential integrity. In situations where data are not in accordance with each other, we speak of *inconsistent data*. In summary, it can therefore be stated that redundancy or lack of referential integrity can lead to inconsistent data.

Summary

Database

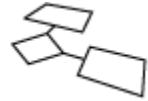
In an organisation people work together to achieve a certain goal. In order to achieve this goal, all kinds of data are used, which also need to be stored somewhere. Nowadays, this usually happens in one central storage: a **database**.

Database structure

When all the data is stored in a database on one large pile, **redundancy** quickly arises: the same data occurs several times, with all the dangers that entails. The **relational model** offers a solution to this problem: the large pile of data is split into a number of small piles: **tables**. These tables are connected by **keys** (numbers or codes that make a **record** unique). The references between the tables that emerge must of course be correct, this is what we call referential **integrity**.

Concepts

- process-oriented approach
- data-oriented approach
- file
- database (database)
- redundancy
- relational model
- table
- attribute
- record
- key
- composite key
- referential integrity
- inconsistency



2

The Entity Relationship Diagram

2.1 Introduction

In the previous chapter, we saw what problems can arise when building a database. That's why it's a good idea to create a design before we build a database. In this design we can then determine which tables will be needed, which attributes these tables should have and what the structure of the database should be.

relational
database
ERD

Now we could make this design by writing some text and thus describe the database, but this does not make it clearer in practice. We would rather draw the database, in order to get a clear picture. A picture often says more than a thousand words. To design a *relational database* (a database based on the relational model) multiple techniques have been developed over time. The technique we will work with in this reader is the *entity relationship diagram* (or abbreviated: *ERD*).

2.2 Symbols of an ERD

When drawing an ERD, three symbols can be used: the *entity*, the *relationship*, and the *attribute*. In addition, a number of characteristics of relationships can be indicated: the *cardinality*.

2.2.1 Entities

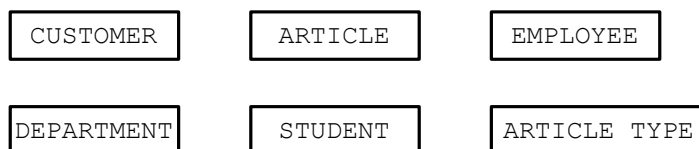
Entity

One of the symbols we can use when drawing an ERD is the *entity*. An entity is something we want to capture data about (now or in the future) and that is significant for the organisation we design the database for. In the ERD, we use a rectangle to designate an entity.



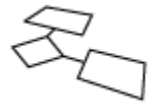
► Figure 2.1 The symbol for an entity

In the rectangle we put the name of the entity. It is common to write down this name in the singular. Figure 2.2 shows a number of examples of entities.



► Figure 2.2 A couple of example entities

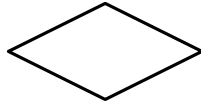
The entities in the ERD become tables in the database that we will eventually build. A database built using the above entities would therefore include a table CUSTOMER, a table ARTICLE, an EMPLOYEE table, a table DEPARTMENT, etc.



2.2.2 Relationships

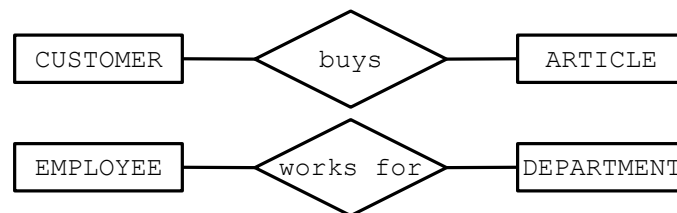
Relationship

In Chapter 1, we have seen that there are references between the tables in a relational database. For example, we found that a sales data table contains a customer number attribute that refers to a record of customer data in the customer table. These references can also be included in the ERD and are called *relationships*. A relationship is a meaningful connection between entities and is indicated by a diamond.



► **Figure 2.3** The symbol for a relationship

In the diamond we put the name of the relationship. Relationships connect entities, as Figure 2.4 shows below.



► **Figure 2.4** Two examples of a relationship

We can choose the direction in which we draw the relationship (from left to right, from top to bottom) ourselves. As the figure above shows, the name of the relationship is chosen in such a way that from left to right (or from top to bottom) a readable "sentence" is created. In the example we can read: *"a customer buys an article"* and *"an employee works in a department."*

Instead of the above relationship between CUSTOMER and ARTICLE, we could have also drawn the following:



This is exactly the same as the customer-article relationship in Figure 2.4. The fact *that* there is a relationship between the two entities is important. The direction we draw them in and the name we give them, we can further determine ourselves.

In the eventually built database, some relationships become tables, other relationships disappear when they are translated into a relational model. In Chapter 3, we will go into more detail about this.

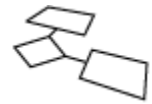
2.2.3 Attributes

Attribute

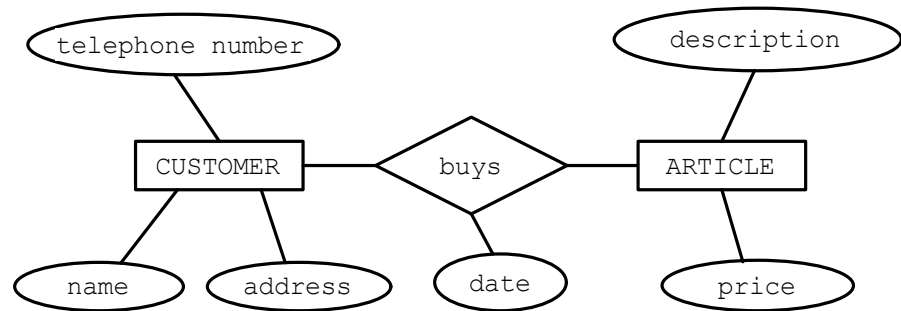
Attributes are the properties of an entity (or relationship) we want to keep track of. For example, if an organisation is interested in their customers' name, address, and phone number and wants to keep it in the database, we need an entity CUSTOMER with the NAME, ADDRESS, and PHONE NUMBER attributes attached to it. An attribute is indicated in the ERD by an ellipse.



► **Figure 2.5** The symbol for an attribute



In the ellipse we put the name of the attribute. Both entities and relationships can contain attributes, as the example below shows.



► Figure 2.6 Some examples of attributes

It is not required to have attributes for every relationship. Entities always have one or more attributes. If an entity has no attributes, we apparently don't want to keep track of anything about it, so it's usually pointless to include this "blank" entity in the design.

attribute list

By including attributes as ellipses in the ERD, the design quickly becomes cluttered because they take up a lot of space. This is especially true when the design is complicated and there are many attributes. Therefore, it is common not to display the attributes in the diagram, but to attach them to the ERD in a separate overview. Figure 2.7 shows an example of such an *attribute list*. In this reader, both notations will be used interchangeably.

Attribute list

CUSTOMER	: name, address, phone number
ARTICLE	: description, price
buys	: date

► Figure 2.7 Example of an attribute list

2.3 Cardinality

cardinality

In order to be able to build a good database, in practice we usually want to know more about a relationship than has been described so far. This additional information that we record of a relationship, we call *cardinality* and can be divided into *Functionality* and *Totality*.

2.3.1 Functionality

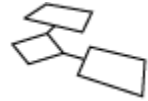
Check out the example below.



► Figure 2.8 A design for a database that tracks teachers and courses, as well as which teacher gives which course

The relationship between TEACHER and COURSE in this example can be interpreted in several ways:

- A particular teacher can give multiple courses
- A particular teacher cannot give more than one course
- A particular course can be given by multiple teachers
- A particular course can be given by no more than one teacher



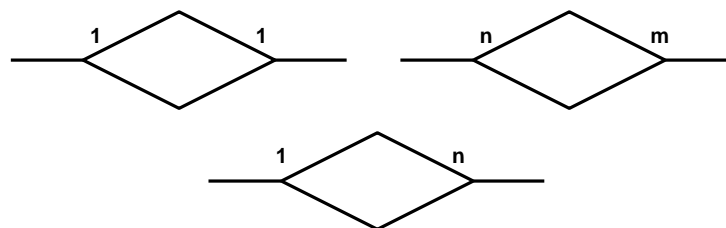
It is important for our design to capture the interpretations used in the organisation we design the database for. Let's say that we have designed the above ERD for a school where only specialized teachers work, who can only teach one course, but where a particular course can be given by several of those teachers. This situation can have a major impact on the structure of the database, and we can record this in an ERD as follows:



► **Figure 2.9** An ERD indicating its functionality

Functionality

The maximum number of occurrences for one entity in relation to another is called the *Functionality*. When a certain entity can occur at most once in relation to another entity, we write down a 1 near the relationship, on the side of that entity. If she can occur multiple times, we write down an n. In this way, the following possible combinations arise:



► **Figure 2.10** The different types of functionality

The functionality of a relationship is determined by how the organisation we create a design for works, and therefore needs to be reviewed for each organisation. At one school, for example, a course is given by multiple teachers (n), and on another school it is always given by a maximum of one (1). This results in two different designs.

To avoid confusion when determining functionality, it is best to use the following phrases:

- (1) "A certain ... can occur once in relation to a ... "
- (n) "A certain ... can occur several times in relation to a ... "

where on the dots (...) the names of the relevant entities are filled in. Below are two examples.



Read:

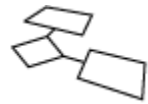
- A particular TEACHER can give multiple COURSES
- A particular COURSE can be given by up to one TEACHER



Read:

- A particular EMPLOYEE can work in multiple DEPARTMENTS
- Multiple EMPLOYEES can work in a particular DEPARTMENT

► **Figure 2.11** Two examples of how functionality can be read



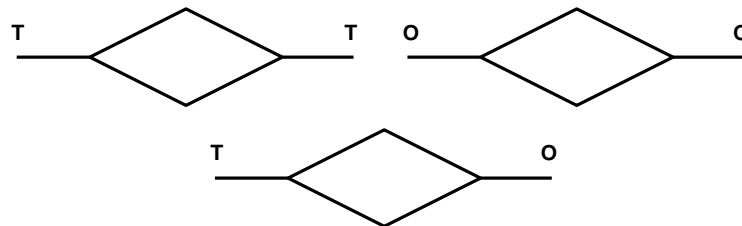
2.3.2 Totality

Another question we can ask ourselves about the relationship in Figure 2.8 is whether *every* teacher teaches a course, or if there are also teachers who do not teach courses (but for example only do research). We can give the following interpretations to the relationship from Figure 2.8:

- Every teacher teaches courses
- Not every teacher teaches courses
- Every course is taught by a teacher
- Not every course is taught by a teacher

Totality

The minimum number of occurrences for one entity in relation to another, we call the *Totality*. When a particular entity always appears at least once in relation to another entity, we write down a T (of "total") on the side of that entity. If it does not always occur, we write down a O (from "optional"). In this way, the following possible combinations arise:



► **Figure 2.12** The different types of totality

We can also use phrases to help determine the totality:

- (T) "*For every ... this occurs in relation to ...*"
 (O) "*Not every ... occurs in relation to ...*"

where on the dots (...) the names of the relevant entities are filled in. Below are two examples.



Read:

- Every EMPLOYEE works for a DEPARTMENT
- In every DEPARTMENT, EMPLOYEES work

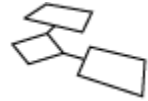


Read:

- Every CUSTOMER has bought an ARTICLE
- Not every ARTICLE is sold to a CUSTOMER

► **Figure 2.13** Two examples of how totality can be read

The same applies to totality as to functionality: it is determined by the situation in the organisation we are creating our design for. In one organisation, every customer will have bought an article, for example because customers will not be registered until they buy something (T). Another organisation keeps a large customer base with customers who have not (yet) bought anything (O).



2.4 Historical and up-to-date databases

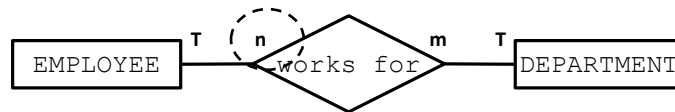
historical database

current database

There are databases that track both today's data and a historical overview. Such databases are called *historical databases*. This type of database is by far the most common, as organisations are often interested in past data. Think, for example, of the sales results of the past year in a trading company. In order to be able to extract such data from the database, it must be stored over time. Opposite is the *current database* which only records the data of this moment.

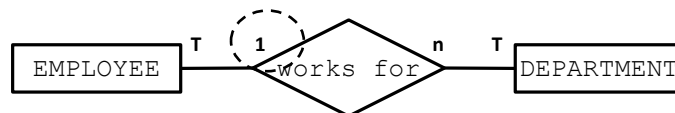
In this reader we will usually rely on a historical database, because in practice this is the most common situation. The fact whether the database should be up-to-date or historical can have an effect on the cardinality of the ERD that we create for it. The example below illustrates this.

At the company *Crocon* e.g. employees always work in one department at a time. There are always several employees in a department.



Functionality in a historical database, read:

- A particular EMPLOYEE may have worked in multiple DEPARTMENTS *over time*



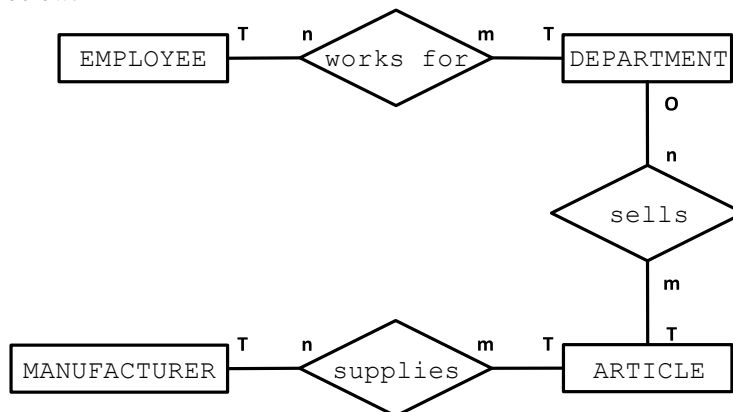
Functionality with an up-to-date database, read:

- A particular EMPLOYEE *currently* works in one DEPARTMENT

► **Figure 2.14** The difference in functionality between a historical and up-to-date database

2.5 Example of an ERD

To demonstrate the theory covered so far, an example of an ERD is shown below.



Attribute list

EMPLOYEE	: name, first name, salary
DEPARTMENT	: name, telephone number
ARTICLE	: description, sales price
MANUFACTURER	: name, address, city, zip code, phone number
works for	: start date
supplies	: date, purchase price

► **Figure 2.15** Example of an ERD



Figure 2.15 is the design of a (historical) database for an organisation that works as follows (check this yourself):

Employees can work in multiple departments over time and always work in a particular department. There are always several employees working in a department. A number of departments sell articles. All articles can be delivered by multiple manufacturers and sold by multiple departments. A manufacturer can supply multiple types of articles and is only included in the manufacturer's file when articles are first ordered from him.

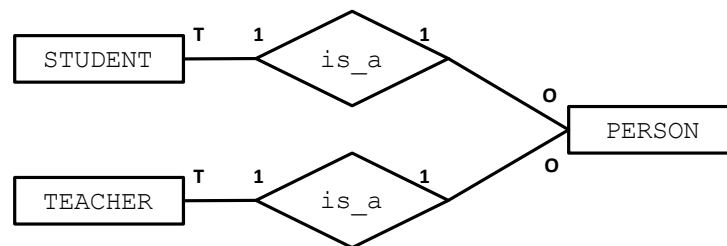
For employees they want to keep track of the first name, name and salary and from what date they have worked in a particular department. Over time, it has proved useful to keep a name and (internal) telephone number per department. For articles the description and sales price is always recorded, for manufacturers the name of the company, the address (including place name and postal code) and the telephone number. Finally, the date on which a article was purchased from which supplier and at what purchase price this has been done is always recorded.

2.6 Special relationships

2.6.1 Generalisation

It sometimes happens that two (or more) entities in an ERD have a lot of attributes in common. For example, think of the entities TEACHER and STUDENT. Both entities may have the attributes name, first name, address, city, zip code, telephone number, email and date of birth (furthermore, for a TEACHER the salary could be recorded and for a STUDENT the student number). With an ERD we try to design the most efficient database possible and having two entities with so many similar attributes is not very efficient. The solution to this is called *Generalization* and looks as shown in Figure 2.16.

Generalization



Attribute list

PERSON	: name, first name, address, city, zip code, telephone number, e-mail, Date of birth
STUDENT	: student number
TEACHER	: salary

► **Figure 2.16** An example of generalisation

All attributes that appear in both STUDENT and TEACHER are merged into a new entity PERSON, so that these attributes appear only once in the ERD. In STUDENT and TEACHER, only the attributes that apply specifically to that entity remain (and therefore cannot be merged in PERSON). In the example, this applies to SALARY and STUDENT NUMBER.

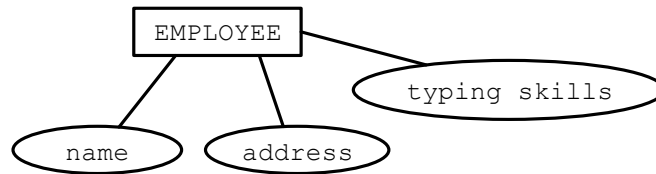
The functionality and totality are *always* the same with generalization, namely a 1-on-1 relationship, with a O on the side of generalisation and a T on the other side of the relationship. The name of the relationship, *IS_A*, is a reserved name and should only be used in a special relationship (as used here in the case of generalisation). An IS_A relationship never has attributes.

is_a relationship



2.6.2 Specialisation

Check out the example below.



► **Figure 2.17** Entity employee with name, address, and typing skills attributes

The database that will be build based on this design will contain a table EMPLOYEE with at least the attributes NAME, ADDRESS, and TYPING SKILLS. Let's say that the organisation the design was created for has 1,500 employees, 25 of whom are a typist. The EMPLOYEE table will then include 1500 records, with no less than 1475 records leaving the TYPING SKILLS attribute blank.

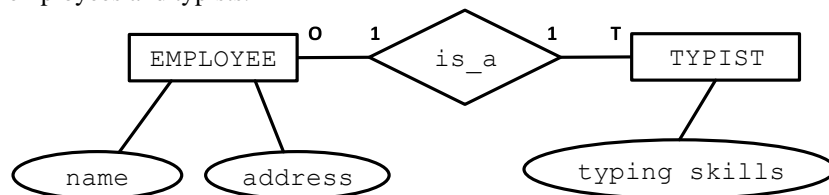
EMPLOYEE		
Name	Address	Typing skills
Arie	Arielaan 6	
Piet	St. Petersburg 89b	
Jan	Jweg 17	
Klaas	Klastraat 2	
Miep	Mistraat 90	Typediploma B
Jannie	Jannesquare 8	
...

← Of the 1500 records, 1475 will contain blank fields

► **Figure 2.18** Examples of some records in the employee table

Specialization

Such a table naturally takes up a lot of (disk) space and is also not very efficient to use. To solve this problem, we can use *Specialization* in our ERD. Figure 2.19 below shows how we would apply specialization in our example of the employees and typists.



► **Figure 2.19** An example of specialisation

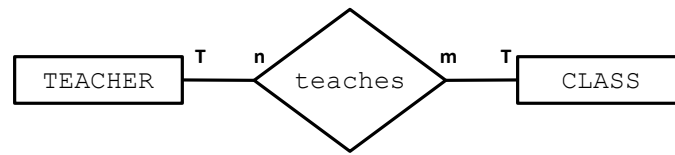
We're adding a new ENTITY TYPIST, which is a specialization of the entity EMPLOYEE. All attributes that apply to typists only, we then include in TYPIST. In this way, the employee table EMPLOYEE from Figure 2.18 is thus split into two tables: one with the data of all employees, and one with the typist data of a number of special employees: the typists. The EMPLOYEE table will then contain 1500 records, and the table TYPIST 25 records. There will be no more empty fields in both tables.

As with generalization, specialization uses an IS_A relationship. Again, that relationship *always* has the same functionality (1-on-1) and totality (T on the side of the specialization, O on the other side of the relationship).



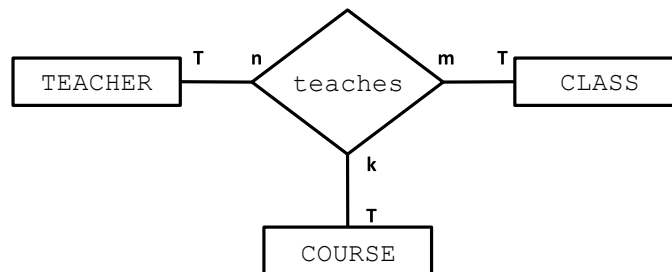
2.6.3 Multidimensional Relationships

Until now, we have assumed that a relationship always connects two entities. However, sometimes it is useful to have a relationship connect three or even more entities. Check out the following example.



► **Figure 2.20** Each teacher teaches one or more classes, each class is taught by one or more teachers

The above ERD results in a database in which we can record data about teachers and classes, and thanks to the relationship, also track which teacher is giving lectures to which class. Let's say that we also want to keep track of which teacher gives which class which *course*. The relationship will then have to connect not only the entities **TEACHER** and **CLASS**, but also the (new) entity **COURSE**. Figure 2.21 shows how this is done.



► **Figure 2.21** An example of a three-dimensional relationship

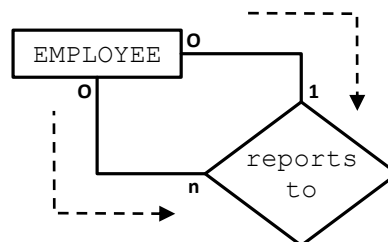
multidimensional relationship

The relationship in Figure 2.21 is called a *multidimensional relationship*, because it connects more than two entities. In this case, there is actually a *three-dimensional relationship*. Similarly, we can also connect four, five or more entities. When choosing the name for such a relationship, we try to reflect the coherence between the entities as clearly as possible (it does not necessarily have to be a correct sentence).

In chapter 4, we elaborate further on the best way to use multidimensional relationships in practice, and what impact this has on the database to be built.

2.6.4 Relationships that connect an entity to itself

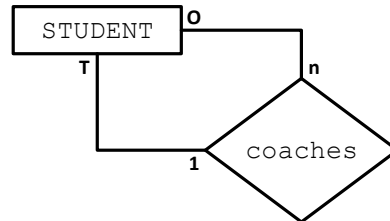
So far, relationships have been able to connect two or more entities. However, sometimes it is useful to connect a relationship twice to the same entity. Let's say we want to indicate that an employee reports to another employee (his manager), and that this other employee receives reports from multiple employees. This can be modelled as follows:



► **Figure 2.22** An example of a relationship that is connected to the same entity twice, and how such a relationship should be read



As the example shows, we only need one entity because managers are also employees and (in this example) have the same attributes. Like any relationship, we can also read the relationship from Figure 2.22 in two directions. It is common to draw these kinds of relationships at the lower right of the entity, so that there can be no misunderstanding about the direction in which we should read. Finally, here's another example of a relationship that's been linked to the same entity twice.






► **Figure 2.23** Some students coach several fellow students. All students are coached by one fellow student.

Summary

ERD

Before we build a database, it's wise to create a design first. The **entity relationship diagram** (ERD) is a technique for designing databases. We have three symbols at our disposal:

- **entity**  something the organisation wants to capture something about
- **relationship**  a meaningful relationship between two entities
- **attribute**  properties of an entity or relationship that are recorded

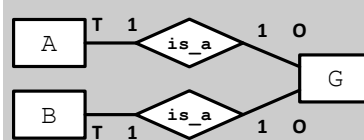
Cardinality

For a relationship the **cardinality** is also captured in an ERD. The cardinality consists of **functionality** and **totality**. The functionality indicates whether an entity can occur multiple times in relation to another entity (**n**), or up to once (**1**). Totality can indicate whether an entity always occurs in relation to another entity (**T**), or not always (**O**).

Special relationships

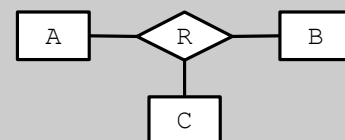
In order to design the most efficient database possible, we have a number of special relationships at our disposal:

■ generalization



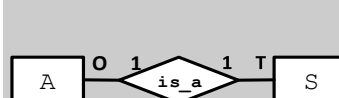
The overlapping attributes of A and B are accommodated with G

■ multidimensional relationship



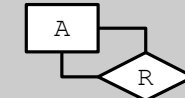
Relationship R connects entities A, B and C

■ specialization

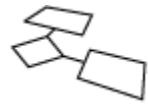


Attributes that cause many empty fields in A are accommodated in S

■ twice with the same entity



A is connected to itself via relationship R



3

From ERD to relational model

3.1 Conditions

To be able to convert an ERD into a relationship model there are two primary conditions.

Condition 1: Give each entity type and relationship type a unique name.

In a database it is not allowed for two tables to have the same name. Therefore, in an ERD, we must ensure that every entity type and relationship type has a unique name. Only the special relationship with the name is_a may occur more than once (it can never be a name). Often relationship names as 'has' and 'contains' occur multiple times. These relationships must be given a more meaningful name; if that is really not possible, this can be solved by numbering them (has 1, has 2, etc.).

Condition 2: All entity types have a primary key.

A primary key is:

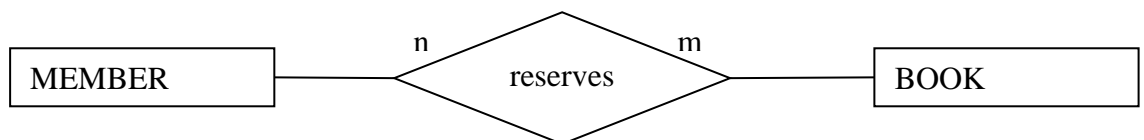
- uniquely identifying;
- minimal (with as few attributes as possible);
- not empty (a key attribute must have a value for each entity).

For further information, see section 1.2.4 of the ERD Reader.

3.2 Converting relationships

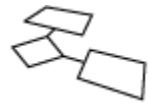
Converting n-to-m relationships

Given the following ERD with attribute list:



Attribute list

MEMBER: member number, name, address, city
 BOOK: ISBN, title
 reserves: booking date



The conversion of n-to-m-relationships is very straightforward. The relationship type in an n-to-m relationship is converted into a table, which, in addition to its own attributes, contains the primary keys as reference keys to the participating entity types. These two primary keys together form the composite key for the new table. This table can be seen as an entity type and also the naming is adjusted accordingly (instead of a verb in lowercase a noun in capital letters).

The relational model then looks like this:

MEMBER: member number, name, address, city

BOOK: ISBN, title

RESERVATION: member number, ISBN, reservation date

foreign key 'member number' refers to member number in LID

foreign key 'ISBN' refers to ISBN in BOOK

Note, a primary key is underlined, a reference key is typed *italics*, or underlined with a circuit line (when you write by hand).

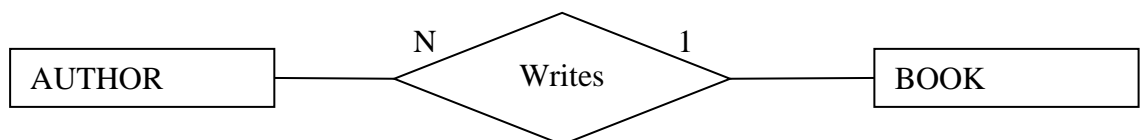
This leads to the following general conversion rule:

Conversion rule 1

An n-to-m relationship from an ERD is represented in a relational model by means of three tables: two for the entity types and one for the relationship type. In the table created by the relationship, reference keys are included to both entity types. These reference keys together form the primary key to the new table.

Conversion n-on-1/1-on-n-relationships

Given the following ERD with attribute list:



Attribute list

AUTHOR: author code, name, city

BOOK: ISBN, title

writes: start date, end date

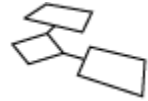
The relationship "writes" between AUTHOR and BOOK can be created by including a reference key author code in BOOK. The attribute types of the relationship are also included in BOOK

The relational model then looks as follows:

AUTHOR (author code, name, city)

BOOK (ISBN, title, *author*, start date, end date)

foreign key 'author' refers to author code in AUTHOR



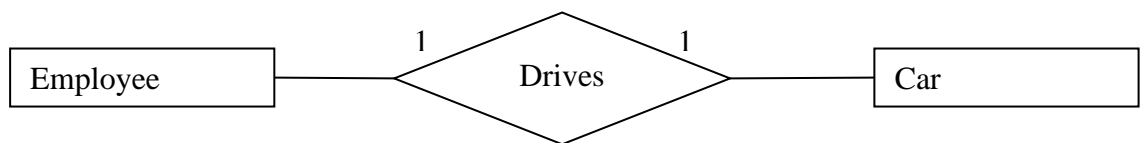
It is clear that with this method of modelling each book always includes only one author, but one author can have several books. This leads to the second general conversion rule:

Conversion rule 2

In the case of an n-to-1 or a 1-to-n relationship from an ERD, the relationship type is not converted into a separate table. The 1-sided entity type is converted into a table that, in addition to the attribute types of the relationship, also contains a reference key to the n-side entity type. The other entity type remains unchanged.

Conversion 1-to-1 relationships

Given the following ERD with attribute list:



Attribute list

EMPLOYEE: staff number, name, address, city

CAR: registration plate, make, type, colour

driving: start date

Unlike with an n-to-m relationship or a 1-to-n relationship, there cannot be made an unambiguous choice where to include which reference key. In principle there are two possibilities, analogous to conversion rule 2.

Relational Model A

EMPLOYEE (staff number, name, address, city, *registration plate*, start date)

foreign key 'registration plate' refers to registration plate in CAR

CAR (registration plate, brand, type, colour)

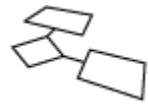
Relational model B

EMPLOYEE (staff number, name, address, city)

CAR (registration plate, mark, type, colour, *staff number*, start date)

foreign key 'staff number' refers to staff number in EMPLOYEE

However, we are not finished yet. So far, we've actually worked the same way as we did with the 1-to-n-relationships. And indeed, the two above models allow 1-to-n-relationships. As an example, we take the CAR table of the second modelling:



Registration plate	Brand	Type	Color	Staff no.	Start
AA-BB-11	Renault	Megane	Red	123456	1-6-2003
AA-CC-22	Renault	Megane	Red	678912	1-8-2003
BD-CD-32	Opel	Astra	Green	454166	3-10-2003
AD-RE-45	Peugeot	307	Grey	456146	5-12-2003
CB-RE-65	Renault	Laguna	Blue	123456	2-9-2003

Every registration plate only appears once. That makes sense, because registration plate is the key attribute of CAR and it belongs to the definition of a key that each value is unique. In other words, the key to a table is based on a uniqueness restriction. However, the staff number 123456 occurs twice, which of course should not happen in a 1-to-1 relationship. But we have no means to forbid that so far: staff number is a reference key in CAR and the requirement of uniqueness does not apply automatically. We must therefore explicitly add this requirement to a 1-to-1 relationship in a relational model in the hope that the chosen or developed database package will be able to monitor this requirement in the implemented database. We choose the following form for that:

EMPLOYEE (staff number, name, address, city)

CAR (registration plate, mark, type, colour, *staff number*, start date)

foreign key 'staff number' refers to staff number in EMPLOYEE

staff number is unique in CAR

This leads to the third general conversion rule:

Conversion rule 3

In the case of a 1-to-1 relationship from an ERD, the relationship type is not converted into a separate table. One of the entity types is converted into a table that contains all attribute types of the relationship as well as a reference key to the other entity type. This reference key in the table is subject to a uniqueness restriction. The other entity type remains unchanged.

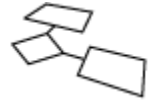
3.3 Converting totality

Converting n-to-1/1-to-n-relationships and n-to-m relationships

If an entity participates in a relationship partially, we do not need to take additional measures when transposing the relational model: the normal relationship between two tables with a common attribute type is partial.

However, in order to enforce totality, we need to add additional restrictions to the relational model. To this end, we have to choose between two types that we will explain below.

Given the ERD (as above):



In this relationship, both entity types participate in total. The conversion of this piece of ERD into a relational scheme is as follows:

AUTHOR (author code, name, city)

BOOK (ISBN, title, *author*, start date, end date)

foreign key 'author' refers to author code in AUTHOR

How can we ensure that the BOOK entity type participates in the relationship totally (i.e. that each book has an author)? The answer is to require that the author code attribute type always has a value in the BOOK table. After all, when the author code field remains blank in a record from that file, it means that a BOOK is not written by an AUTHOR. So the relationship is partial. This situation is illustrated by the following sample table:

BOOK

ISBN	Title	Author code	etc
1231456545	The Assault	AO231	
4545516165	I'm always right	AB099	
4546456545	Among professors	AB099	
5456625656	The Little Prince		
4554656652	Eline Vere		
5465465552	The Evenings	VB212	

The situation above is not the situation we wish for. We want that every BOOK is written by an AUTHOR.

We formulate the restriction on the employee reference key as follows:

AUTHOR (author code, name, city)

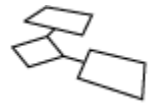
BOOK (ISBN, title, *author*, start date, end date)

foreign key 'author' refers to author code in AUTHOR, NULL not allowed

This leads to the fourth general conversion rule:

Conversion rule 4

When a table with a reference key participates in a total relationship, it is mandatory to fill the reference key in that table.



How can we ensure that the AUTHOR entity type fully participates in the relationship (i.e. that there is at least one book from each author)? We do this by requiring that every key value of AUTHOR occurs (at least once) as a reference key in BOOK. Like in the tables below:

AUTHOR

Author code	Name	City
AO231	H.Mulisch	Amsterdam
AB099	W.F.Hermans	Groningen
VB212	G.Reve	Machelen

And

BOOK

ISBN	Title	Author code	Etc.
1231456545	The Assault	AO231	
4545516165	I'm always right	AB099	
4546456545	Among professors	AB099	
5465465552	The Evenings	VB212	

In the example above, all author codes appear in the AUTHOR table in BOOK, i.e. from each author there is at least one book, i.e. the author entity type fully participates in the relationship writes. We describe this as follows in the relational model:

AUTHOR (author code, name, city)

BOOK (ISBN, title, *author*, start date, end date)

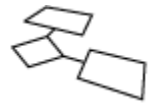
Foreign key 'author' refers to author code in AUTHOR, NULL not allowed

Foreign key author in BOOK contains all individual author code values in AUTHOR

This leads to the fifth general conversion rule:

Conversion rule 5

When a table without a reference key participates in a relationship totally, all key values must appear at least once as the value of the reference key in the other table.



Converting 1-to-1 relationships

Given the following ERD with attribute list:



Attribute list

EMPLOYEE: staff number, name, address, city

CAR: registration plate, make, type, colour

driving: start date

This leads, not taking into account totality, to one of the following relational models.

Relational Model A

EMPLOYEE (staff number, name, address, city, *registration plate*, start date)
foreign key 'registration plate' refers to registration plate in CAR

CAR (registration plate, brand, type, colour)
registration plate is unique in EMPLOYEE

Relational model B

EMPLOYEE (staff number, name, address, city)

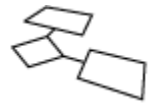
CAR (registration plate, mark, type, colour, *staff number*, start date)
staff number is unique in CAR
foreign key 'staff number' refers to staff number in EMPLOYEE

Where there is no preference for model A or model B.

However, working out model A you get the following tables.

EMPLOYEE

Staff no.	Name	Address	City	Registration plate	Start date
123456	Alders	Village Street 1	Our Village	AA-BB-11	1-6-2003
678912	Barend	Wood Street 3	Haarlem	AA-CC-22	1-8-2003
554485	Cornelisse	Grote Markt	Groningen		
152562	Driessen	A.Kerkhof	Groningen		
454166	Evers	Blaak	Rotterdam	BD-CD-32	3-10-2003
522458	French	Hofsquare	Rotterdam		
456146	Great	Neude	Utrecht	AD-RE-45	5-12-2003



CAR

Registration plate	Brand	Type	Colour
AA-BB-11	Renault	Megane	Red
AA-CC-22	Renault	Megane	Red
BD-CD-32	Opel	Astra	Green
AD-RE-45	Peugeot	307	Grey

In this case, empty fields arise in the EMPLOYEE table for employees without a car (EMPLOYEE participates optionally in the relationship drives).

Working out Model B you get the following tables.

EMPLOYEE

Staff no.	Name	Address	City
123456	Alders	Village Street 1	Our Village
678912	Barend	Wood Street 3	Haarlem
554485	Cornelisse	Grote Markt	Groningen
152562	Driessen	A.Kerkhof	Groningen
454166	Evers	Blaak	Rotterdam
522458	French	Hofsquare	Rotterdam
456146	Great	Neude	Utrecht

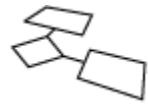
CAR

Registration plate	Brand	Type	Colour	Staff no.	Start
AA-BB-11	Renault	Megane	Red	123456	1-6-2003
AA-CC-22	Renault	Megane	Red	678912	1-8-2003
BD-CD-32	Opel	Astra	Green	454166	3-10-2003
AD-RE-45	Peugeot	307	Grey	456146	5-12-2003

In this case, there are no empty fields!

The preference is always that there are no empty fields. That would only cost more storage space and also reduces the possibility for enforcing required fields.

In the case of a 1-to-1 relationship, the totality can therefore determine the preferred option to choose.



4

Designing an ERD

4.1 Introduction

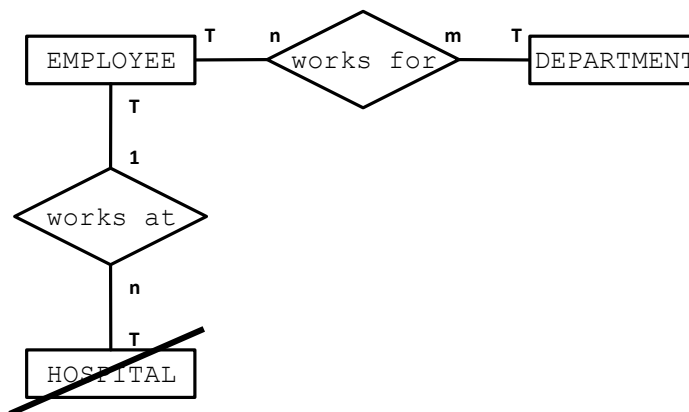
Chapter 2 explains how we can make an ERD and what rules we should abide by. Chapter 3 then described how we can get from an ERD to a relational model, on the basis of which the database can be built. During the design process, we often face important choices that have a big effect on the database that we will eventually build from it. Some of these choices and some pitfalls in designing an ERD will be addressed in this chapter.

4.2 Relevance

When designing an ERD, we must always keep in mind what the purpose of the design is: ultimately, we want to build an efficient database that meets the requirements of the organisation. It needs to be possible to find the data that make the organisation function properly in the database, nothing more and nothing less. Therefore, we must ensure that we only include relevant entities, relationships, and attributes in our ERD.

4.2.1 Relevance of entities

When we determine which entities are needed, we risk including too few or too many. The example below is a design for the workforce of a hospital.



► **Figure 4.1** An ERD for the workforce of a hospital with an irrelevant entity

The HOSPITAL entity is irrelevant in this ERD. The easiest way to make this clear is to imagine what this table will look like in the database. What kind of records will be in it? When we do this, it quickly becomes clear that there will always be only one record in this table, filled with the data from the hospital we make the design for.

irrelevant entities Rule: *Entities that become a table that always contain one record, are irrelevant and are not included in an ERD.*



The HOSPITAL entity from Figure 4.1 would look like table like this:

HOSPITAL		
Name	Address	Place
Tin village hospital	Village street 8	Tin village

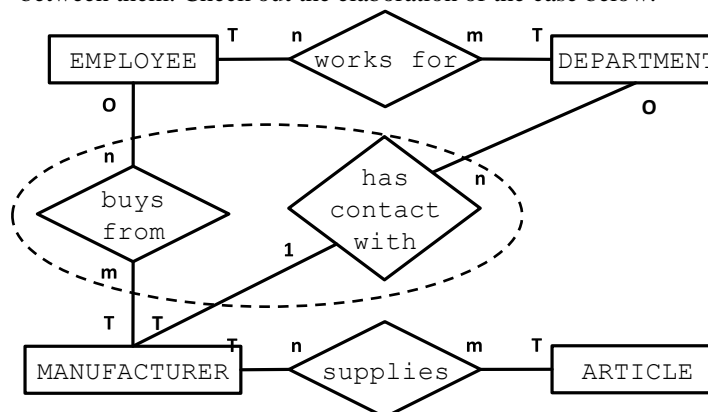
► **Figure 4.2** A (useless) table that always contains one record, containing some data about your own organisation

4.2.2 Relevance of relationships

In the previous chapters, we have seen that establishing a relationship between entities has important implications for the structure of the database. Among other things, it ensures that references between the tables arise, with the help of keys. Designing an ERD is therefore not a matter of first determining the entities, and then drawing some relationships between them. It's important that we only establish relationships in those places where we really want to keep a relationship. Read the case below, which serves as the basis for the design of a database for a trading company.

Employees can work in multiple departments over time and always work in a particular department. There are always several employees working in a department. A number of departments sell articles. All articles can be delivered by multiple manufacturers and sold by multiple departments. A manufacturer can supply multiple types of articles and is only included in the manufacturer's file when articles are first ordered from them. One would like to keep track of which manufacturer delivers which articles.

The necessary entities have already been underlined: EMPLOYEE, DEPARTMENT, ARTICLE and MANUFACTURER. However, determining the entities is not the most difficult in this case, it is more difficult to establish the right relationships between them. Check out the elaboration of the case below.

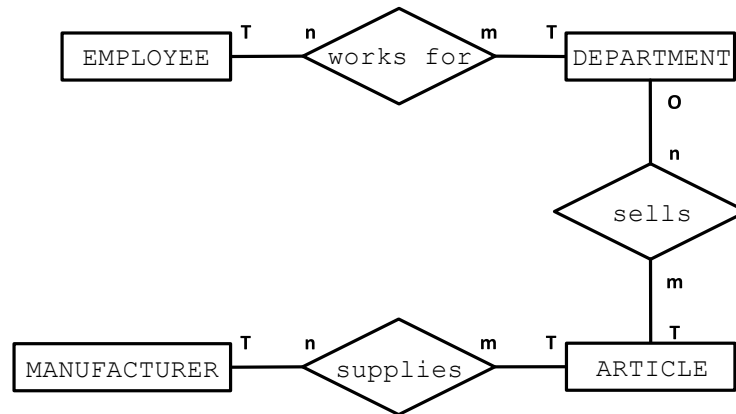


► **Figure 4.3** An (erroneous) elaboration of the case

Figure 4.3 incorporates the right entities, but does not match the situation in the organisation. The case description shows that one is only interested in the relations between the entities:

- EMPLOYEE and DEPARTMENT
- DEPARTMENT and ARTICLE
- ARTICLE and MANUFACTURER

The relationships BUYS FROM and HAS CONTACT WITH in figure 4.3 may look logical, but they are irrelevant for this case. There are undoubtedly organisations for which the given elaboration is fine, but for the organisation from the case it will result in a wrong database. One is not interested at all in the contacts that each department has with the manufacturers and therefore does not want to keep data on this in the database. For example, one is interested in which articles are sold by which department, but that data will not be found in the database with the above design. The correct design for the case:



► **Figure 4.4** The correct elaboration of the case

In this ERD, the aforementioned relationships can all be found. The database will therefore be able to find:

- in which departments an employee has worked (thanks to the relationship WORKS FOR)
- which articles each department sells (thanks to the relationship SELLS)
- which manufacturer supplies which articles (thanks to the relationship SUPPLIES)

4.3 Again: attributes to relationships

dates
numerical data

For attributes from entities, we can usually easily imagine something: for an entity CUSTOMER for example, we often want to keep a name, address, zip code etc. But when do relationships actually have attributes? And what kind of attributes are they? To begin with the last question: attributes of relationships almost always concern *dates* or *numerical data* (data that can be counted, such as a price or an amount). They usually tell you something about the when or about the quantity of the relationship. An ERD for a car dealer is given below.



Attribute list

CUSTOMER	: customer no, name, address
CAR	: chassis no, brand, type, suggested retail price
buys	: sales date, sales price

► **Figure 4.5** An ERD for a car dealer

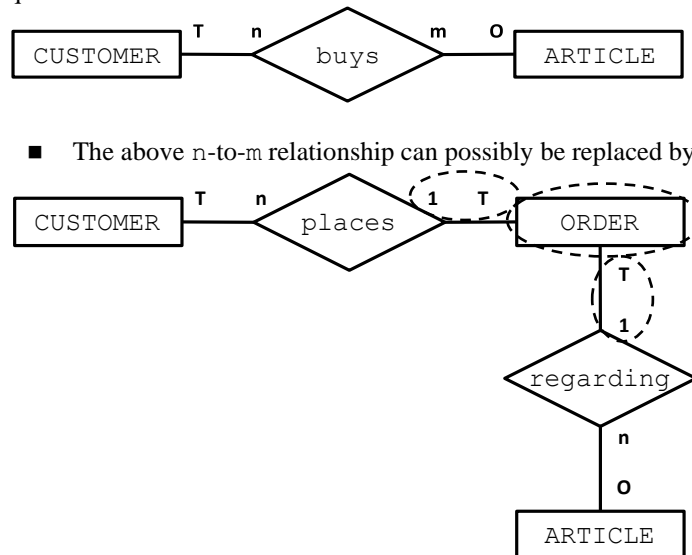
The data we want to keep track of concerning the customers and cars speaks for itself. But the price the car is sold for and the date this happens are neither part of the customer nor the car. That's why these are typical attributes that we include in a relationship: they tell something about the link between the entities, and not directly something about the entities themselves.

4.4 Once again: n-to-m relationships

Relationships with the functionality n-to-m eventually become independent tables in the database. We can also choose to include such a (junction) table in our ERD right away. Figure 4.6 shows that the resulting entity is surrounded by T1-relationships, creating the same database when restructuring as if we had used an n-to-m relationship. Both variants have their advantage: the n-to-m relationship looks clearer, for the second variant a separate entity ORDER can be included, which perhaps better fits in with the conventions of the organisation in



question.

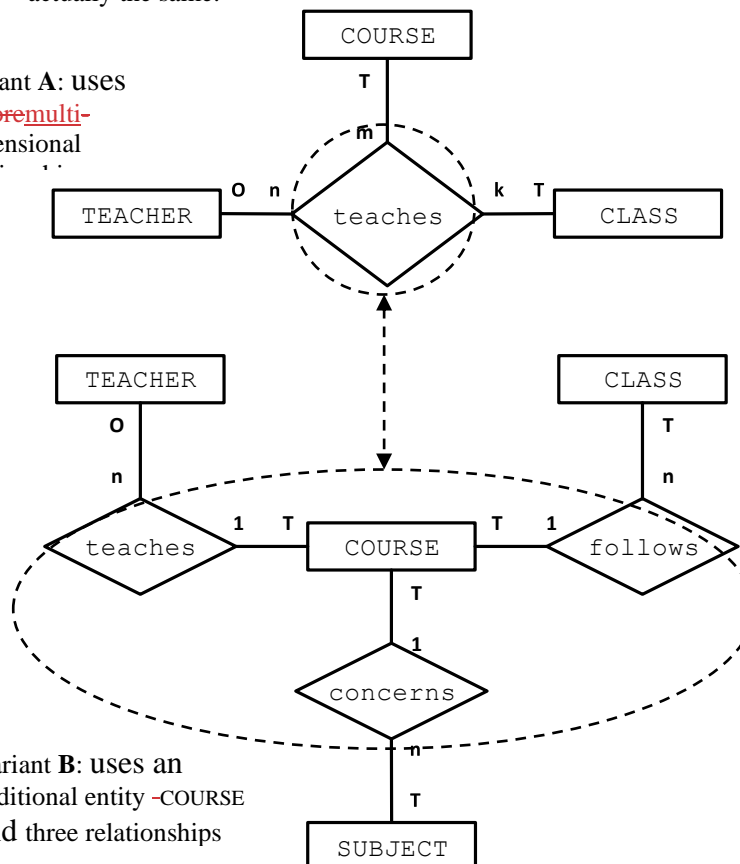


► **Figure 4.6** The n-to-m relationship buys is replaced by entity order

4.5 Again: multidimensional relationships

In practice, it often proves difficult to determine when a multidimensional relationship can be used or not. To understand exactly what a multidimensional relationship means, it's worth remembering that we could also use an additional entity instead of such a relationship, like we could for an n-to-m relationship. Below are two ER diagrams that look very different at first glance but are actually the same.

- Variant A: uses a ~~more~~multi-dimensional relationship



- Variant B: uses an additional entity -COURSE and three relationships

► **Figure 4.7** Two different ER diagrams that will yield the same database



Both variant A and variant B would be a good solution, although variant A looks a lot clearer. And that's also why we would choose a multidimensional relationship instead of an additional entity: it's clearer. For the operation of the database it doesn't matter what we choose, because the entity COURSE in variant B is surrounded by T1 relationships. This means that when restructuring variant B the attributes of the relationships GIVES, FOLLOWS and CONCERNS all are moved under the entity COURSE. This creates exactly the same tables as if we had chosen variant A.

It is also important to realize that a multidimensional relationship, like an ordinary two-dimensional relationship, is intended to connect entities. We want to track not only data about the entities, but also what record from one entity is linked to what record from the other. The following attribute list (and table) is part of variant A from Figure 4.7:

Attribute list

TEACHER : teacher no, name
 CLASS : class code, comments
 COURSE : course code, description
 teaches : course date

- After basic rules and restructuring, the relationship TEACHES will become a table in the database, as shown below (check for yourself):

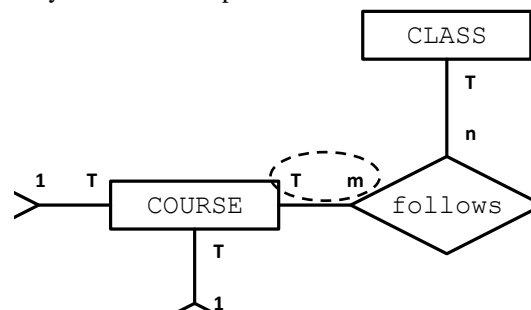
TEACHES

Teacher no	Class code	Course code	Course date
D20	1i3	IK13	23-04-2002
D21	1i2	BA12	01-12-2002
...

► **Figure 4.8** How variant A's multidimensional relationship eventually becomes a table that **links three other tables** together

The table TEACHES will contain a composite key consisting of teacher NO, CLASS CODE and COURSE CODE. Because the table contains these three keys, it links the tables TEACHER, CLASS and COURSE together. When it needs to become a historical database, the teaching DATE attribute will also have to be included in the key. Otherwise, it would not be possible for a particular teacher to give the same subject to the same class several times over the years (which is of course very common within a school).

However, a multidimensional relationship cannot always be used easily. When variant B from the ERD of Figure 4.7 is also used to register lectures (given to multiple classes at once), a problem arises: COURSE is no longer surrounded by only T1-relationships.



► **Figure 4.9** A particular lesson is given to one or more classes at a time

If we were to use a multidimensional relationship (variant A in Figure 4.7), problems would arise in the database. The attributes and keys of all relationships connected to the entity COURSE (including FOLLOWS), will then be placed in CLASS. For the T1-relationships this goes well, the Tm-relationship creates

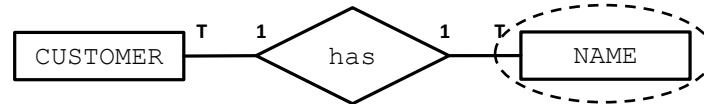


redundancy (see also paragraph 3.3.2).

4.6 Pitfalls

4.6.1 Entities that are actually attributes

When determining the required entities, it may happen that we mistake an attribute for an entity. The ERD below is an example of such a situation.



► **Figure 4.10** An attribute modelled as an entity

Such a construction is not entirely wrong, because when restructuring, NAME is moved to CUSTOMER, but it does not make the ERD clearer. Furthermore, it's confusing because what attributes does the NAME entity have in this example?

4.6.2 Attributes that are actually entities

Conversely, it sometimes occurs that we accidentally include an attribute that should actually be an entity. If we do nothing about such a situation, redundancy will occur in the database. Check out the example below.



Attribute list

ARTICLE: article code, description,
price, article type

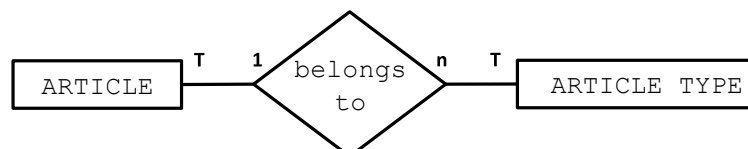
- After basic rules and restructuring, this ERD results in the following table:

ARTICLE

Article code	Description	Price	Article type
2508	Spinach	€ 2,25	Vegetable
2509	Orange	€ 0,75	fruit
3651	Banana	€ 2	fruit
2921	Apple	€ 1,50	fruit
...

► **Figure 4.11** The article type attribute causes redundancy and should have been an entity

The ARTICLE TYPE attribute in the example above causes redundancy. This means we've made a design flaw: ARTICLE TYPE is not an attribute but an entity. The ERD of Figure 4.10 should have looked like this:



Attribute list

ARTICLE : article code, description, price

ARTICLE TYPE: type code, type name

Attribute list after restructuring

ARTICLE : article code, description, price,
type code

ARTICLE TYPE: type code, type name

► **Figure 4.12** An improvement in the ERD of Figure 4.10



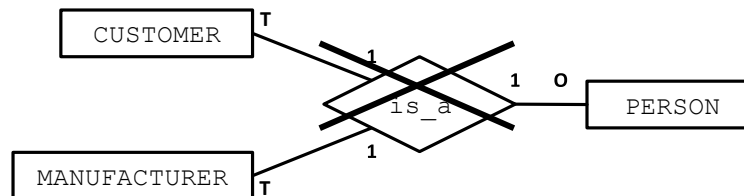
4.6.3 The relationship is_a

rules is_a

When working with the special relationship with the name IS_A, a number of rules apply. An IS_A Relationship:

- must have the name IS_A
- should only be used in generalisation and specialisation
- has no attributes
- is always two-dimensional
- has always the functionality 1-to-1
- always has totality T-to-O
 - for specialization: the T on the side of specialisation
 - for generalization: the O on the side of generalization

These rules always apply and the construction in figure 4.13 is therefore wrong. This is actually logical because this construction suggests that a CUSTOMER is not only a PERSON but also a MANUFACTURER, which of course can never be the intention of this design.



► **Figure 4.13** An is_a relationship is always two-dimensional, so this ERD is wrong

4.6.4 Attribute lists

Each ERD must list the attributes per entity. When we choose not to draw the attributes of the ERD as ellipses because this becomes confusing, we create an attribute list. For each entity and relationship, we list the data we want to keep track of (the attributes) in the attribute list. At this stage, we don't do anything with the attributes. In the attribute list of the ERD:

rules attribute list

- keys may not be declared
- basic rules may not be applied
- should not be restructured

The above steps are implemented at a later stage during the basic rules and restructuring. It is important to make a clear distinction between the attribute list of the ERD and the attribute lists that arise from basic rules and restructuring, otherwise the risk for errors is very high.

4.6.5 Process data

Process data

Process data is data that results from a calculation. Think, for example, of "the total price of an order" or "the number of customers." Normally, this data is not stored in a database because it carries risks: every time data is changed somewhere, the calculations have to be re-executed to update the process data. When we (as a programmer) forget to carry out such a calculation somewhere, the relevant process data is inconsistent with all the consequences for the reliability of the database that entails. Therefore, no process data may be included in the attribute list.



Summary

Relevance

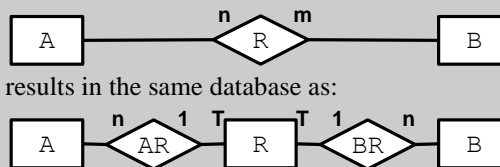
When designing an ERD, we need to make sure that we only include relevant entities and relationships. A tool to discover **irrelevant entities** is to imagine what records will appear in that entity once it has become a table. If such a table always contains only one record or meaningless data, the entity is irrelevant.

In the case of **relationships**, especially the situation in the organisation should be looked at: what connections between entities are one interested in? Only the relationships wanted by the organisation are included in the ERD.

Deepening

In addition to the theory, the following can be noted:

- Attributes of relationships are usually **dates** or **numeric data**
- N-to-m relationships (also multidimensional) can possibly be replaced by an entity, i.e.:

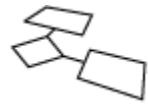


- We use a multidimensional relationship when we are interested in the joint coupling of more than two entities
- A multidimensional relationship can only be used when each record from that relationship always occurs up to once (T1) in relation to the connecting entities

Pitfalls

When designing an ERD, the following things should be accounted for:

- sometimes we accidentally model attributes as entities
- sometimes we accidentally model entities as attributes
- when using the relationship IS_A we need to take into account a number of rules
- in an ERD's attribute list, we are not yet allowed to indicate keys and perform any restructuring
- An attribute list does not include **process data**



Assignments

The only way to learn to create ER diagrams is to do it. Therefore, in addition to the theory, these assignments are included in the reader. The assignments follow the construction of the theory: first only entities, relationships and attributes, then extended with cardinality, special relationships and translation by relational model. All assignments are based on a historical database, unless otherwise stated.

■ Assignment 1

In a library it was decided to keep all the data on the loans in one large table. Below are some records from that table.

LOANS				
ISBN	Title	Author	Loaned to	Date
930564445	The Romans	P. Hallway	A. Jannes	01-02-2002
930564445	The Romans	P. Hallway	J.J.S. Dokal	23-05-2002
600288889	Bockbins	A. Christo	P. Polstok	12-12-2002
930563330	Old Egypt	P. Hallway	A. Jannes	12-03-2002
...

- a. What problems will arise when the loans are registered in this way?
- b. What is the cause of these problems and what is this phenomenon called?
- c. The problems mentioned can be solved by splitting up the table. In which tables should the table OF LOANS be split?
- d. On paper, create the tables from question c and specify which keys they refer to each other (add new attributes where necessary to function as key).

Entities, relationships, and attributes

■ Assignment 2

A company wants to build a database to keep track of its projects. The situation is as follows:

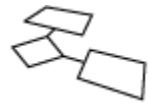
In a company, all employees work on projects. 1 to 20 people are working on such a project. Each employee is part of a maximum of 2 projects. One would like to keep track of which projects employees work on.

- a. Determine which entities (tables) are needed for the database and draw it.
- b. Determine which relationship(s) (references) those entities should have with each other and draw them between the entities of question a.

■ Assignment 3

A school uses teachers to write teaching materials. One of these teachers is Mrs. drs. M. Hendriks from Leiden. Between 1 October 2001 and 6 January 2003, she wrote her *Course for Beginners*, which has the code CVB27. Over time, most teachers write a lot of teaching materials, but not all teachers write teaching materials. Teachers always work on the teaching material on their own. Of course, not all teaching materials are written by teachers.

- c. What entities and relationships do you recognize in this description?
- d. And which attributes?
- e. Draw the ERD including attributes.



Cardinality

■ Assignment 4

Below you will find an ERD that is made for a theatre.

A theatre keeps track of which people visit the performances. A person can visit multiple performances over time. Not everyone who is included in the person file has visited a performance before. Fortunately, audiences always show up for every performance.



Assignment: Indicate the functionality in the ERD above.

■ Assignment 5

- a. In the assignment 2 ERD, indicate the functionality.
- b. In the assignment 3 ERD, indicate the functionality.

■ Assignment 6

- a. In the Assignment 2 ERD, indicate totality.
- b. In the Assignment 3 ERD, indicate totality.
- c. In the Assignment 4 ERD, indicate totality.

Complete ER diagrams

■ Assignment 7

In a hospital, all patients are in a room: sometimes alone, sometimes with several. It sometimes occurs that a room is empty. For patients one is keeping track of a patient number and the name. For rooms a room code and the number of beds is kept track of. Also, one often wants to know when a patient has been brought in the room and when he has left. Patients are sometimes transferred to another room.

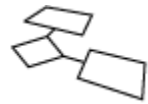
- a. Determine which entities, relationships, and attributes are needed to build a database for this hospital with the ability to register the room where each patient is located (or has been located over time).
- b. Draw the ERD, including cardinality and attribute list.

■ Assignment 8

A library would like to keep track of the data about its books in a database and gives you the following description of the organisation:

For books the ISBN and the title are kept track of. It is also recorded by which publisher the book was published, in which year this happened and who wrote it. Books are sometimes written by several authors, but in this library only one of the authors is registered. For the author one only keeps track of the name, and for the publisher the company name and the location. New publishers and authors are not registered until they publish a book.

- a. Determine which entities, relationships, and attributes are needed to build the database for this library.
- b. Draw the ERD, including cardinality and attribute list.



■ Assignment 9

The database you designed for the library in assignment 8 is received so well that they decided to have it extended by you with a lending system:

For example, one or more copies of each book in the library are present (for example, of the book "*Hocuspocus*" there are no less than 8 copies on the shelf because it is very popular). Each copy has a unique copy number.

Copies can be borrowed several times over time by the members of the library. Not every copy has been loaned out before. A member can of course borrow several copies over time and is only registered as a member on the first loan. One would also like to register on which date a copy is lent (it can then be inferred when it needs to be returned).

Finally, for members it is registered: the member number (is also on the library pass), the name, telephone number and address (street name, house number, postal code and city).

Assignment: Now extend the assignment 8 ERD with the lending system described above.

■ Assignment 10

After the work you did on assignment 9, the library is completely euphoric. Finally, that is why you are also granted the opportunity to design the reservation system:

Members can reserve multiple books, and a particular book may have been reserved by multiple members. The date is being kept track of for such a reservation. This is necessary to determine who takes precedence when multiple members reserve the same book.

Assignment: Now extend the assignment 9 ERD with the reservation system described above.

Special relationships

■ Assignment 11

For years, all buildings have been registered at the municipality of *Knuppeldam*. This registration has always been done on pieces of scrap paper and has become a reasonable mess by now. Now, with the upcoming elections, one wants to put things in order quickly by starting to keep track of the buildings in a database. The mayor gives you the following description:

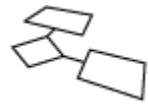
The land registry number and address (street and house number) are registered for each building. Furthermore, for business premises the square meters (m²) are tracked, while for residential buildings one is interested in the number of rooms.

Assignment: Using the above description, create an ERD for the database with buildings and thus save the election campaign of the mayor's party.

■ Assignment 12

At an airline, a pilot (pilot number, name) can be coached by one or more other pilots. A pilot can never coach more than one pilot, and not all pilots have enough time or experience to coach another pilot.

Assignment: Create an ERD for the airline's pilot guidance database.



■ Assignment 13

A large conference centre asks if you want to build a conference database for them. From the organisation you have learned the following:

For the conferences are known:

- date and place
- at a conference, at least one speaker speaks, but usually several speakers speak

For the speakers at the conferences is known:

- name and title (professor, doctor etc.)
- all speakers speak at one or more conferences

Speakers, of course, always talk at such a conference on a particular subject of which the following is known:

- title and brief description of the subject matter
- a topic can be discussed more often over time
- some topics are never discussed

Assignment: Create an ERD for the conference database.

■ Assignment 14

The computer chip factory *Wintel* has 643 machines. In Excel files it is currently tracked when which machine is being repaired, but this way of registering is starting to get out of control. Therefore, you are asked to design a database for this based on the following description:

There is a team of highly specialized repairers that performs repairs on the machines. A repairer can repair several machines and all repairers have already done a repair. In principle, machines are always repaired by the same repairer, but if they are absent (sick, holiday etc.) it is done by another. For the repair the date is registered, and for the repairer the name. Some machines are so solid or new that they have never had to undergo a repair.

A machine number and the function of the machine are kept track of for all machines. Of the 43 machines that make processors, it is also registered what manufacturing technology they use (0.18 microns, 0.13 microns, etc.).

Assignment: Create an ERD for the description above.

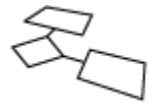
■ Assignment 15

An information analyst has been researching the subscription registration of a telephone company. Below is an excerpt from the research report he created.

"At the telephone company, customers can have multiple subscriptions. An initial interview showed that customers are only registered as customers when they subscribe. For a customer one keeps track of the name and address (street, postal code and city) and the date on which each subscription has started. A certain subscription can of course be bought by multiple customers. Some subscriptions have just been introduced and do not have any subscribers yet, according to the documentation of the marketing department."

"There are two types of subscriptions: for mobile phones and for fixed landline connections. For each subscription a unique code, name and monthly price is being kept track of. Furthermore, the SIM card number of mobile subscriptions, and a benefit number (a phone number provided by the customer that can be called at a lower rate) of fixed connections is tracked. Finally, the rate (per second) of both types of subscriptions is kept track of."

Assignment: Make an ERD of this phone company's subscription registration.



■ Assignment 16

Housing association *Scrapsight* wants to keep track of its data about members and their registrations in a database. The managing director will give you the following information:

Currently, members' data is recorded on member cards. When a member enrolls, in addition to the membership card, a so-called registration form is filled in (see images below).

Housing association <i>Scrapsight</i>	
MEMBER CARD	
Member number 5689	Date of birth : 12-06-1980
Name : A. De Bruin	Phone : (0233) 564954

Housing association <i>Scrapsight</i>	
REGISTRATION FORM	
Date:	02-02-2002
Member number 5689	
Name : A. De Bruin	
Date of birth : 12-06-1980	
Phone : (0233) 564954	
<i>Register as a house-seeker:</i>	
Type of house : Single-family house or flat	
District code(s) : B, E, G	
Max. rental price € 450,-	
Comments : Would love a swimming pool	

Assignment: Based on the above information and forms, create an ERD for the housing association database.

Translating into relational model

■ Assignment 17

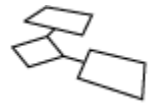
A school wants to build a database for tracking its student and course data. Below is the design that was created for this:



Attribute list

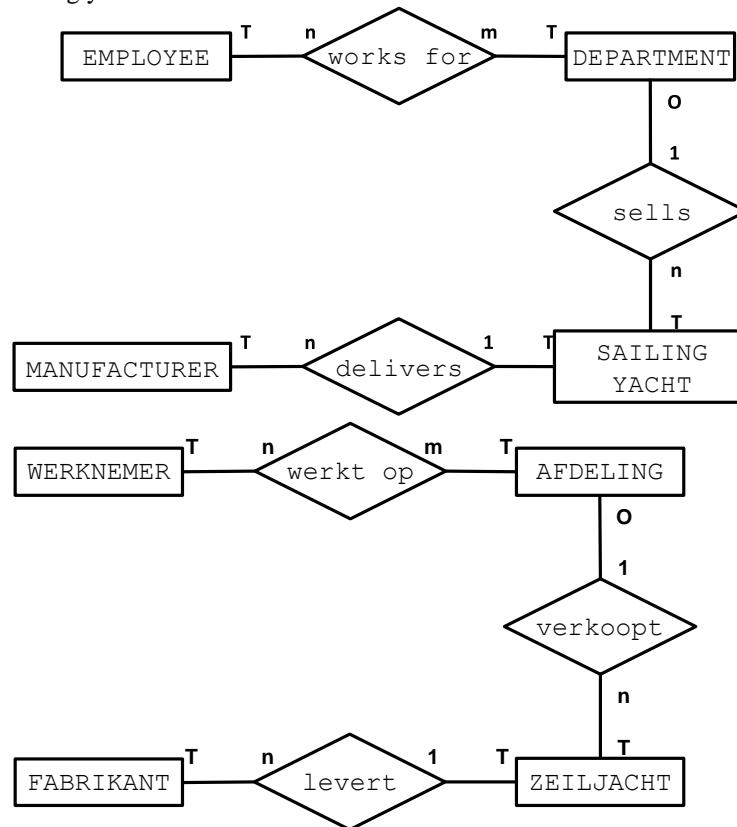
STUDENT	: student number, name, phone
COURSE	: course code, name
follows	: cohort

Assignment: Translate the ERD above into the relational model.



■ Assignment 18

The ERD below is made for a company that sells sailing yachts. There are a number of sales departments that each specialize in selling one particular type of sailing yacht.

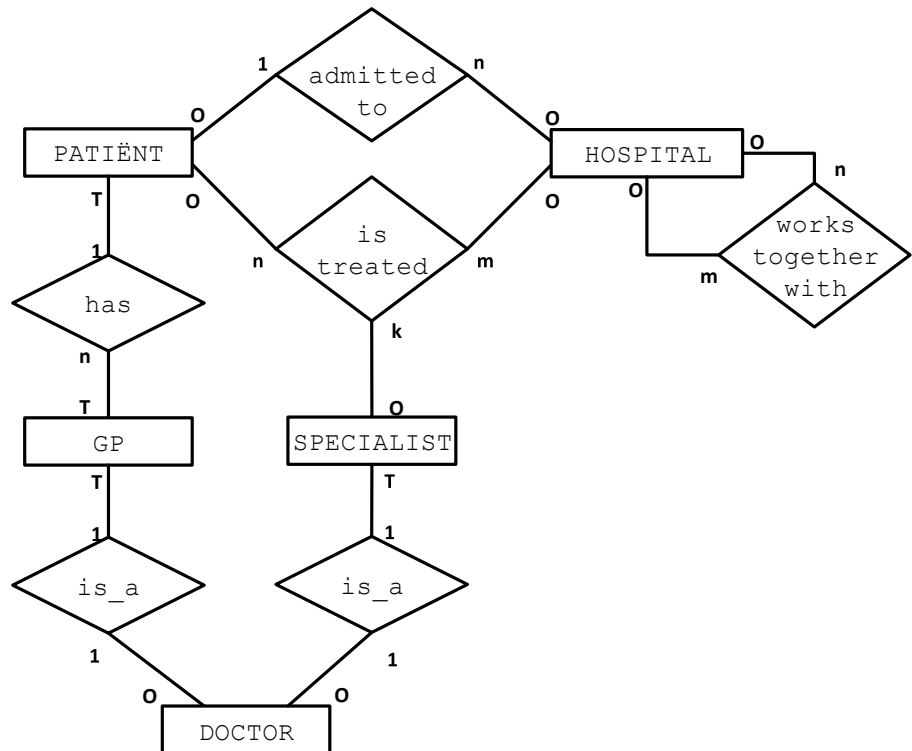
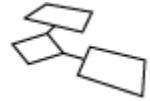


Attribute list

EMPLOYEE	: name
DEPARTMENT	: department code, name, telephone
SAILING YACHT	: description, suggested retail price
MANUFACTURER	: name, e-mail
sells	: sales price
supplies	: purchase price

Assignment: Translate the ERD above into the relational model.

■ Assignment 19



Attribute list

PATIENT	: pcode, name
HOSPITAL	: hcode, name, place
DOCTOR	: dcode, name, city
GP	: practice size
SPECIALIST	: specialism
admitted to	: date

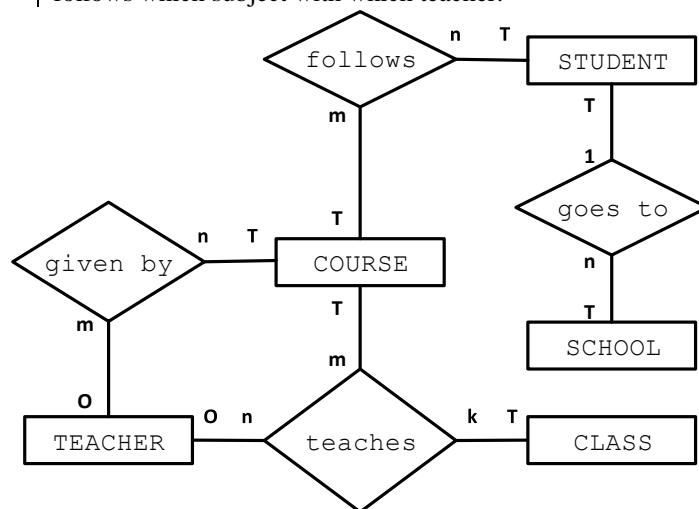
Assignment: Translate the ERD above into the relational model.

Design ER diagrams

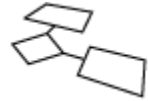
■ Assignment 20

The ERD below was created for a school based on the following description:

The school would like to keep track of which students are studying and in which class each student is in. Furthermore, one wants to know which class follows which subject with which teacher.



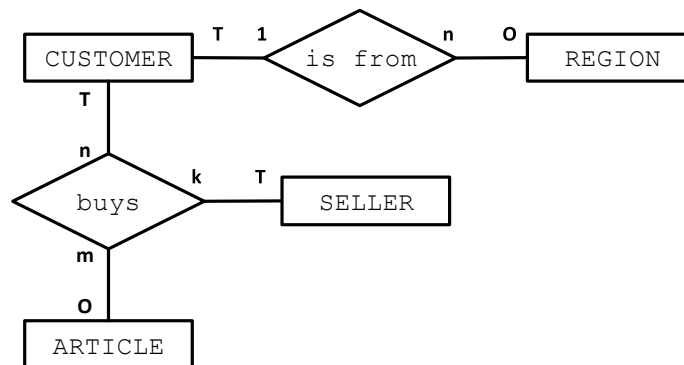
Assignment: Improve the above design in such a way that it matches the description and will provide a good database.



■ Assignment 21

The ERD below was created for the marketing department of a wholesaler based on the following description:

The marketing department is interested in a couple of things about customers (their name and the region where they come from and in which country that region is located) and when they have purchased certain articles. For articles one keeps track of the price and description and which article group the article belongs to.



Attribute list

CUSTOMER : name, country

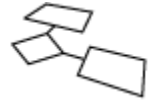
REGION : name

SELLER : name

ARTICLE : description, price, article group

buys : date

Assignment: Improve the above design in such a way that it matches the description and will provide a good database.

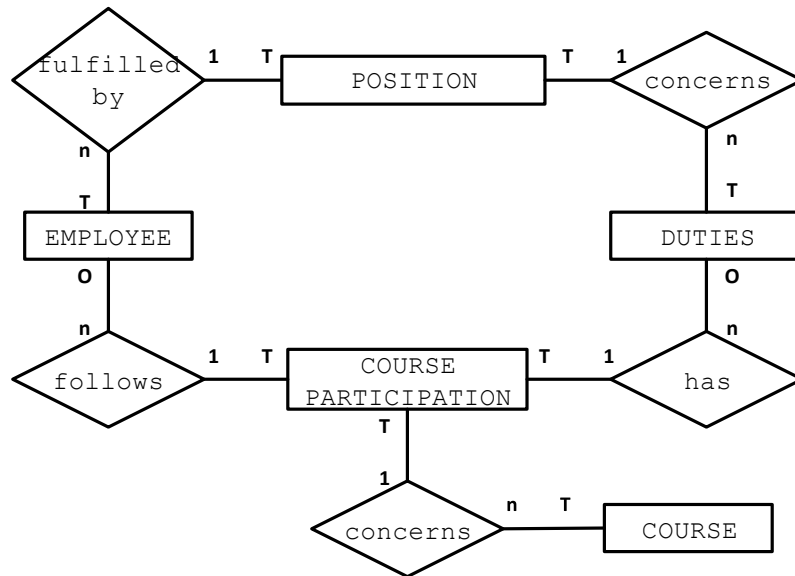


■ Assignment 22

The following ERD is made for the following situation:

For employees the positions they hold and which courses they have participated in for their position are to be registered. Thus, one is interested in the following relationships:

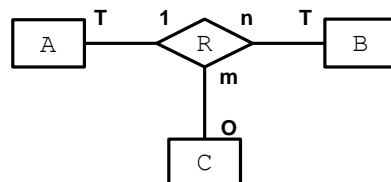
- what positions an employee has held over time
- what courses an employee took part in when they held a certain position



Assignment: The above ERD complies in principle but is unnecessarily complicated. Make the design simpler.

■ Assignment 23

Let's say that we decide to build a database using the ERD below to build a database with tables A, B, C and R.



Attribute list after restructuring

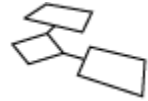
A: As

B: Bs

C: Cs

R: As, Bs, Cs

Assignment: If the table R contains 3 records, how many records does each of the other tables contain maximum and how much minimum?



Additional assignments

■ Assignment 24

Regional transport company *Claxxion n.v.* asked you to design a database for the following situation:

The company has buses and a small number of taxis. The area in which the company operates is divided into a number of areas. Each bus and taxi is serviced by one rayon. Each area maintains a number of buses and taxis. They want to keep track of the following data:

- for a bus: vehicle code, registration plate, description, brand, type and passenger capacity
- for the areas: rayon code and description
- for the taxis: registration plate, vehicle code, brand, type, description and fuel type (gas, petrol, etc.)

A number of employees work in such an area, with all employees (except the director of course) reporting to the employee above them. The employees can be deployed in each area. For each employee one wants to register their name, address, postal code and city.

Assignment: Using the above description, make the ERD for *Claxxion n.v.* and translate it into the relational model.

■ Assignment 25

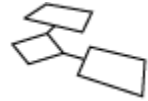
Wholesaler *MultiProcz* trades in computer components. They sell processors, CD-ROM players, monitors etc. to computer stores in the region. They are ready for a new sales system and give the following description:

When an employee of the sales department comes to work in the morning, he first drinks a cup of coffee at the coffee machine and then takes a seat at his desk. When a customer calls, he records the articles mentioned by the customer in a sales order. Such a sales order consists of a number of order lines, whereby the number of ordered articles and the price per piece is recorded in each order line. Of the sales order is further recorded: a unique order no and the date on which it was placed. An agreement is made with the customer about when the order must be delivered at the latest.

For employees and customers one keeps track of the name, address, postal code and city. For employees the salary and employee code are also registered, for customers the credit limit and a unique customer no. The total price of customers' pending orders cannot exceed their credit limit. New customers are only included in the customer base when they place their first order.

The articles are classified in article groups for convenience, such as an article group "processors" and an article group "hard drives." For articles the price, description and a unique article code is kept track of. Of course, one would also like to know how many pieces of each article are in stock and there is often a need to capture additional data for an article (for example for a particular processor that it does not work with Windows 2000, or for a monitor that it has blue dots on the side etc.).

Assignment: Using the above description, create the ERD for *MultiProcz* and translate it into the relational model.



■ Assignment 26

At *Schiphol*, we need a new terminal system: the system that is responsible for capturing the flight data and showing it to passengers via monitors. This is a current database, because it is only about displaying the current flight data on the monitors, what happened in the past (for this system) is not interesting. The system is constructed as follows:

In order to display the correct information on the monitors, one must know from each flight which airline is operating the flight with which type of aircraft. For each flight the flight number is tracked. Furthermore, the flight's departure time and current status (which is stored encoded: D=delayed, B=boarding, T=taking off etc.) is recorded.

The airport (airport code, city) where the flight departs from and the airport it arrives to are also stored. Not every airport is used as the origin or destination of a flight.

For each aircraft the (unique) type (B737, A320 etc.) and the number of seats is also known. Only the airline no and the name (Iberia, KLM, etc.) is kept from each airline. Not every airline (and not every aircraft) in the system operates flights through *Schiphol*.

The terminal screens at the airport should be displayed as shown below:

Departures						time 12:42
flight		departure	from	to	plane	status
IB3485	Iberia	12:30	Amsterdam	Madrid	A320	
BA6654	British Airways	12:40	Singapore	London	B737	delayed
KL644	KLM	12:45	Amsterdam	Frankfurt	B737	boarding
IB6556	Iberia	13:05	Amsterdam	Lima	A340	
TR1324	Transavia	13:10	Frankfurt	New York	B747	

Assignment: Design the ERD for the current database of the terminal system. Also translate the ERD into the relational model so that the database (and then the terminal hardware and software) can be built.