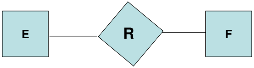
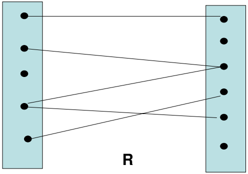
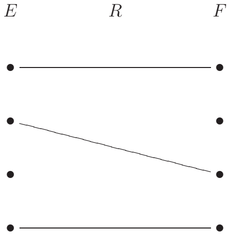
Cardinality Constraints



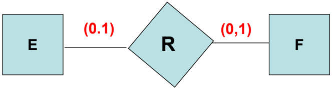


- The model can impose conditions on the number of connections via relationship R between entities E and F
- Common Case:

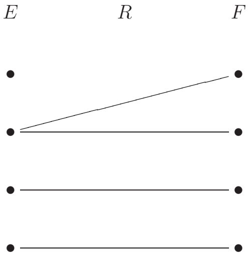


min-card(E,R) $= 0$
max-card(E,R) $= 1$
min-card(F,R) $= 0$
max-card(F,R) $= 1$

- We sometimes say this is a "one-to-one relationship" (meaning "at most one" in each direction)

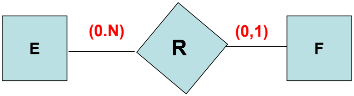


- 1-to-N, N-to-1, etc. Relationships:



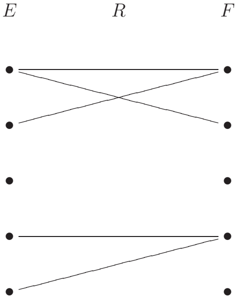
min-card(E,R) $= 0$
max-card(E,R) $=$ N
min-card(F,R) $= 0$
max-card(F,R) $= 1$

N stands for "unbounded"



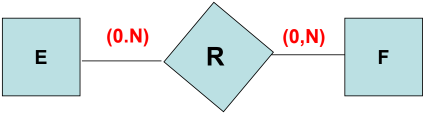
- Each pair on a link beside an entity indicates a lower and an upper bound on the number of connections to the other entity

- N-to-N Relationships:

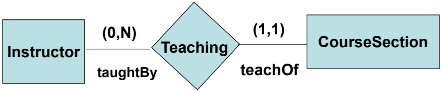


N-to-N Relationships:

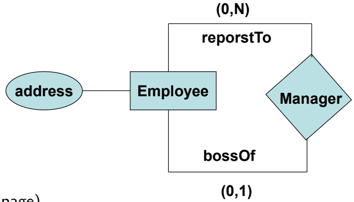
min-card(E,R) = 0
max-card(E,R) = N
min-card(F,R) = 0
max-card(F,R) = N



Some Examples:

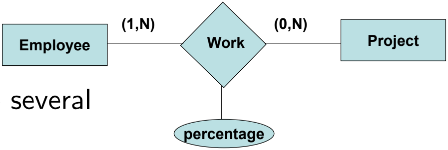


An instructor can teach zero or many course sections, but a course section is taught exactly by one instructor



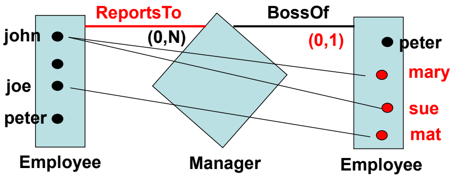
An employee can manage nobody or an arbitrary number of employees

An employee reports to at most one employee (see also next page)



An employee is assigned to at least one project

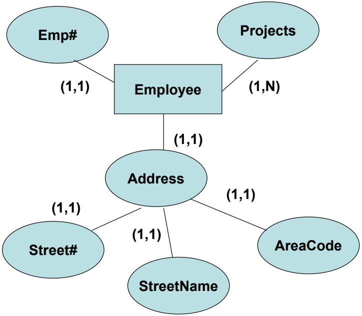
A project can be assigned to several employees or to nobody



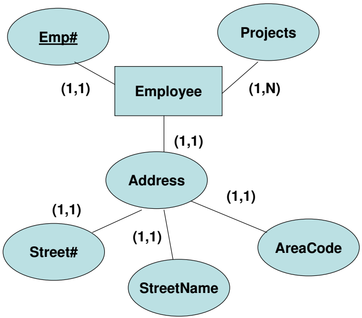
- We can also impose cardinality constraints on attributes

  Including subattributes

  Attribute Projects can be multivalued

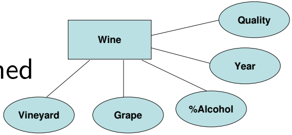


Keys:

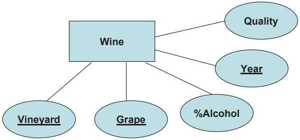


- Underlined attribute Emp# indicates that it is a key for (the attributes of) Employee
- A restriction on the values the attributes for the entity can take
- There cannot be two employees that have the same (value for) employee number, but have different values for the other attributes
- We think of the key as a unique identifier for employees
- Naturally and commonly, its link accompanied by the constraint (1,1)
- Here we have a composite attribute

- A whole set of attributes can be the key for an entity
- When describing wine, the percentage of alcohol and the quality are determined by the attributes Vineyard, Grape and Year



- The key of entity Wine is the set of attributes {Vineyard, Grape, Year}

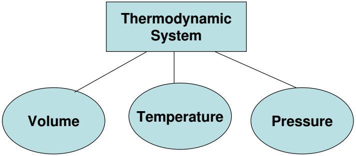


- Same kind of restriction: No two wines with the same values for these three attributes but different values for the other two
- {Vineyard, Year} not a key: it does not determine all the attributes
- Minimality condition: If we declare this set as a key, we understand that no proper subset is also a key

  {Vineyard, Grape, Year, Quality} is not a key, because it properly contains a key

- An entity can have more than one key (if any)
- Boyle's Law for simple gases

$$P \times V \;=\; c \times T$$



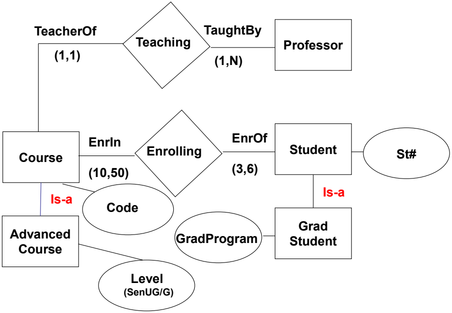
- {Temperature, Volume}, {Temperature, Pressure}, ..., {Volume, Pressure} are all possible keys

  They are candidate keys

  Any of them can be chosen as a primary key

Sub-Entities:

- Sub-entities (subclasses) can be specified using "IS-A"-links connecting entities
- Attributes (properties) are inherited by sub-entities



Sub-entities may have specialized attributes

- Participation in relationships are inherited by sub-entities, e.g. between "Advanced Course" and "Grad Student"
- An "IS-A" label (for a link) has a fixed, built-in semantics

  In contrast to the other, application-dependent labels we have used so far

  Its semantics is a set-theoretic inclusion of classes

Some Remarks:

- ER is a useful but simple modelling language
- It has limited expressive power to specify (create) a model
- In particular, there are limited features to express semantic constraints
- The semantic constraints (so as the whole model) depends on how we (the modelers) understand the outside reality

  And the data associated to it (or emerging from it)
- We as modelers decide what are the relevant entities and relationships

  An entity can participate in more than one relationship (see previous example)
- We have shown data with ER models, but only to illustrate the introduced notions

  There are not data (data items) in an ER model

- An ER model could be seen and exploited as metadata for a relational model emerging from the former
- With an ER model of data we capture static aspects of data

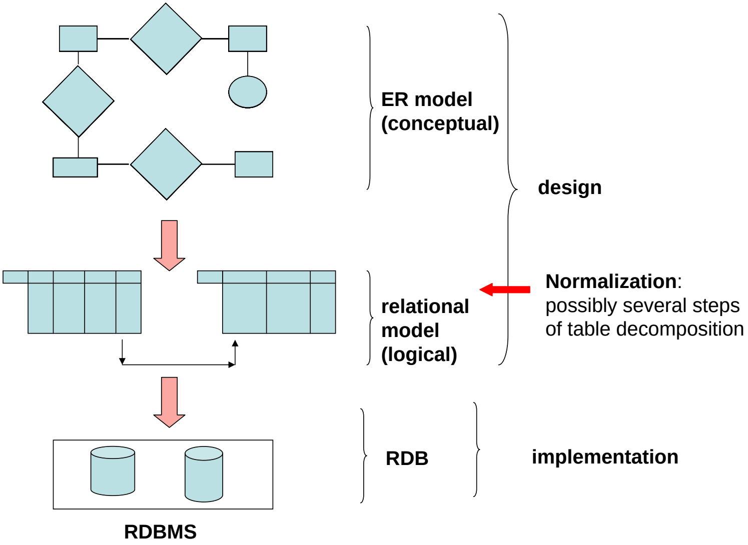
  What about dynamic aspects? (evolution, updates)
- What about the use of data?

  Activities based on data?

  Agents acting on/with data?
- There are more expressive -and still graphical- languages that can be seen as extensions of ER

  For example, UML (Unified Modelling Language)
- Also rich, symbolic languages, such as ontological languages
- Our next goal for now is to use an ER model to produce a relational model, a relational schema (still no data, but later)

  One that can be "improved" after that ...


ER model
(conceptual)
design
relational
model
(logical)
Normalization:
possibly several steps
of table decomposition
RDB
implementation
RDBMS

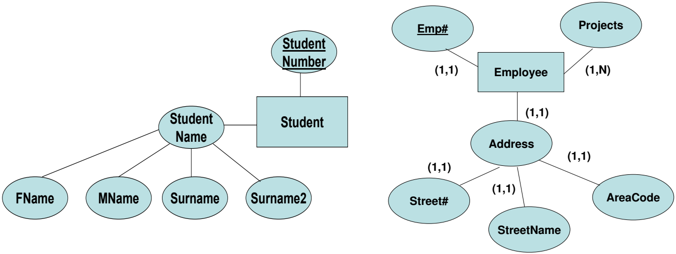
From ER to the Relational Model

- There are several standard guidelines to transform an ER model into a relational model

  We will present them mostly by examples

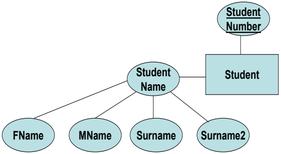
- Semantic constraints on both sides will be critical for the transformation

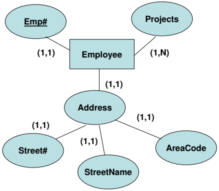
- We start by transforming entities, relationships later on ...



Transformation Rule 1:

1. Each entity is mapped to a relational predicate
   Although without data at this stage, we will call it a “relation”
2. The entity name becomes the relation name
3. The single-valued attributes of the entity become arguments (columns) of the relation
4. If an ER attribute is composite, its single-valued sub-attributes become relational attributes (columns)
5. A unique identifier (key) in the ER model becomes a key for the relation
6. The resulting relational key may contain several attributes, becoming a composite key for the relation





In these examples, we obtain:

Student

Student_Number	FName	MName	Surname	Surname2

Employee

Emp#	Street#	StreetName	AreaCode

- What about the multi-valued attributes?

  Projects in the example?

Transformation Rule 2:

1. For an entity E with identifier P and a multi-valued attribute A, we create a separate relation for A with attributes P and A

- In the second example above, we obtain two tables:

Employee

Emp#	Street#	StreetName	AreaCode

Assigned

Emp#	Projects

- Notice that Emp# is not a key for relation *Assigned*

  Not expected to be one: There may be several projects assigned to an employee

- With extensions this may look like this:

Employee

Emp#	Street#	StreetName	AreaCode
453489	322	El Aromo	687773

Assigned

Emp#	Projects
453489	p586
453489	p321
345566	p333
345566	p666

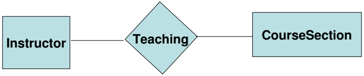
$\overset{?}{\rightarrow}$ 345566 should appear in table Employee

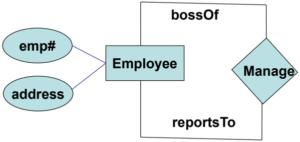
- It is natural to introduce a referential constraint from Assigned.Emp# to Employee.Emp#

  Possibly from Assigned.Proyects to relation for Projects

- Since the referred attribute Employee.Emp# is a key in Employee, we say that Assigned.Emp# is a foreign key constraint in Assigned

- What about relationships?





- A relation for each relationship?

  Maybe ...

- First example:

Teaching	Instructor	CourseSection
	john	comp1805
	john	comp3005
	ken	comp4900
	. . .	. . .

- Second example?

  We may not need to introduce a new relation

- Expand already existing relation for entity Employee

  With an extra attribute

- If we had schema Employee(Emp#, Address), now:

Employee	Emp#	Address	ReportsTo
	453489	···	NULL
	433252	···	453489
	345566	···	453489

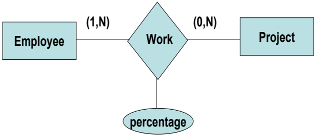
- What about Employee(Emp#, Address,BossOf)?

  Emp# not a key anymore

  Also: redundancy of data

  The same occurrence of an employee number and its description, but for with different subordinates

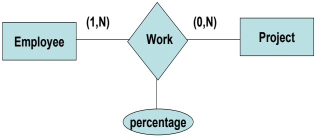
- Cardinality constraints of the ER model play an important role



Transformation Rule 3:

1. If a binary relationship R between entities E, F is N-to-N (as above), then R is mapped to a relation T
2. Relation T contains the keys of the relations associated to the participating entities
3. T will have columns for the single-valued attributes hanging from R (if any)

Example:



Employee	Emp#	Name	ReportsTo
	453489	john	NULL
	433252	rose	453489
	345566	peter	453489

Projects	Proj#	ProjName	Date
	p234	MinSal	3-12-1999
	p983	CNU	8-1-2000
	p328	DBR	6-4-99

Work	Emp#	Proj#	Percentage
	453489	p983	50
	453489	p328	25
	453489	p432	25
	433252	p234	100
	345566	p115	33

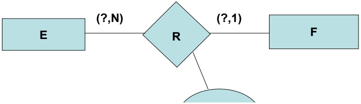
(similar to Example in Chap 1, p. 43)

Assuming ER attribute Percentage is single-valued (otherwise, there is no non-trivial key for Work)

- With referential constraints:

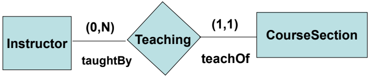
  - From Work.Emp# to Employee.Emp#, and

  - From Work.Proj# to Projects.Proj#



Transformation Rule 4:

1. If binary relationship R between E, F is N-to-1, R is not mapped to a relation
2. We assume, as so far, there are already relations for entities E, F
3. If $\text{max-card}(F, R) = 1$ (F is the "many" side), F's relation will be expanded with columns for the key of E's relation
4. Single-valued attributes of R are also included in F's relation



Instructor	Inst#	Name	Ext.
	i23	john	2345
	i34	peter	3452
	i89	claire	9087

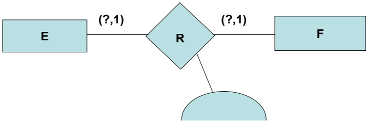
↑

CourseSection	Sec#	Inst#	Course	Time	Room
	s234	i23	CS102	TTh3	JG2
	s543	i34	CS213	MW1	SJ4
	s398	i23	CS214	MW2	SJ4

- With a single table, info about each instructor would be repeated many times
- Notice the foreign key constraint
- If F has optional participation, that is card-min$(F, R) = 0$, NULL values may appear

  For Inst# in CourseSection

  To avoid this, one could create a relation for R



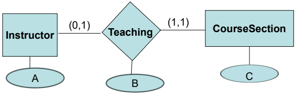
- Now we have a 1-to-1 relationship R between entities E, F

Transformation Rule 5:

(a) Assume E or F have optional participation in R

1. According to Rule 1, there are relations T_E, T_F for E, F, resp.
2. R is represented by adding columns to T_E (or T_F) with the key of T_F (or T_E)

   Both options are possible
3. Single-valued attributes of R are also added as columns



- We already have:

Instructor	<u>IName</u>	A	···
	john	···	···
	peter	···	···
	sue	···	···

Section	<u>SName</u>	C	···
	cs001	···	···
	ar003	···	···
	in080	···	···

- Expand one of them:

SectionPlus	SName	C	···	IName	B
	cs001	···	···	john	···
	ar003	···	···	sue	···
	in080	···	···	peter	···

- With foreign key from SectionPlus.INAme to Instructor.IName

  There may be names in Instructor that do not appear in SectionPlus, but not the other way around

- Since B is single-valued (and the (1,1) condition), we could keep SName as a key for SectionPlus

- What about

InstrPlus	IName	A	···	SName	B

?

- As above, but NULL may appear ...

(b) Assume both E and F have obligatory participation in R

That is: $\text{card-min}(E, R) = \text{card-min}(F, R) = 1$

1. Both entities can be combined into one single table

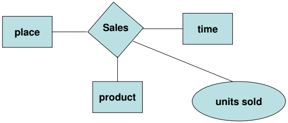
   To avoid foreign key constraints

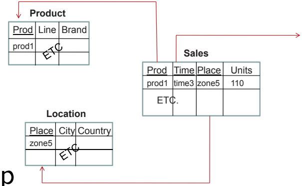
2. The single-valued attributes of R are also added to this table

   Multi-valued attributes of relationships are handled as with Rule 2

Some Final Remarks on the Transformation:

- We will not cover in detail the transformation of relationships of arity greater than 2
- The idea:





1. Create one relation for the relationship
   Columns are the keys of participating entities
2. With one foreign key per entity
   Not always all needed, depending on cardinality constraints
3. The set of keys from item 1. is the key for the new relation
   Same comment as in previous item (see example on page 44)
4. Single-valued attributes for the relationship are added to new relation

- Not all the authors (or practitioners) exactly agree on the procedure for passing from an ER model to a relational model
- We have given some heuristic transformation rules (guidelines) without being exhaustive

  We did not give a full algorithm
- Most important is to be aware and gain intuitions about the issues involved

  Among them: redundancy of data and missing data (the latter giving rise to occurrences of NULL)
- The relational ICs generated during the transformation rules become part of the generated relational schema

  Among them: keys, referential constraints, and foreign key constraints
- The relational schema resulting from the transformation with origin in ER will probably be further transformed

- This is the "Normalization Process"

  For which, the following becomes critical:

  1. The existing (generated) ICs
  2. Newly identified functional dependencies (FDs)
     (not necessarily keys, that are a particular kind of FDs)
  3. The issues of data redundancy and missing data
  4. The related issues of update anomalies

- Basically two ways to reach such a good relational model

  1. Start from an ER model, and apply the transformation rules
  2. Start right away with a relational schema
     One with wide relations (many attributes), maybe a universal relation
     Next, apply normalization techniques to reach a new and better relational schema

- In practice, the two techniques are combined
    1. The initial ER model is transformed into a RM
    2. The resulting RM is improved applying normalization techniques
- We covered the basics of ER, but the ER modelling language is much richer (c.f. any textbook)