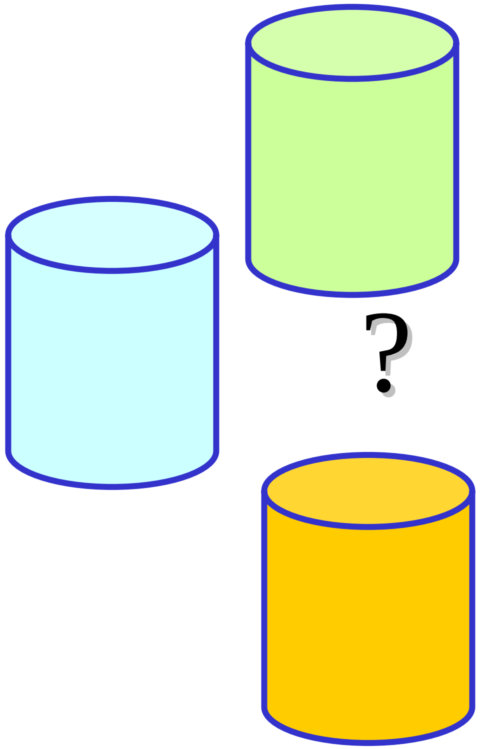
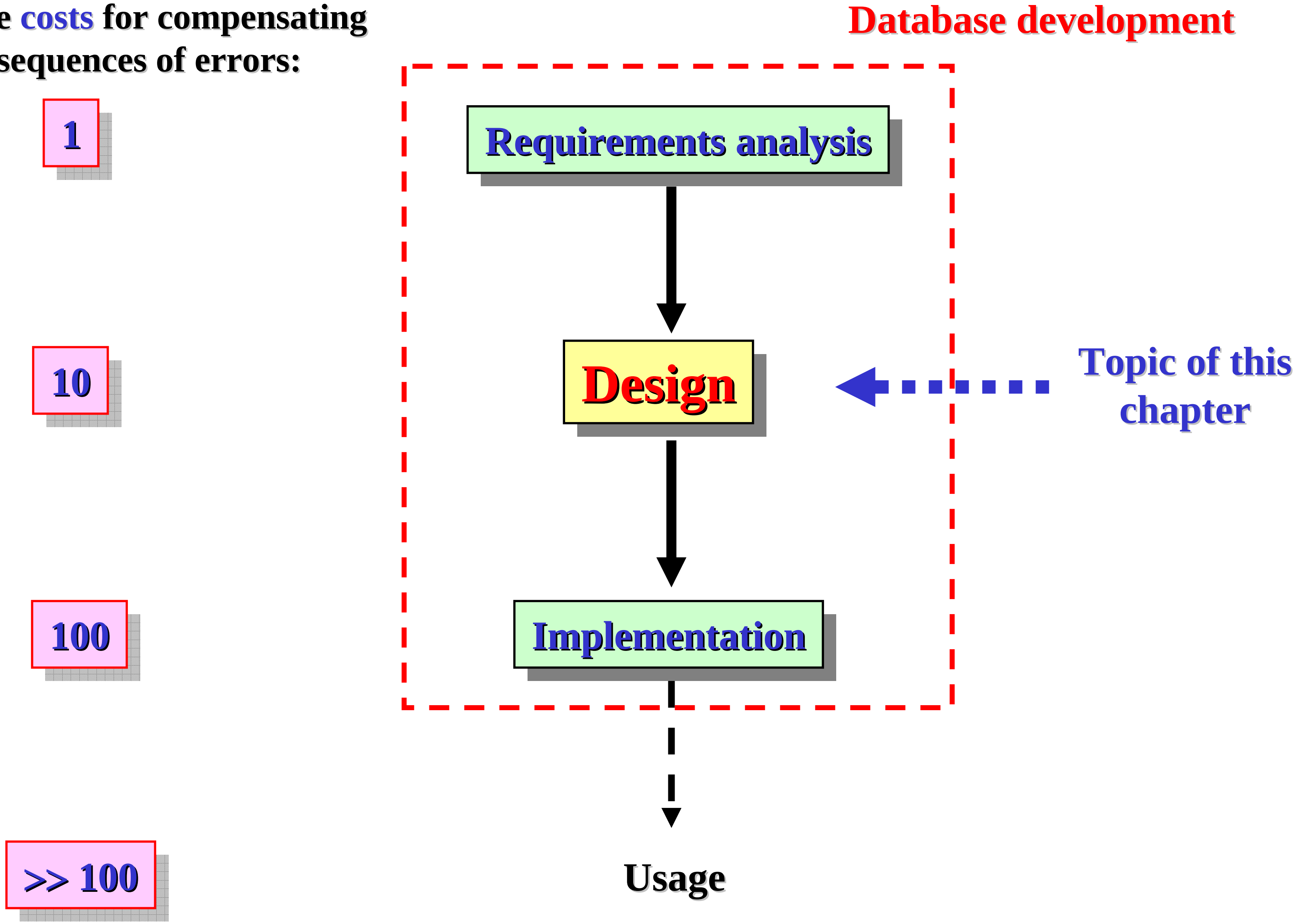
Foundations of Information Management (WS 2008/09)

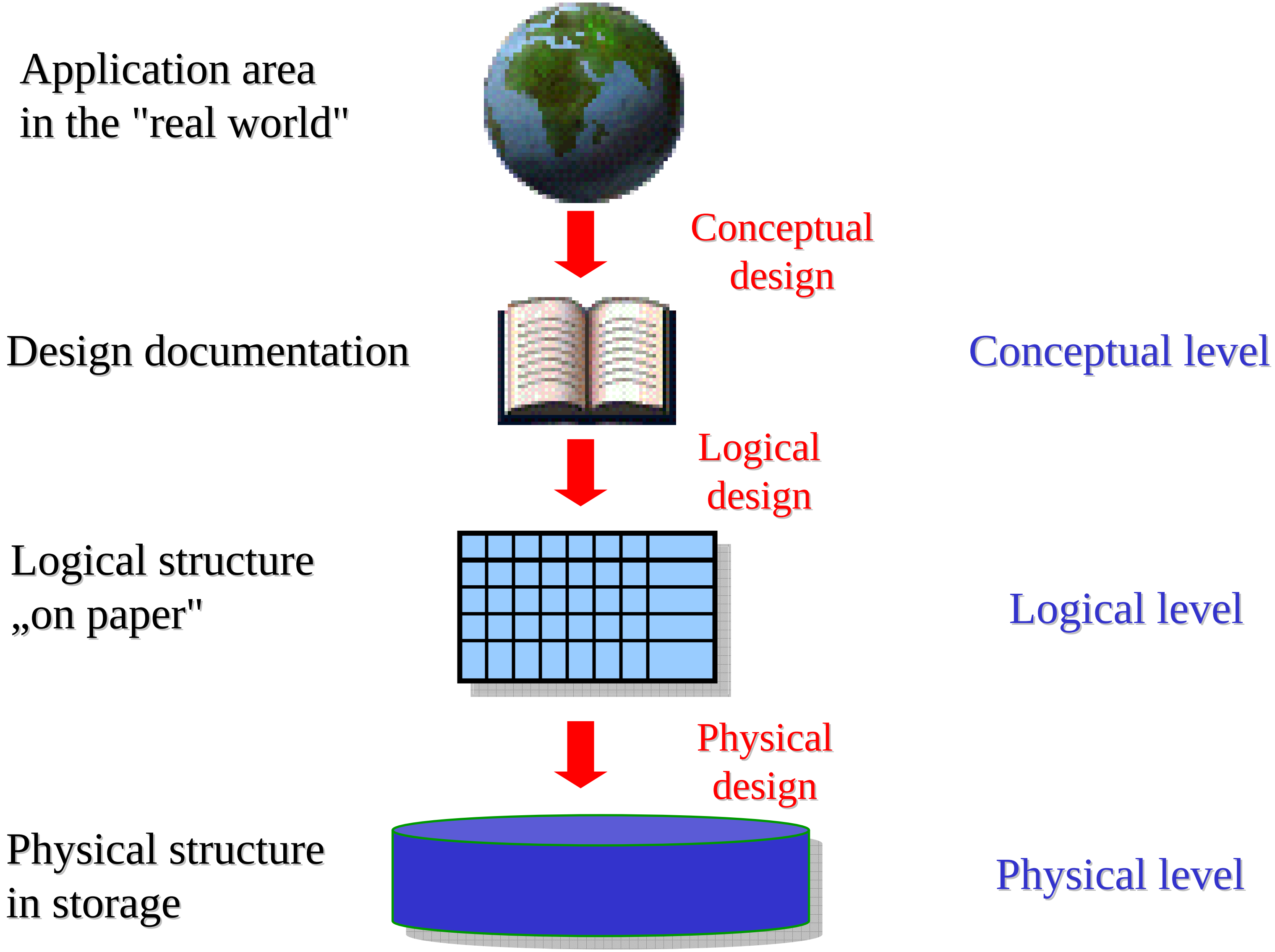


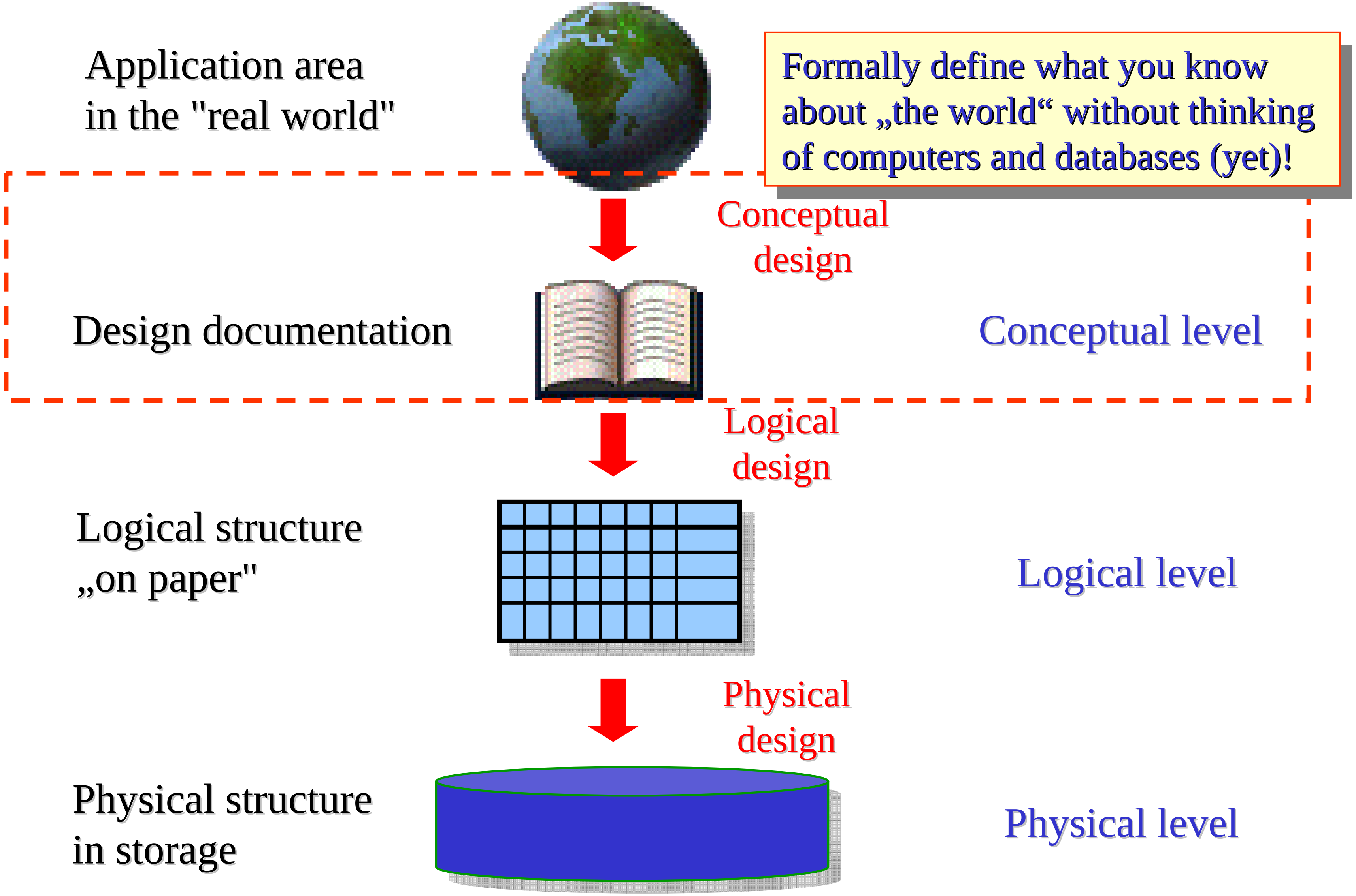

– Chapter 4 –

# Principles of Database Design

Relative **costs** for compensating the consequences of errors:




Application area
in the "real world"
Conceptual
design
Design documentation
Conceptual level
Logical
design
Logical structure
„on paper"
Logical level
Physical
design
Physical structure
in storage
Physical level

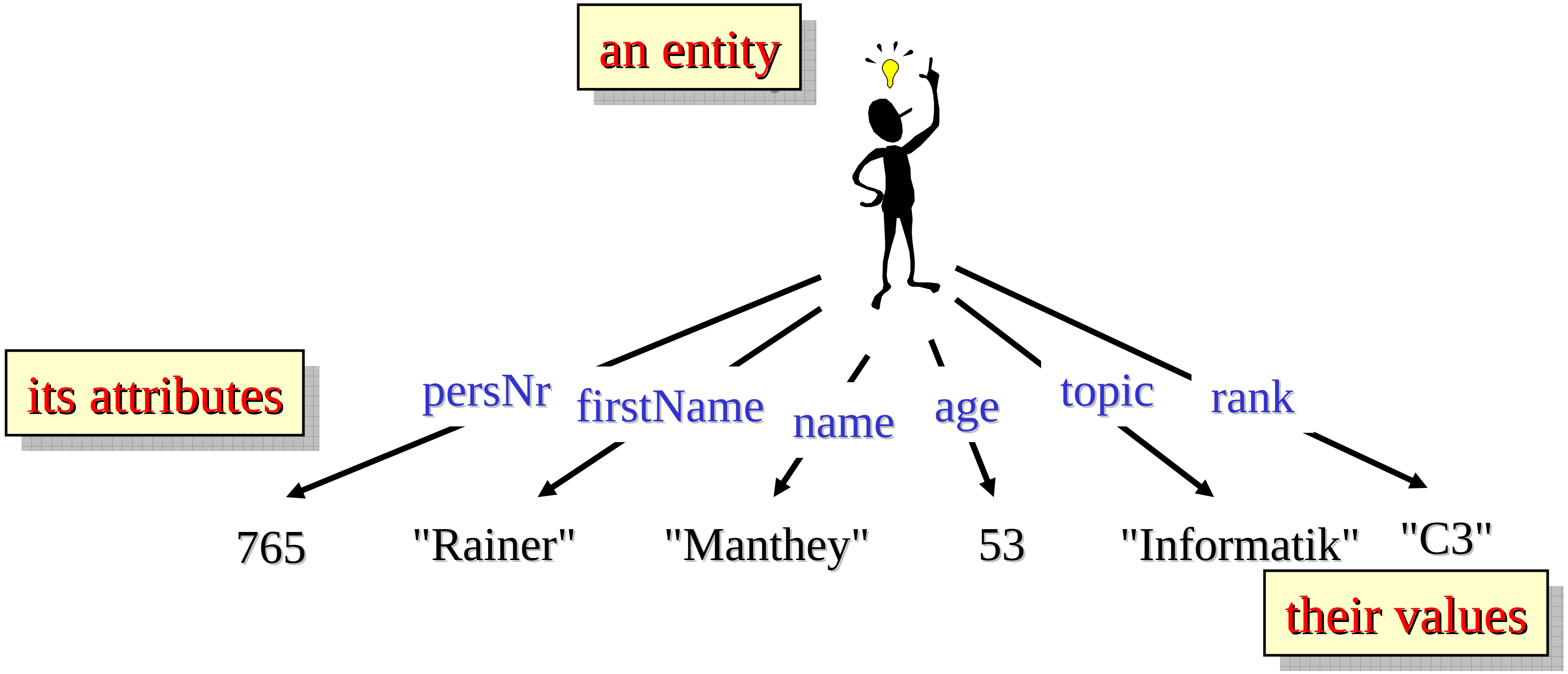

Application area
in the "real world"
Formally define what you know about „the world“ without thinking of computers and databases (yet)!
Conceptual design
Design documentation
Conceptual level
Logical design
Logical structure
„on paper"
Logical level
Physical design
Physical structure
in storage
Physical level

- Entity-Relationship data model (ER model):
    - Proposed in 1976 in a paper by Peter Chen
    - Graphical notation for application modeling (ER diagrams)
    - Independent semantic data model
      (aiming at the meaning of concepts in real world)
    - Predecessor of today‘s object-oriented data models
    - Extremely successful as a means of „pre-design“ of relational DBs

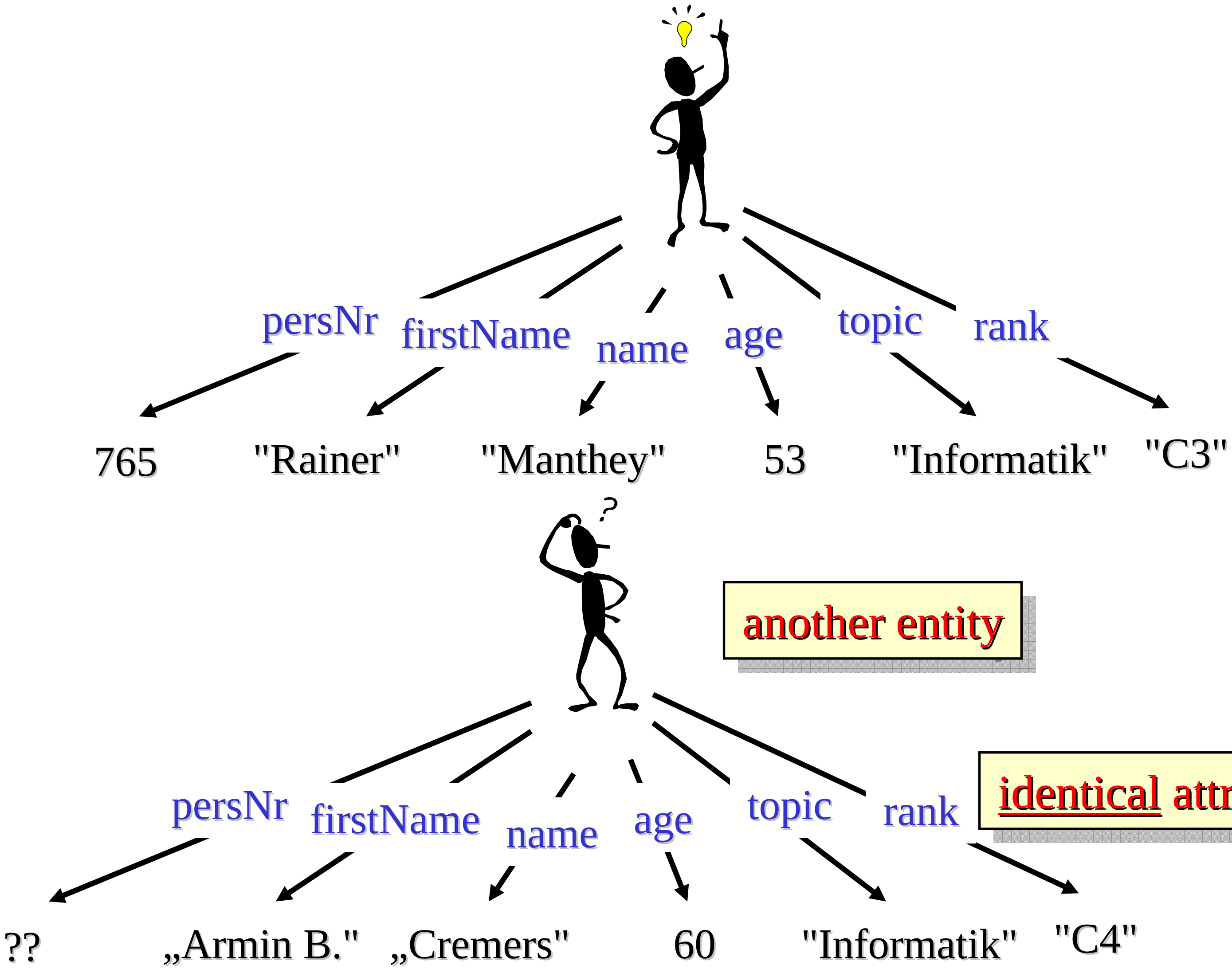
- The ER model offers few very simple and basic concepts:

    - Entities (objects), characterized by attributes (properties)
    - Binary or n-ary relationships between entities,
      possibly characterized by attributes as well
    - Often not mentioned explicitly, but important and basic:
      Values: printable symbols as values of attributes;
      play a subordinate role (characterizing objects)
    - Roles: Names for the special meaning an entity has within a
      relationship

Each entity is completely characterized
by the values of all its attributes.



# Two similar entities

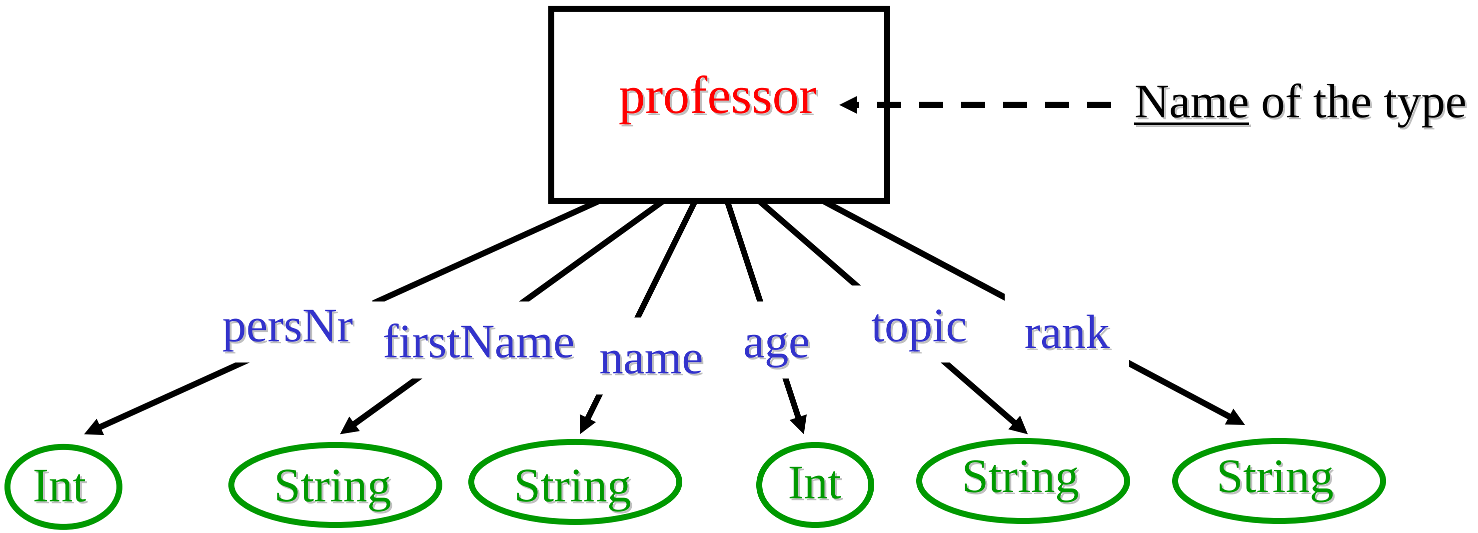



another entity

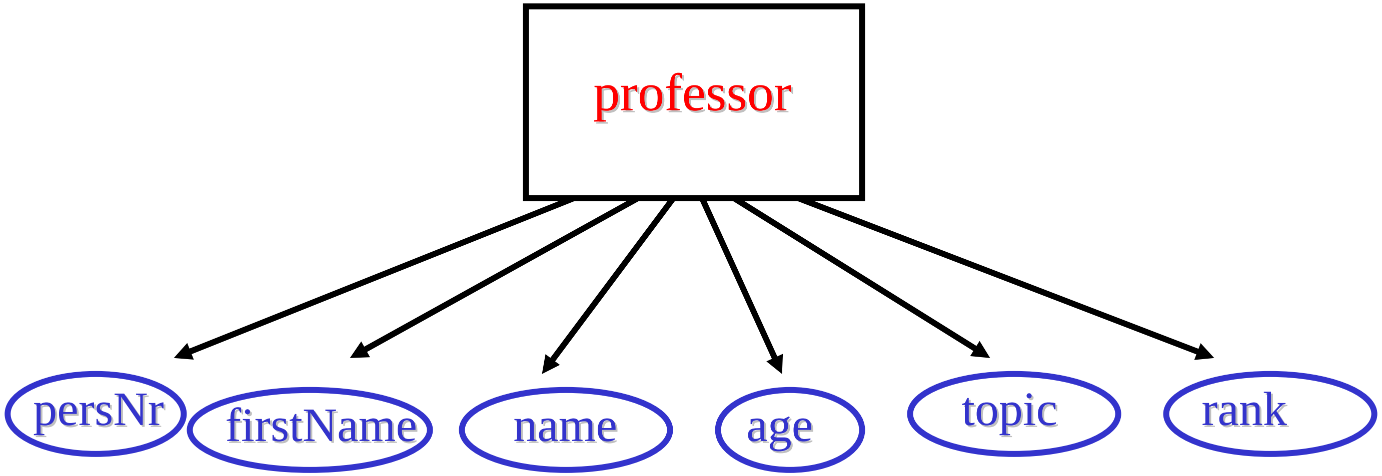
<u>identical</u> attributes

partially different values

- Similar entities can be combined into entity types.
- „Similarity" requires at least identical attribute structure.
  (Attribute names and corresponding value domains are identical.)
- Entity types are graphically represented by rectangles. Attributes label the line connecting an entity type and a value domain (often symbolized by an oval):



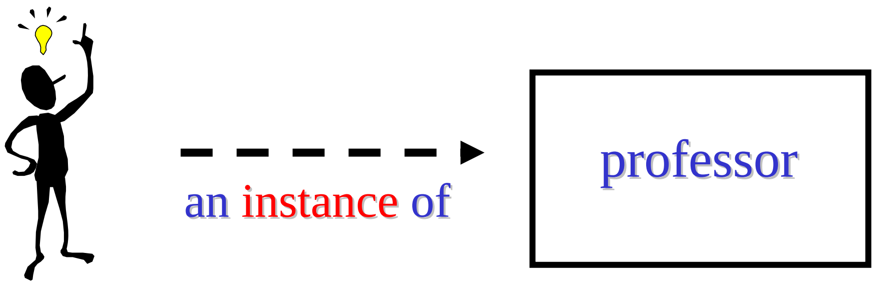
- Domain names are often omitted (in case they are obvious or irrelevant). Instead attributes are placed inside ovals:



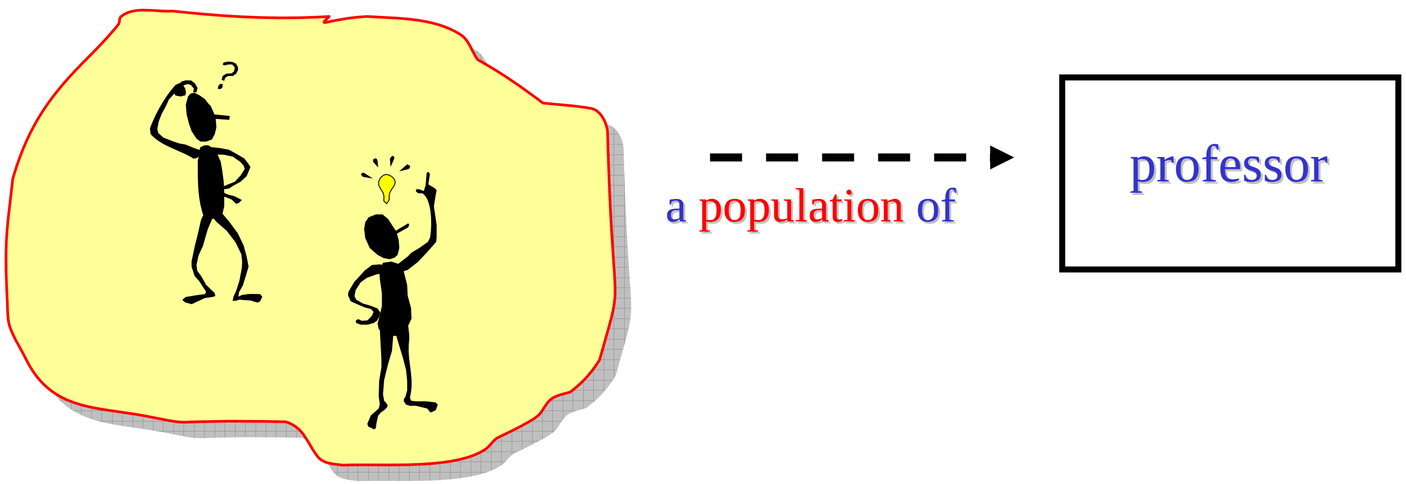

- In larger diagrams, the attribute structure is often entirely omitted in order to save space (or is written down in abbreviated form only):


persNr, firstName, name, . . .

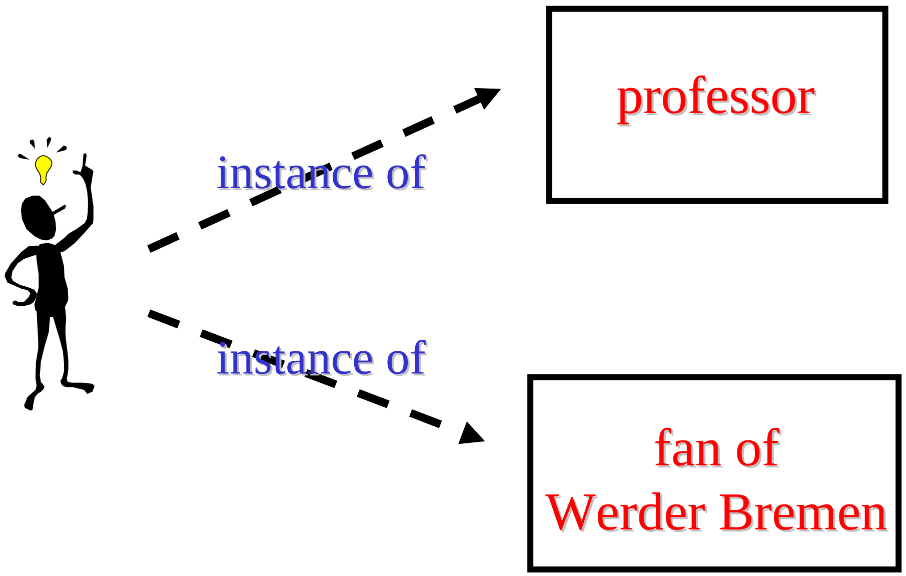
- Each entity „belongs to" at least one entity type:
  It is called an instance of this type.




- The set of all current instances of an entity type is calles its current population.



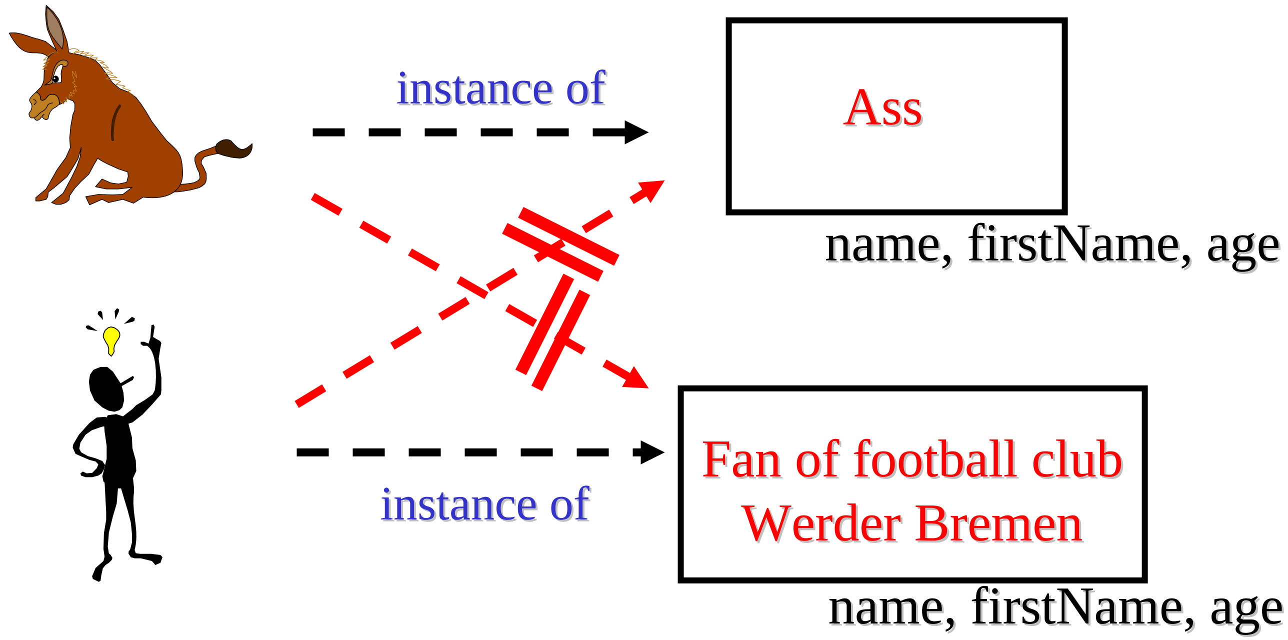
- One and the same entity can be an instance of various entity types.




- In such a case, the attributes of the different types of this entity may well be quite different.

## Classification in presence of identical attributes

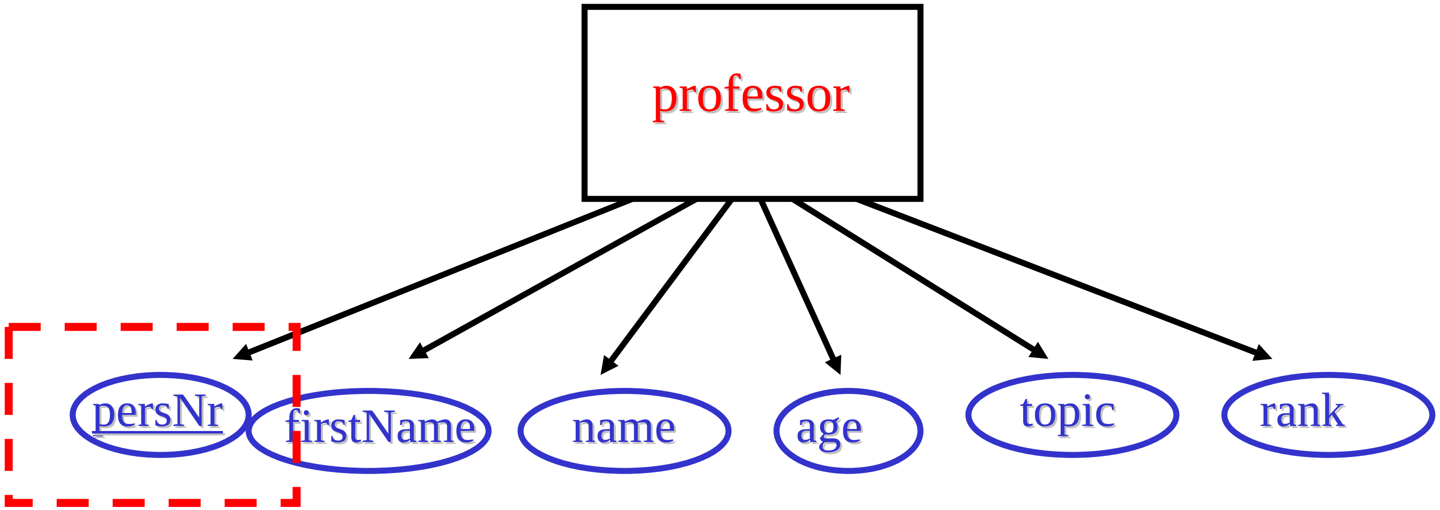
- Entities with the same attributes do not at all have to be instances of the same type!




- Almost always additional classification criteria are required, usually <u>not</u> derivable from the attribute structure.

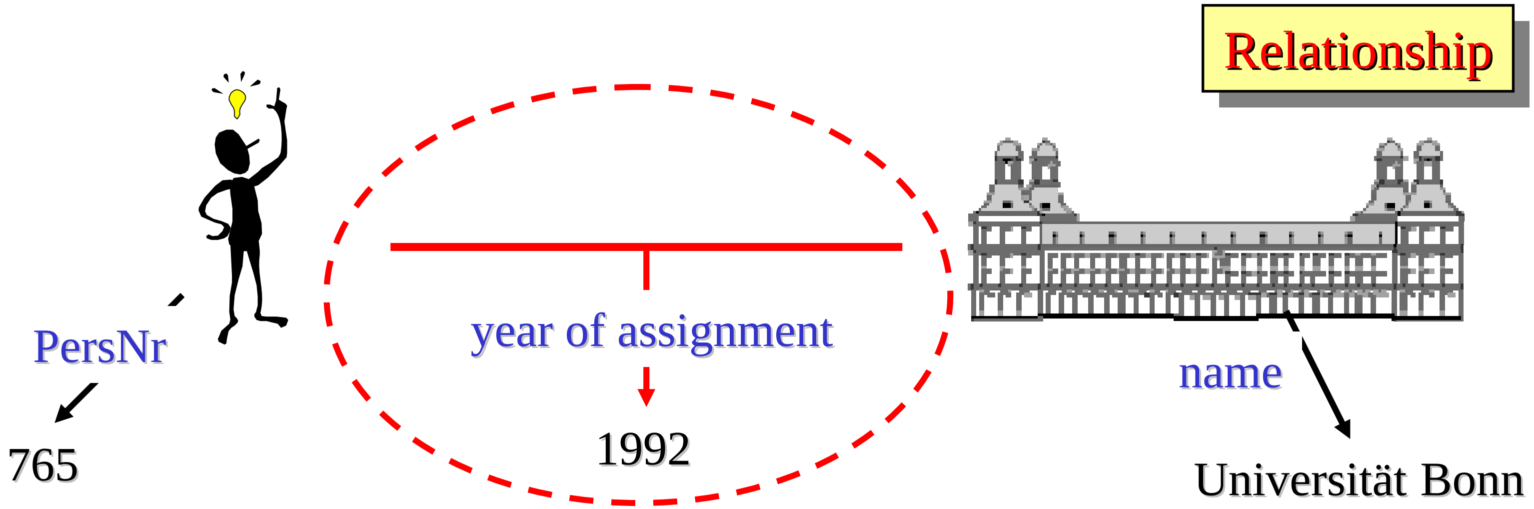
- As in the relational model (Access, SQL), there are usually one or more attributes per entity type the values of which are sufficient for uniquely identifying each instance:

Key attributes




- (Primary) key attributes are usually underlined in an ER-diagram.
- Keys ought to be "minimal" (no attribute can be omitted).
- A distinction between primary key and other candidate keys is not made in the ER-model, even though it would be useful to do so.

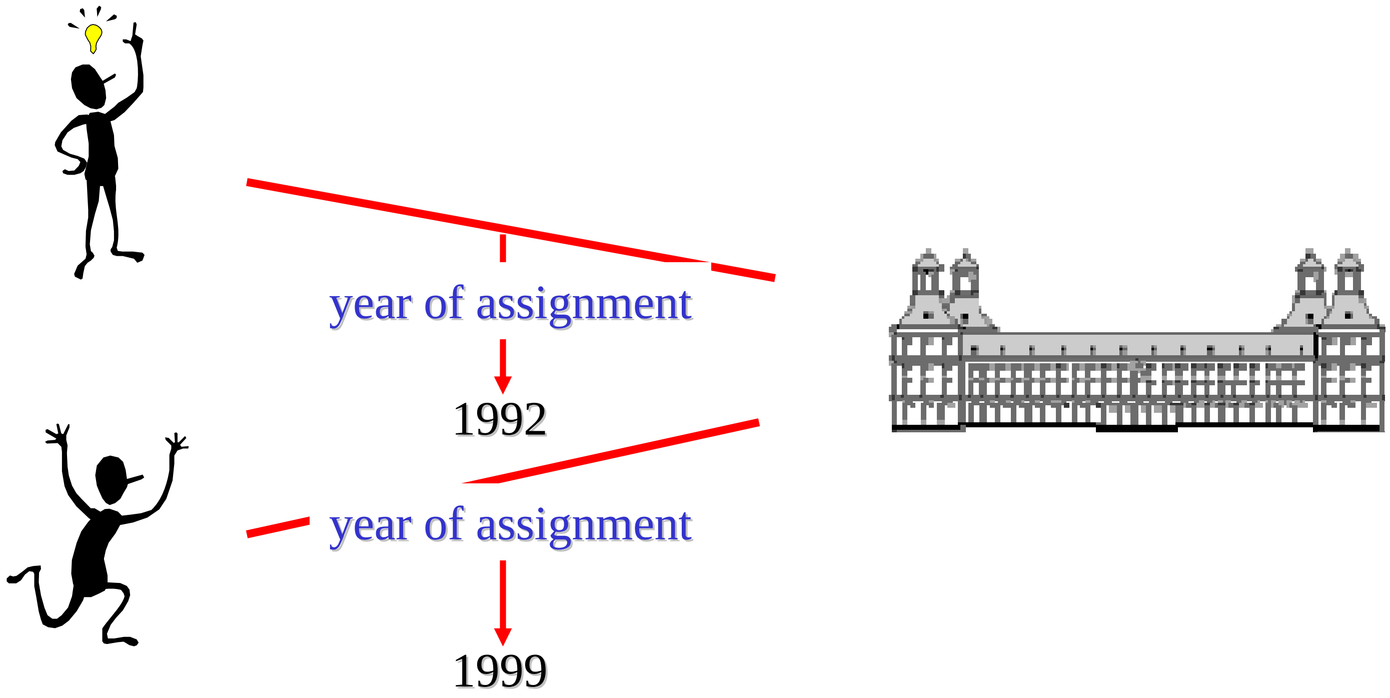
- 2nd main concept of the ER-model: elementary relationships between two or more entities (possibly with their own attribute values)




- Each relationship is uniquely characterized by the key values of the participating entities and by the values of all relationship attributes.

- However, there are no separate key attributes for relationships, as the keys of the participating entities always suffice for unique identification.

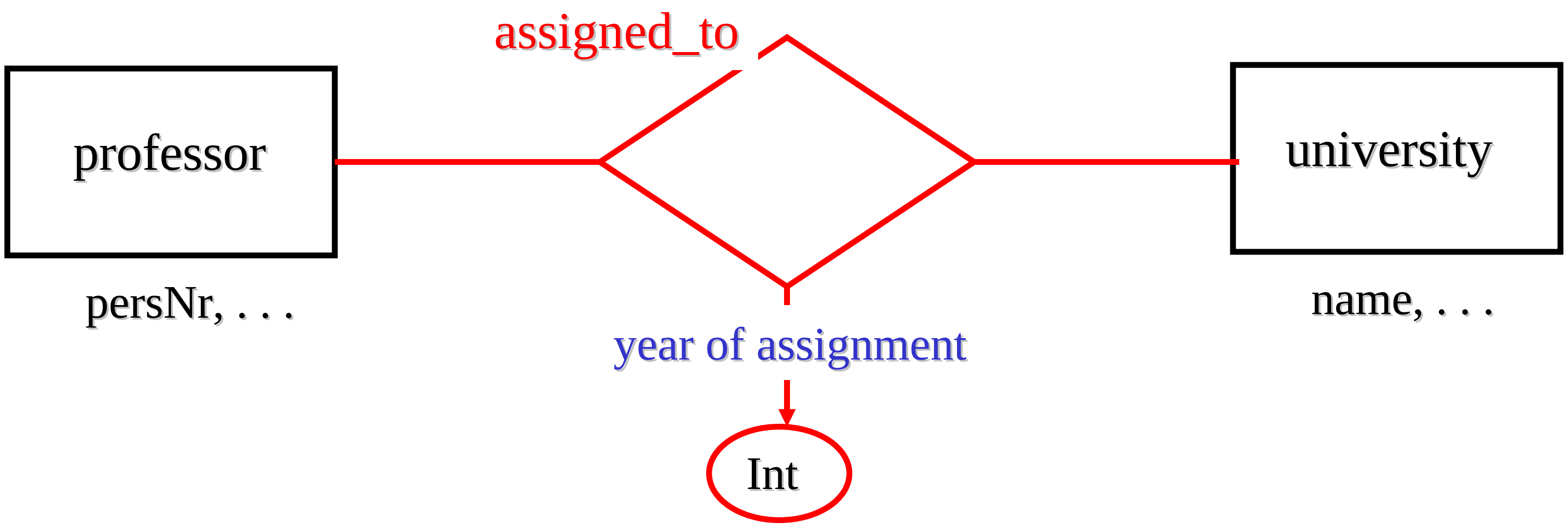
Entitites may participate in various relationships (also similar ones).

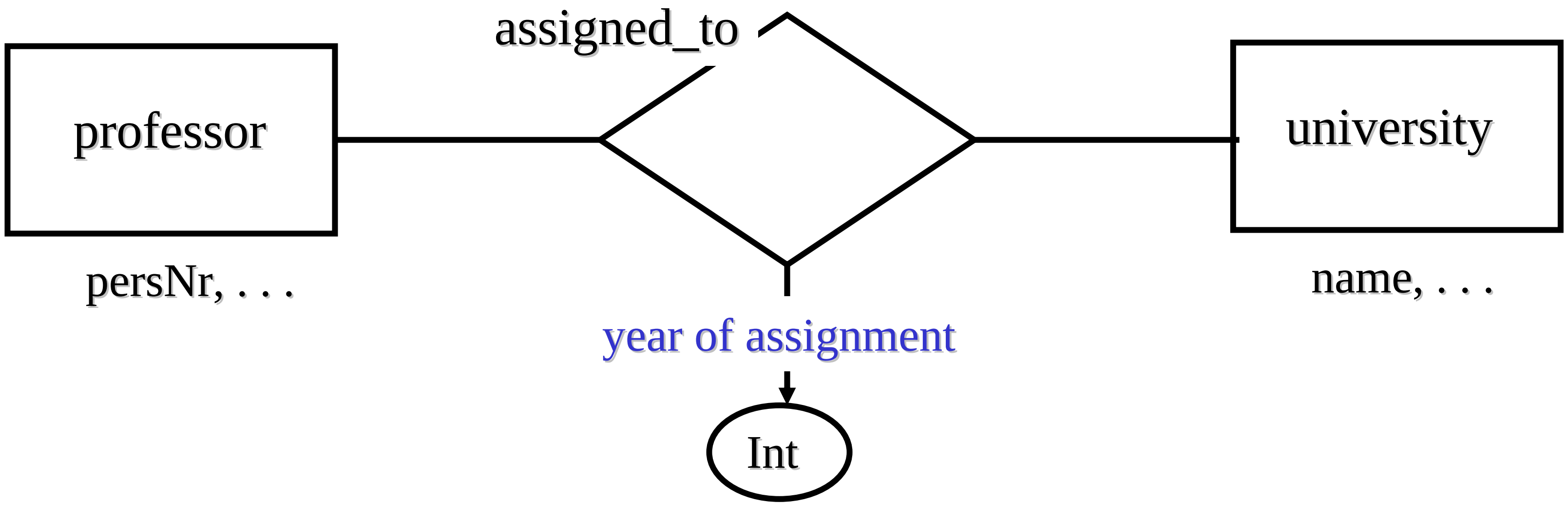


- Similar relationships may be grouped into relationship types.

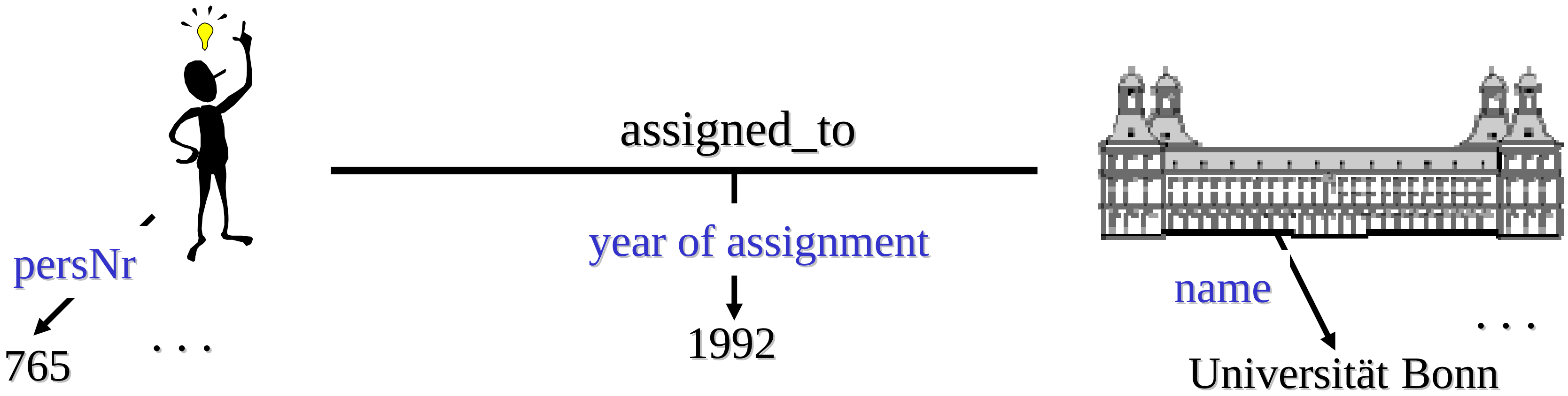
- „Similarity" requires at least identical attribute structure and identical types (and number) of participating entities.

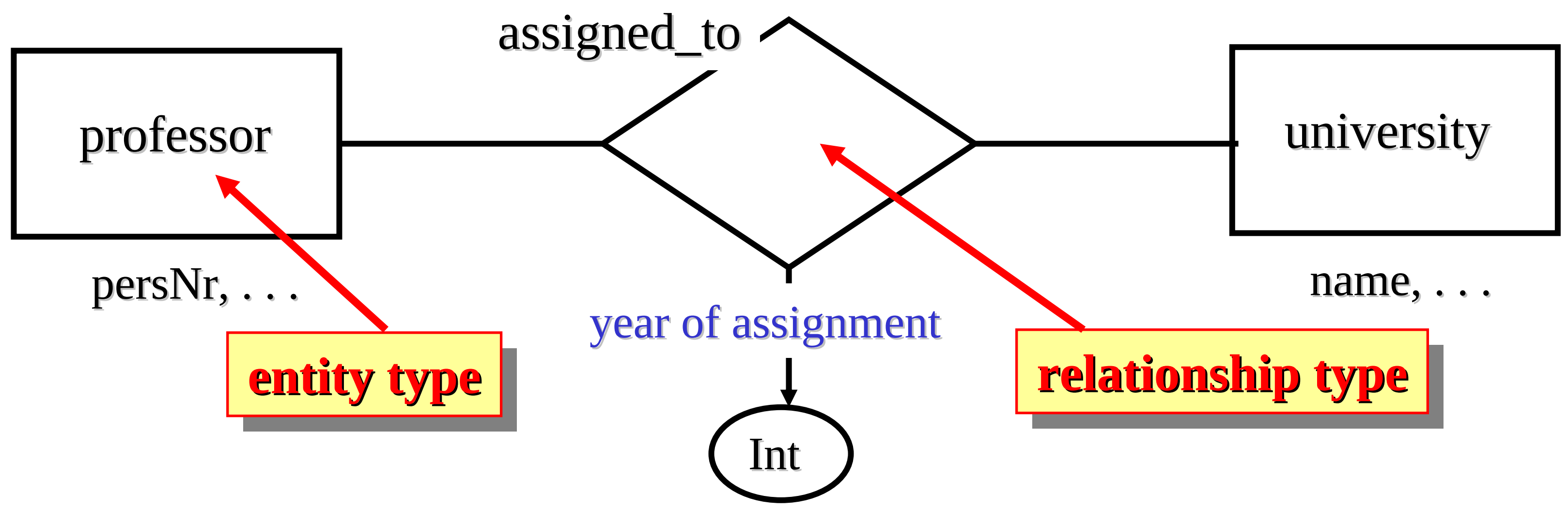
- Relationship types are graphically denoted in the ER-model by a diamond. Attributes are written as for entity types.



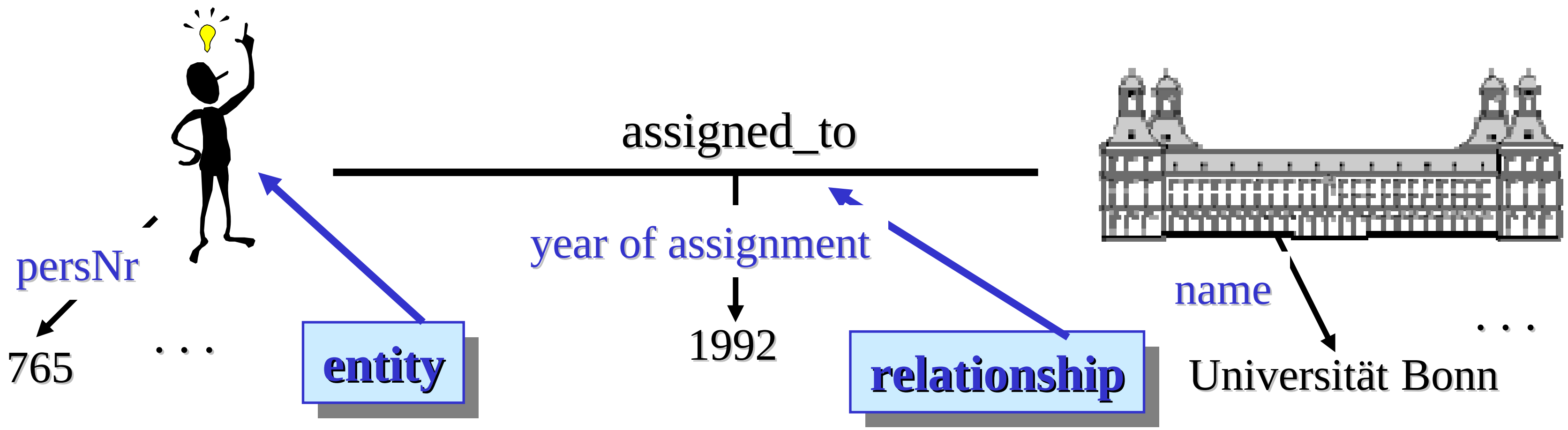



Relationship types have instances, too: Individual relationships between individual entities are analogously considered instances of the corresponding R-type.

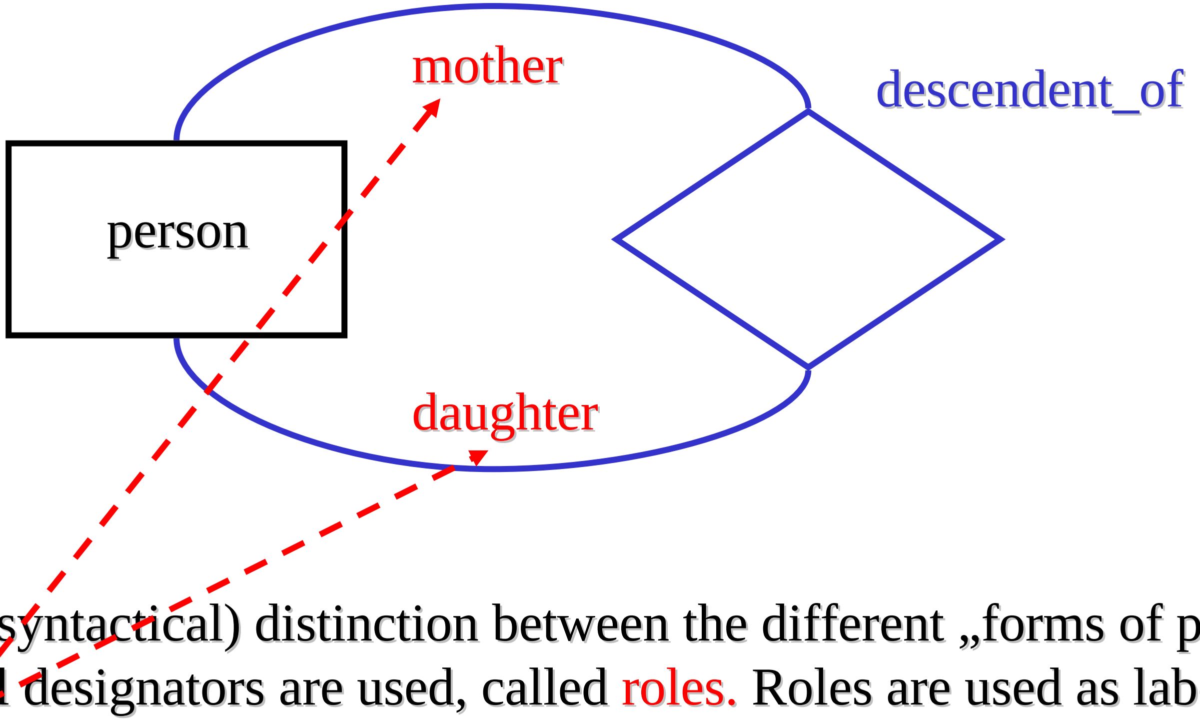




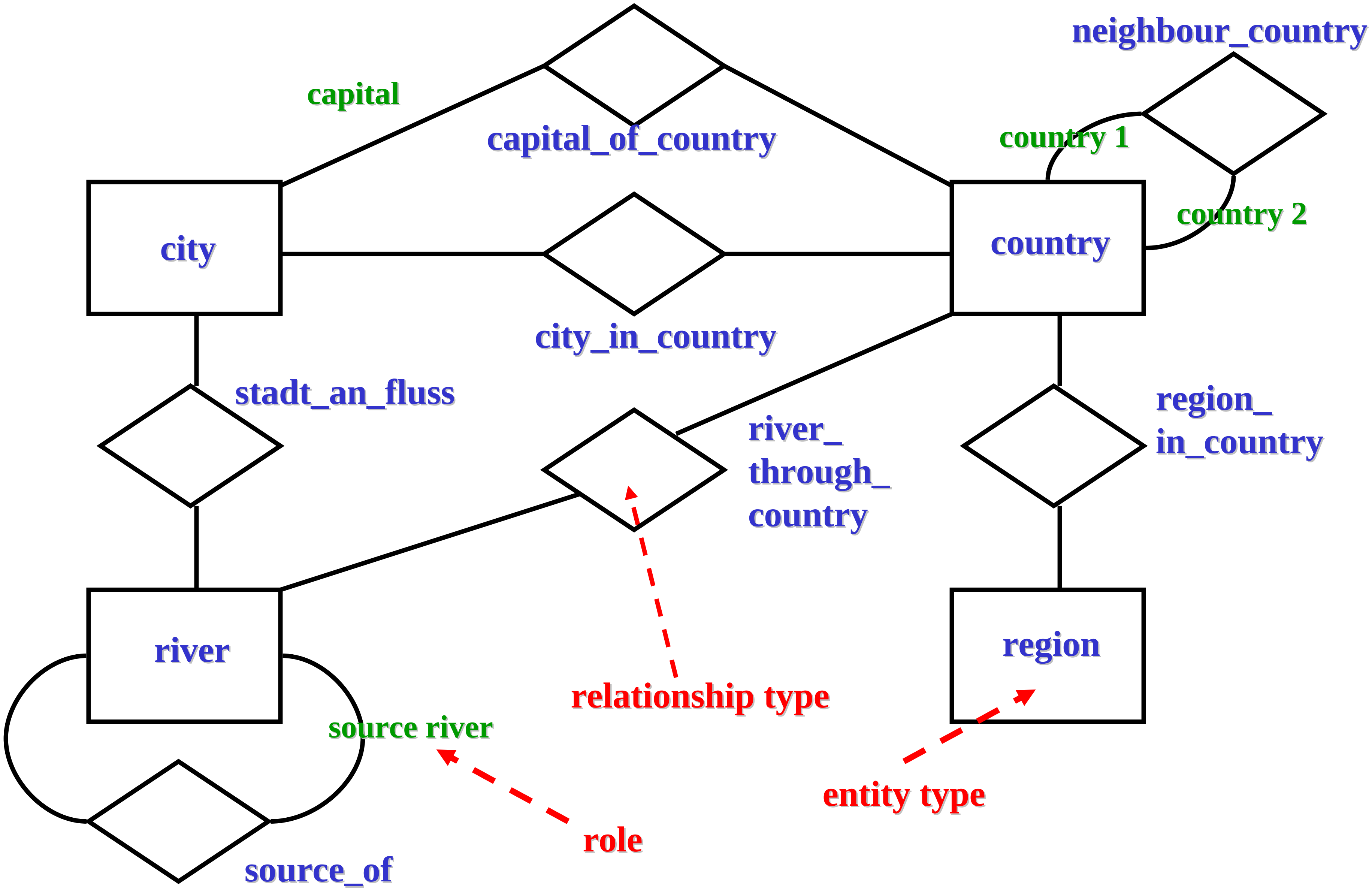

The distinction between types and instances is difficult even for specialists:
Try to be precise from the very beginning!



- One and the same entity type may participate in a relationship type more than once, e.g.:

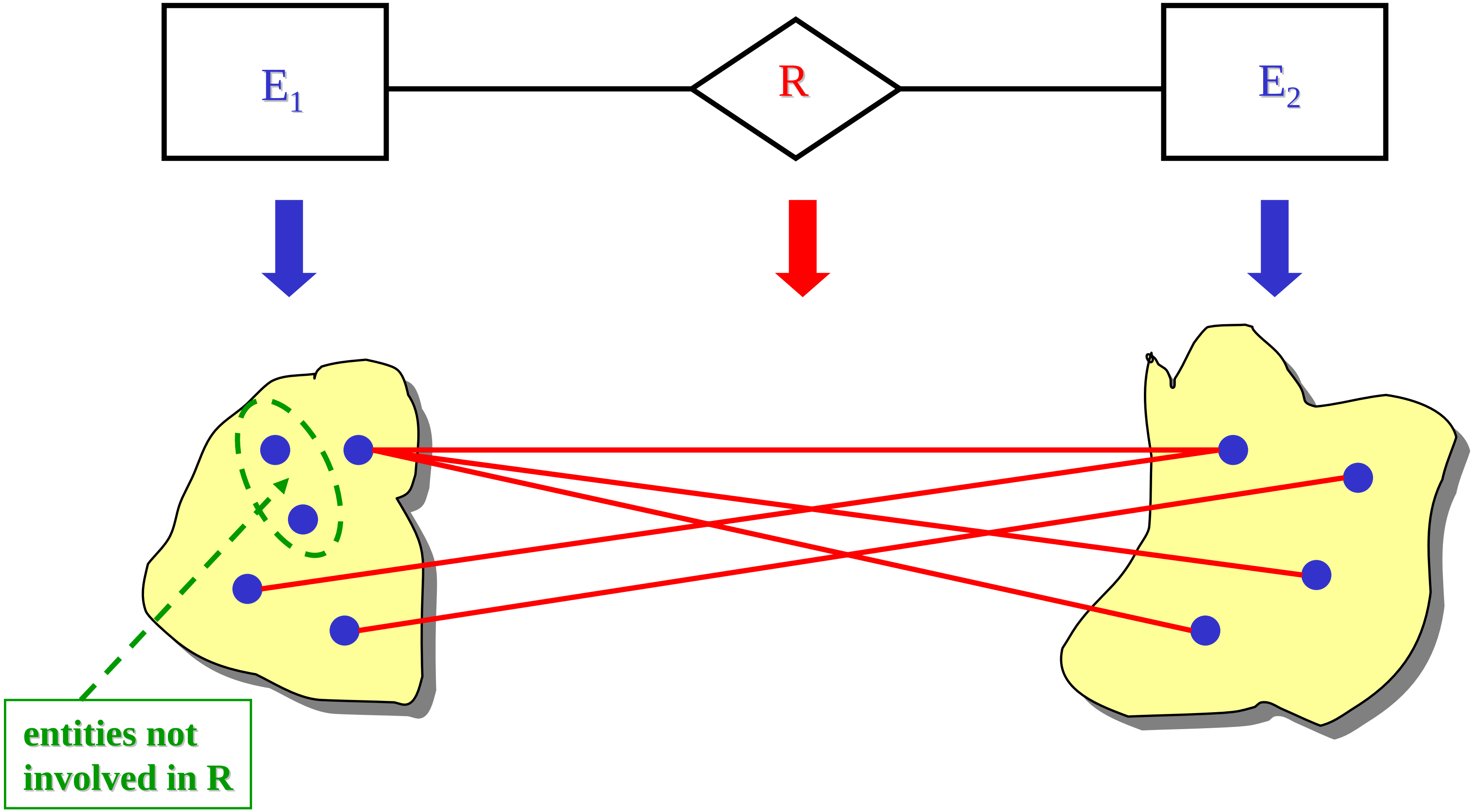



- For a (syntactical) distinction between the different „forms of participation“, special designators are used, called roles. Roles are used as labels of the lines connecting the resp. entity and relationship type.

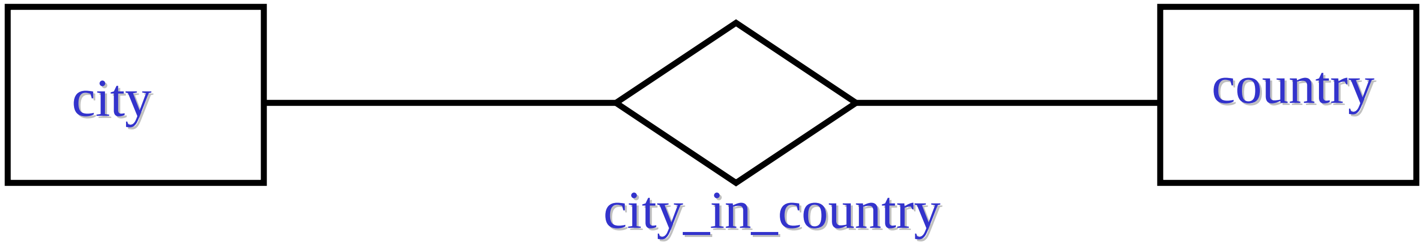


neighbour_country
capital
capital_of_country
country 1
country 2
city
country
city_in_country
stadt_an_fluss
river_
through_
country
region_
in_country
river
region
relationship type
source river
entity type
role
source_of

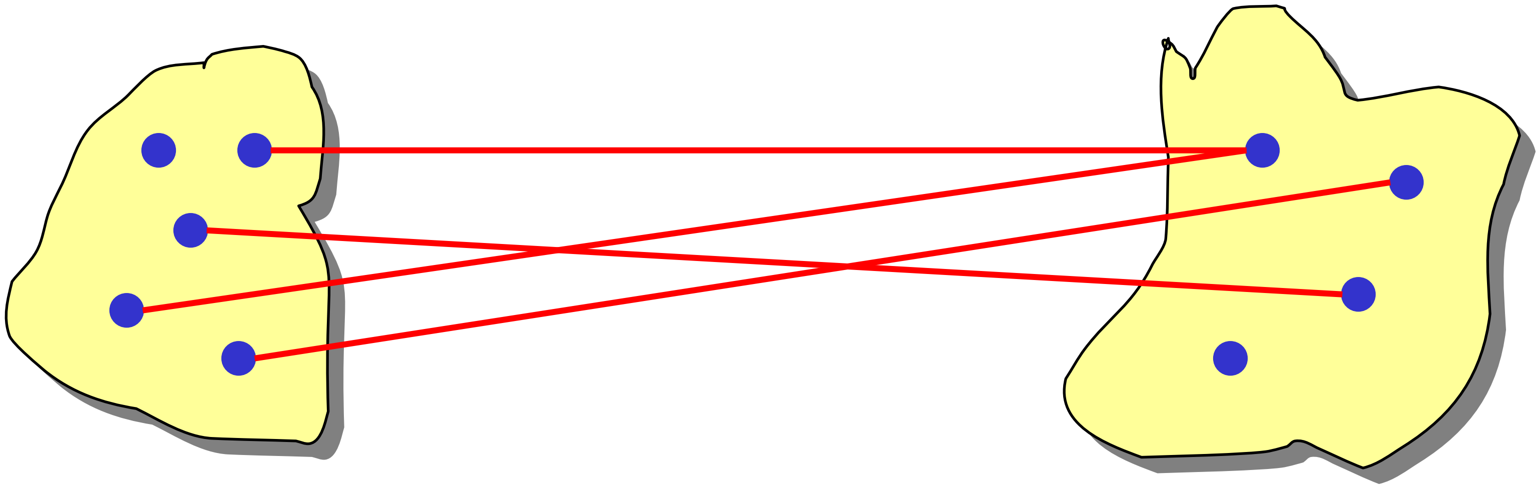
On the level of instances, a relation over the populations of the entity types involved is associated with each relationship type.



In many cases, at most one entity on one „side" of a relationship type may be associated with a particular entity on the other „side“ of the relationship, e.g.:



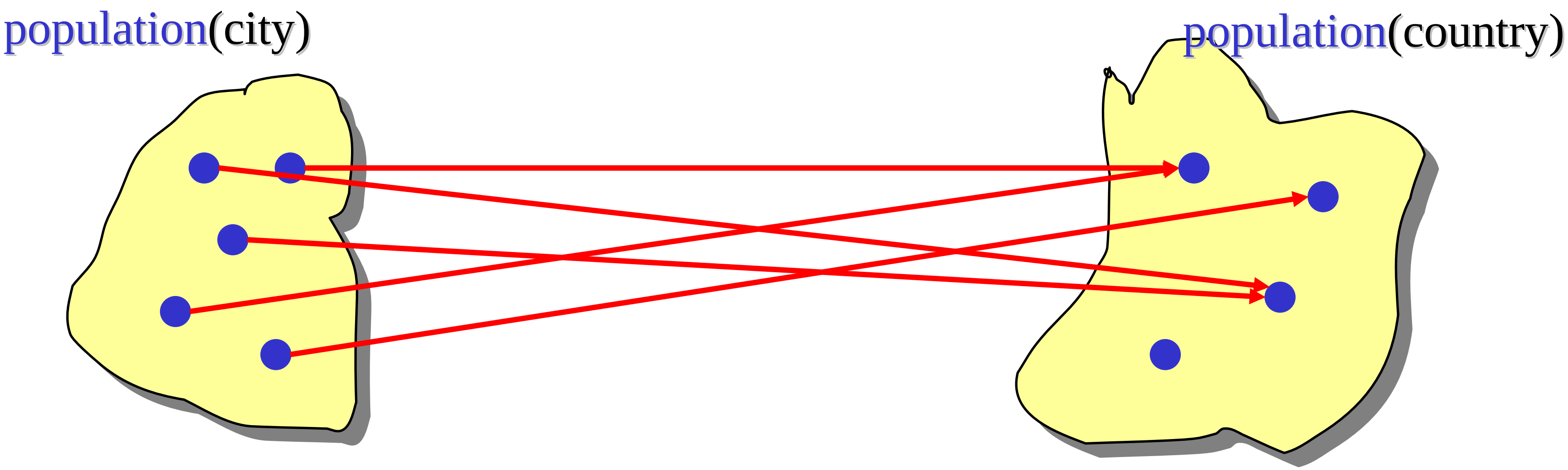

- Each city is in exactly one country.
- In each country, however, arbitrarily many cities may be situated.



- Mathematically, city_in_country is a function:

$$\boxed{\text{city\_in\_country:} \quad \text{city} \rightarrow \text{country}}$$

- More exactly: . . . a function from the population of city into the population of country.



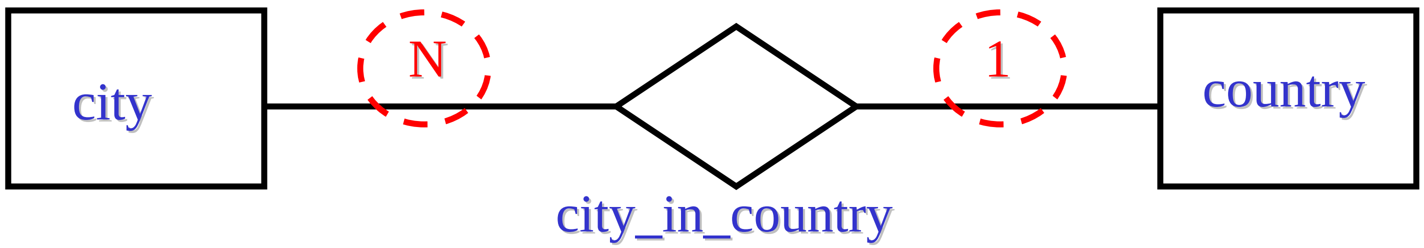
- In the ER-model, such restrictions of the admissible combinations can be expressed by means of so-called functionalities, annotations attached to the edges connecting entity and relationship types.

- There are four different kinds of binary relationships expressible by means of functionalities:

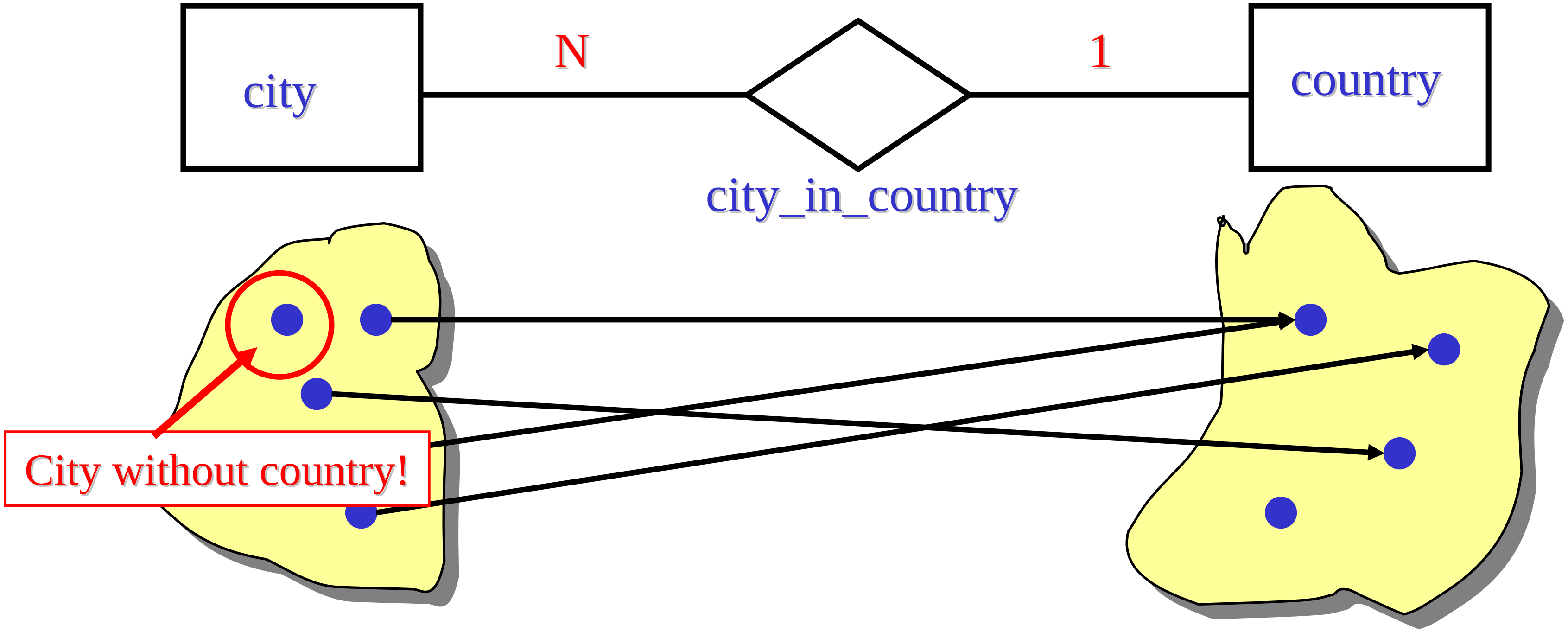
| | | | |
|---|---|---|---|
| 1 : 1 | 1 : N | N : 1 | N : M |

- In this context, N resp. M stands for arbitrary integer values ≥ 0.

- In the ‚city_in_country'-example, an N:1-relationship is appropriate: There is exactly one country per city, but arbitrarily many citites per country (N ≥ 0).

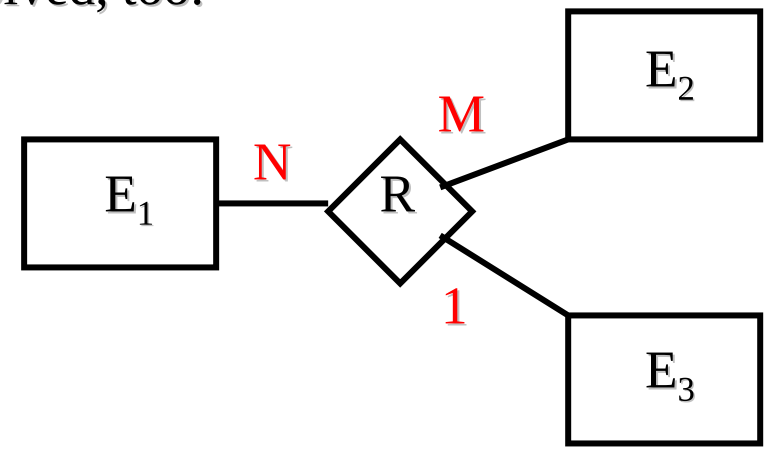


- Functionalities of type 1 : 1, 1 : N or N : 1 define partial functions where some of the instances of the types involved possibly are not related at all.



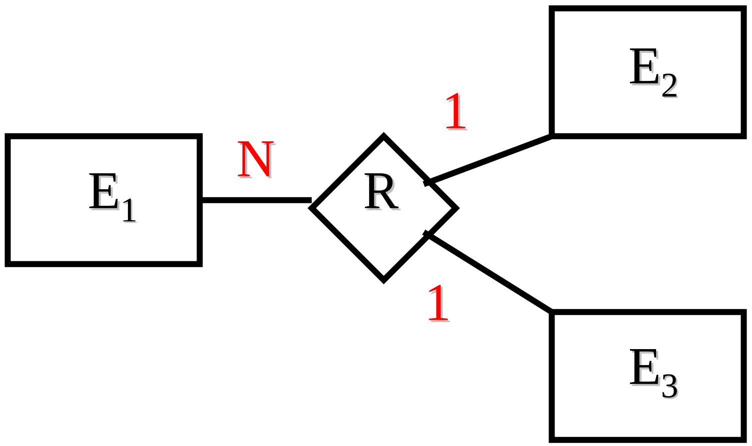

- In the „normal“ ER-model, total functions cannot be distinguished from partial ones – in extensions of the model there are additional graphical means for explicitly stating whether a function is partial or total.

- In a 1 : 1-relationship each instance of one of the entity types involved is related to none or exactly one of the instances of the other entity type.
- An N : M-relationship can be considered the „normal case" without restrictions on the number of participating entities.
- If no functionalities have been stated for a relationship type, then an implicit N : M functionality is assumed.
- Functionalities can be defined for relationships with more than two entities in-volved, too:




$R: E_1 \times E_2 \rightarrow E_3$

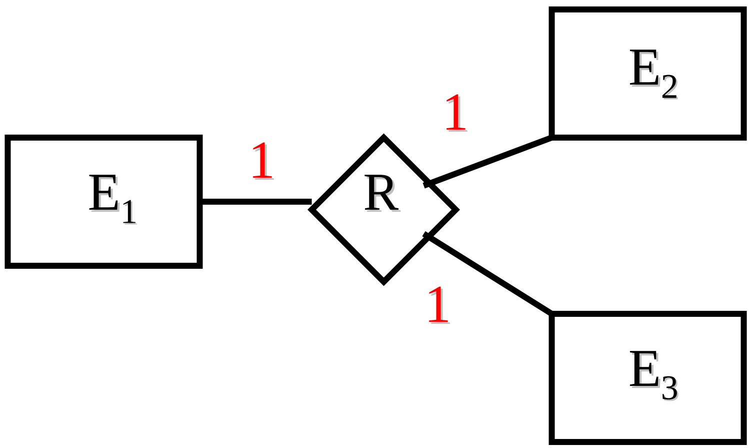
- If in an n-ary relationship several edges are marked by '1', then the resp. relationship type represents several partial functions:




$R^{(1)}$: $E_1 \times E_2 \rightarrow E_3$

$R^{(2)}$: $E_1 \times E_3 \rightarrow E_2$

- . . . and analogously:




$R^{(1)}$: $E_1 \times E_2 \rightarrow E_3$

$R^{(2)}$: $E_1 \times E_3 \rightarrow E_2$

$R^{(3)}$: $E_2 \times E_3 \rightarrow E_1$

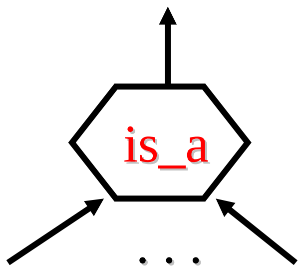
- The concepts introduced so far have been contained in Chen‘s original proposal throughout. Since then, however, various extensions have been proposed:
  Extended Entity-Relationship Model (EER-model)

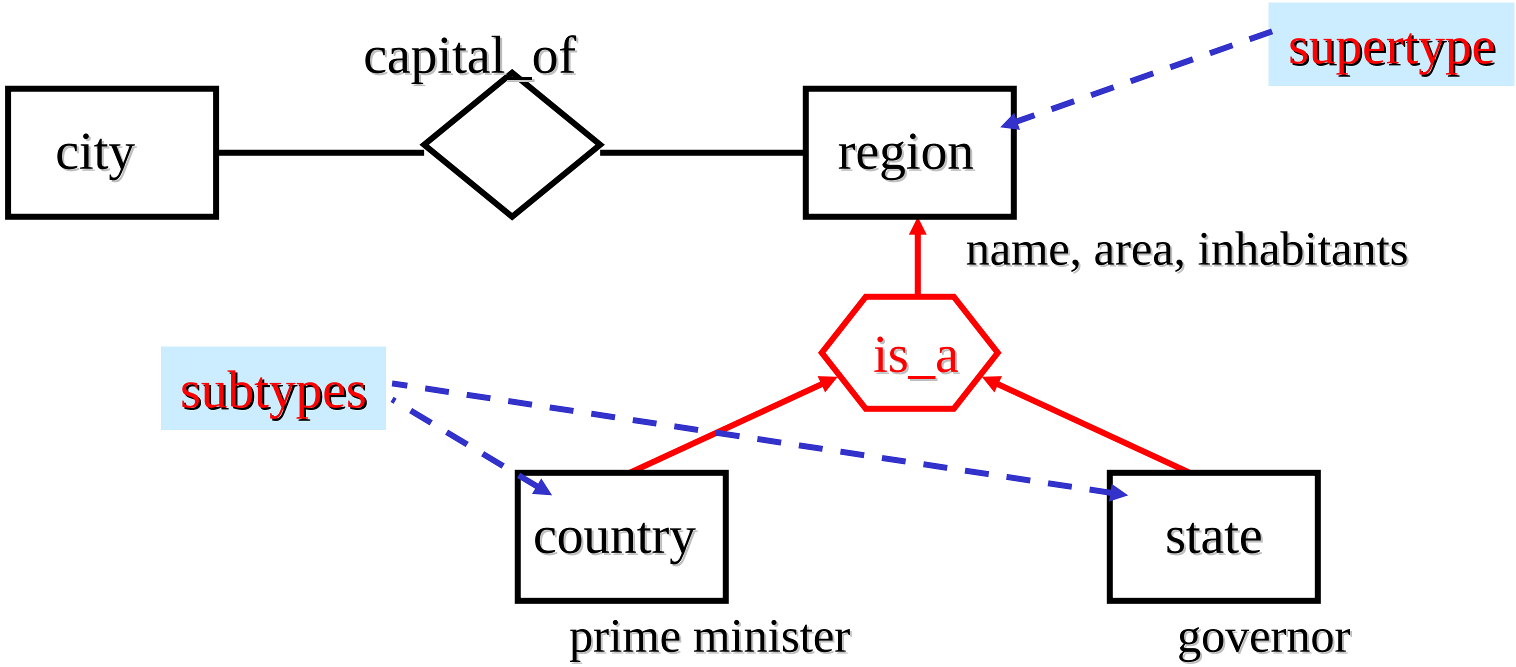
- Most important extensions (as in object-oriented models):

  Generalization

- This means:
  - formation of subtypes of entity types
  - sub-/supertype relationships (type hierarchy)
  - inheritance of attributes and of „participations" in R-types

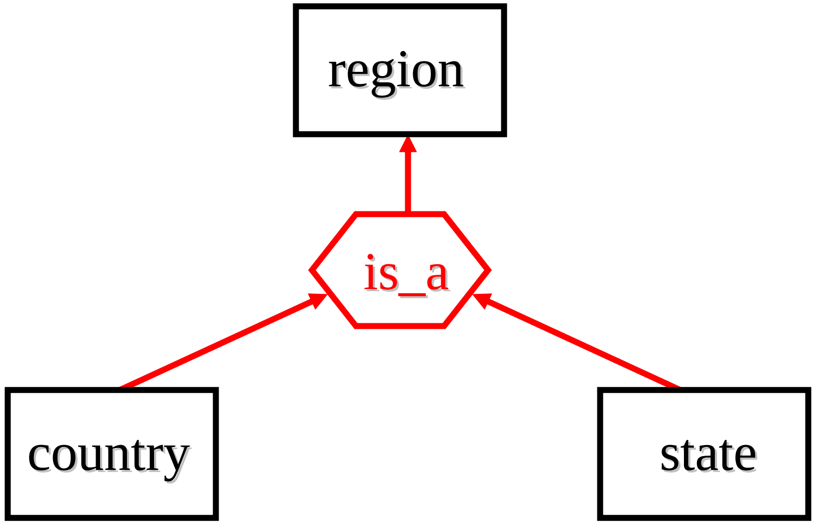
- Special graphical notation for generalization of E-types:



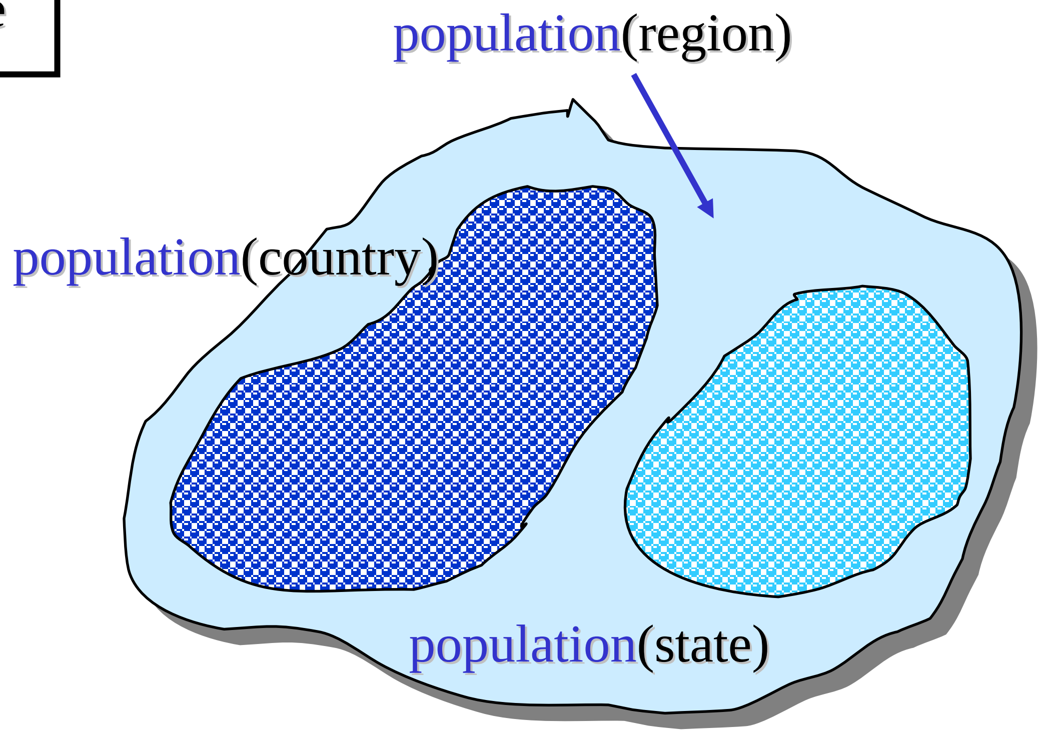



„Inheritance" in this example means:

- Both subtypes inherit all attributes of the supertype, i.e., they „own" these attributes without explicit definition.
- Both subtypes participate in the relationship type capital_of',  which has been explicitly defined for the supertype only.



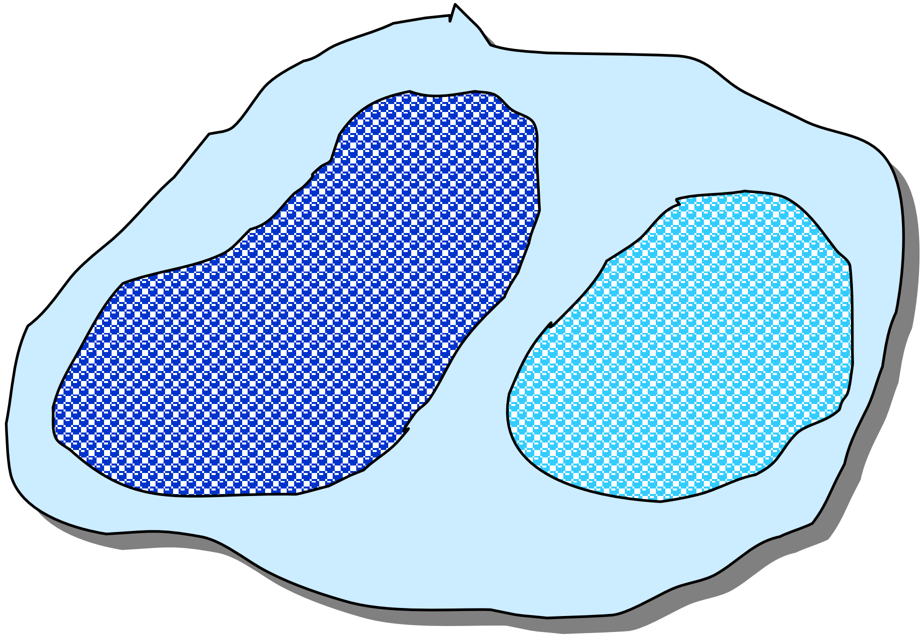

Generalization always means that the populations of the subtypes are subsets of the population of the supertype.



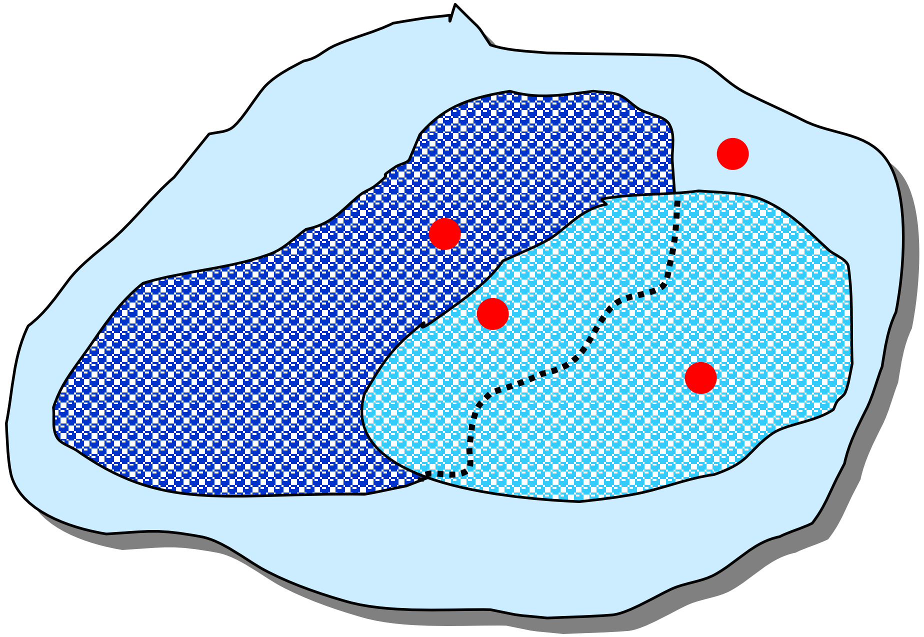

This circumstance motivates the notion 'is_a'-relationship:

"Every country is a region."

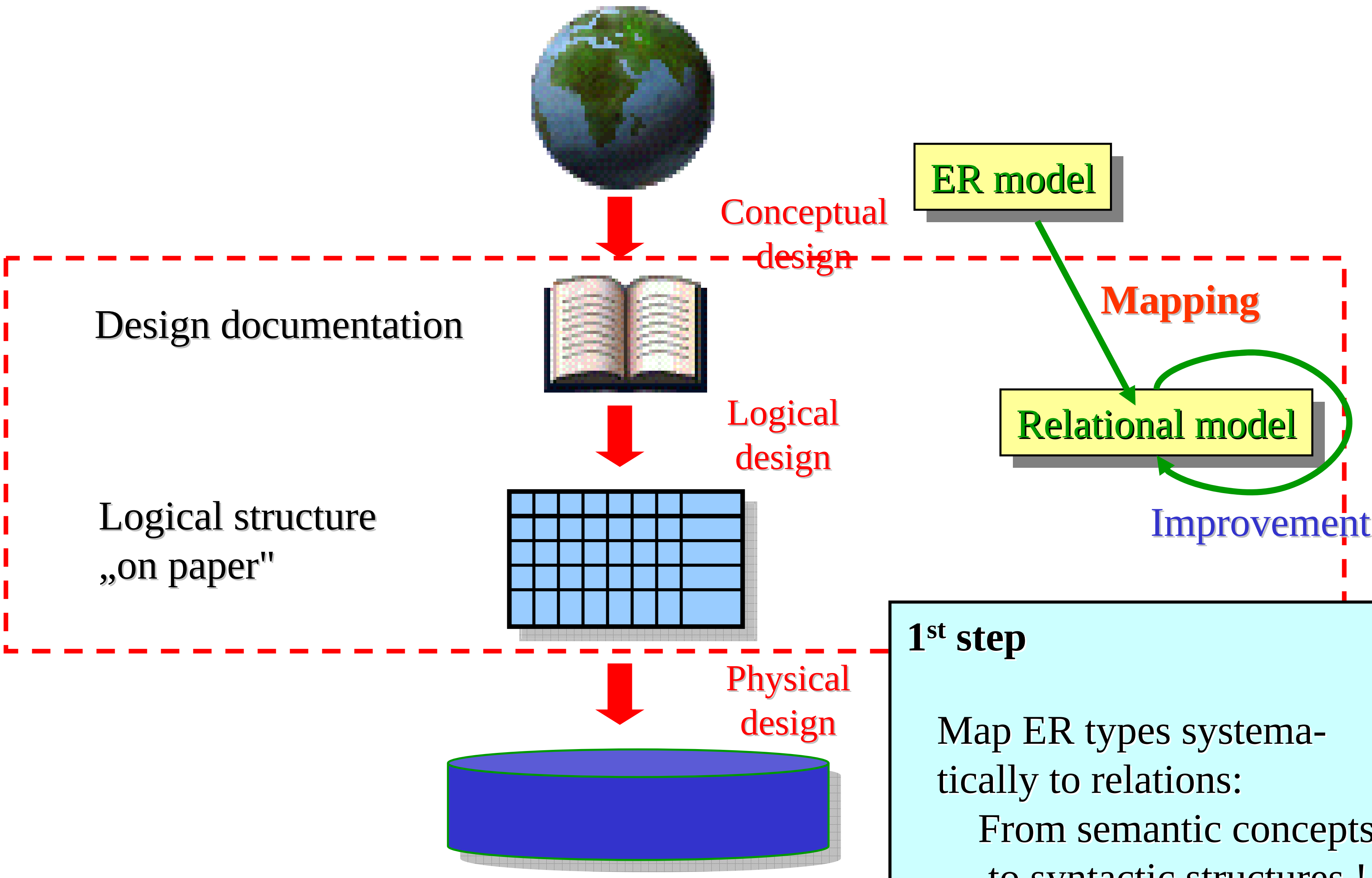
Quantifier over instances!



In the example: special case
disjoint generalization
(Empty intersection of the populations)



In general:
This form of the 'is_a'-notation just means some form of subset formation, i.e. overlapping, incomplete subdivision.

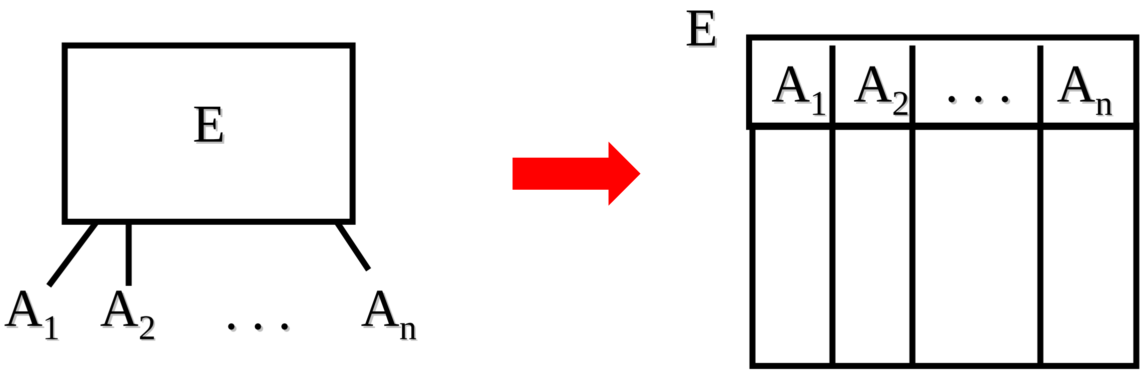



**1st step**

Map ER types systematically to relations:
From semantic concepts to syntactic structures !

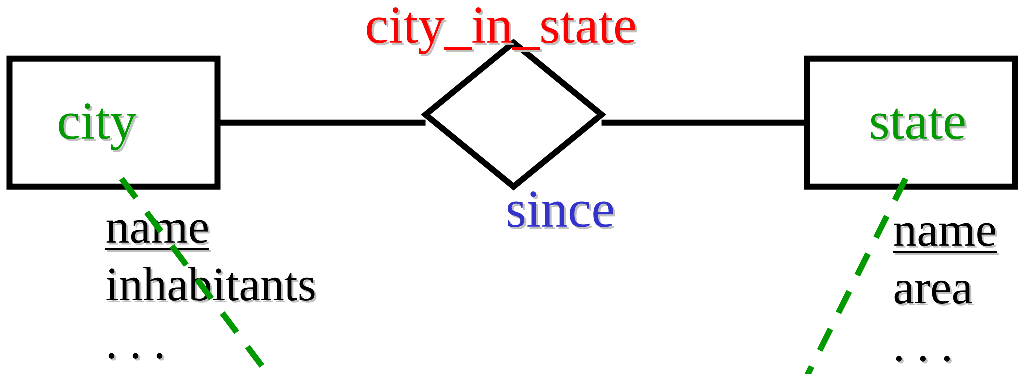
- Mapping from ER model to relational model:
  - in principle very easy: per type one relation (table)
  - in detail and for extensions: quite difficult

- Mapping of entity types: rather obvious
  - type ⇒ table name
  - attribute ⇒ column name




- Key attributes are mapped to primary key columns.

- For relationship types:
  - Use the same basic idea: one relation per type.
  - but: How are participating entity types represented?



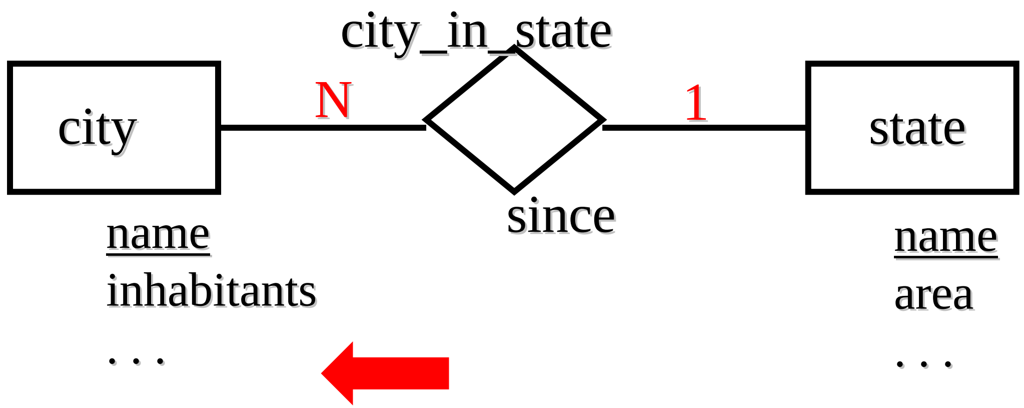

- Obvious solution as well: The participating entities are represented by means of the values of their primary key attributes (possibly after renaming).

city_in_state

| city.name | state.name | since |
|---|---|---|
| | | |

Foreign key

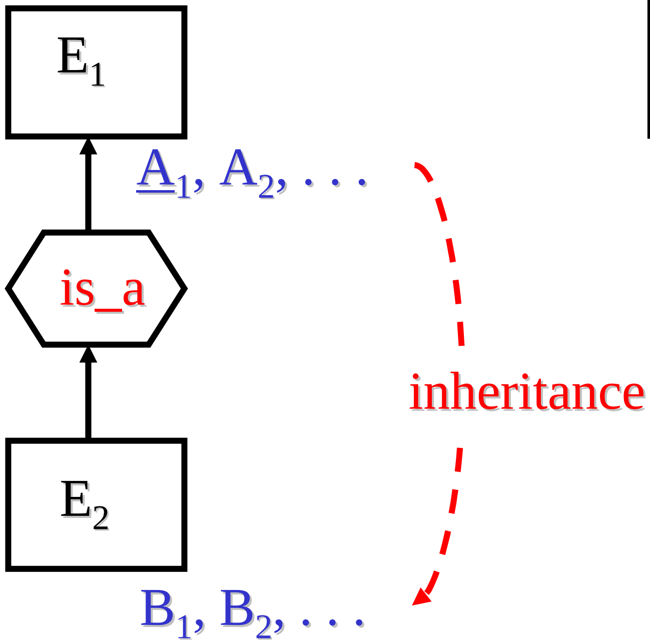
In presence of special functionalities (1 : N, N : 1, 1 : 1 resp.), a separate relationship table is not necessary, as the relationship information can be embedded into the table of the entity type on the N-side:




Foreign key

city

| city.name | inhabitants | . . . | state.name | since |
|---|---|---|---|---|
| | | | | |



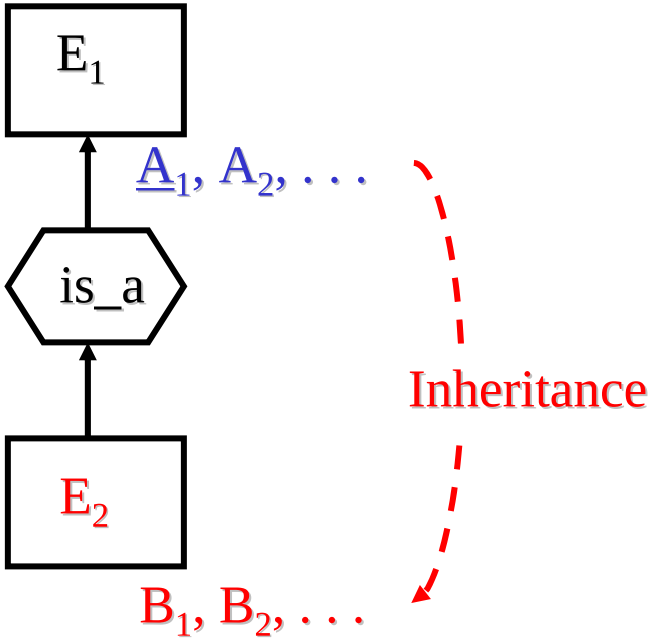

**How to realize inheritance and sub-/super type relationships relationally ?**

- Relational model:
  does not know any inheritance!
- Inheritance thus has to be "simulated" .

Relational representation of a super type $E_1$ is obvious:

$E_1$

| $\underline{A_1}$ | $A_2$ | . . . |
|---|---|---|
| | | |




$E_1$

| $\underline{A_1}$ | $A_2$ | . . . |
|---|---|---|
| $a_{11}$ | $a_{12}$ | . . . |
| $a_{21}$ | $a_{22}$ | . . . |

$E_2$

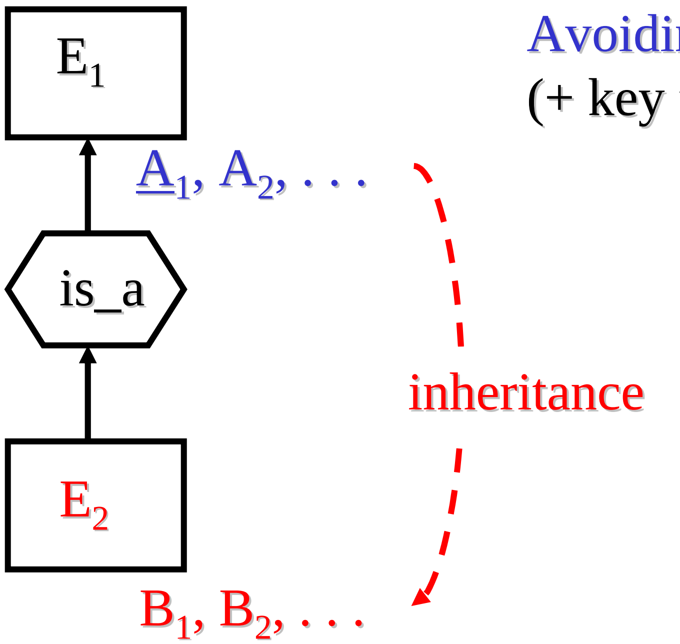
| inherited | | | „native" attributes | | |
|---|---|---|---|---|---|
| $\underline{A_1}$ | $A_2$ | . . . | $B_1$ | $B_2$ | . . |
| $a_{11}$ | $a_{12}$ | . . . | $b_{11}$ | $b_{12}$ | . . . |

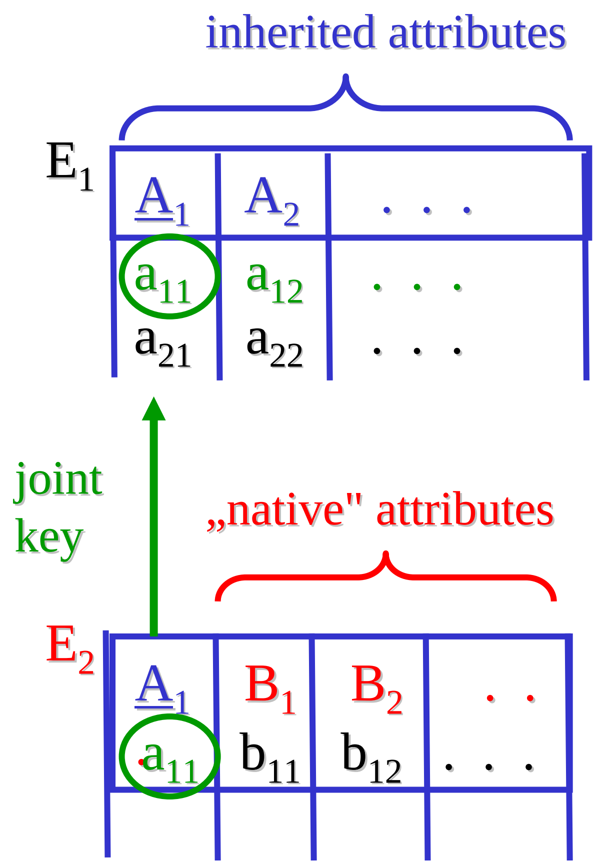
- Obvious relational realization of a subtype:
  subtype relation $E_2$ owns „native" and inherited attributes.

- <u>But</u>: Values of the inherited attributes of all $E_2$-instances have to be (redundantly) repeated in the $E_1$-relation in this case!

- <u>Reason</u>: Each $E_2$ -instance is an $E_1$-instance, too !

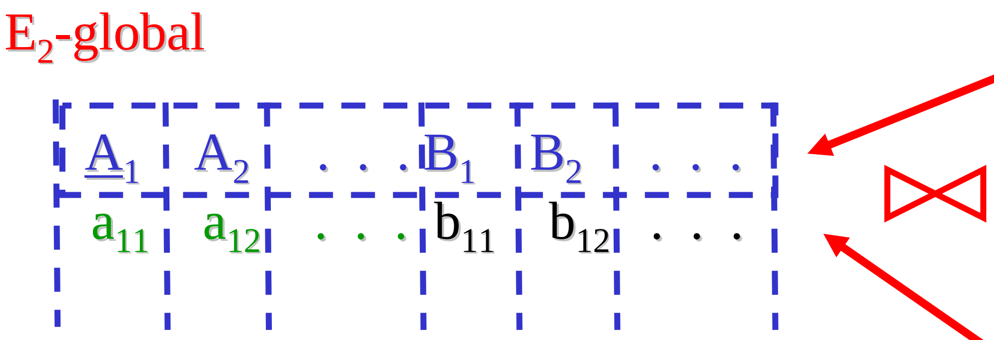
Avoiding duplication: Store only „native" attributes (+ key for joining) in the subtype relation!

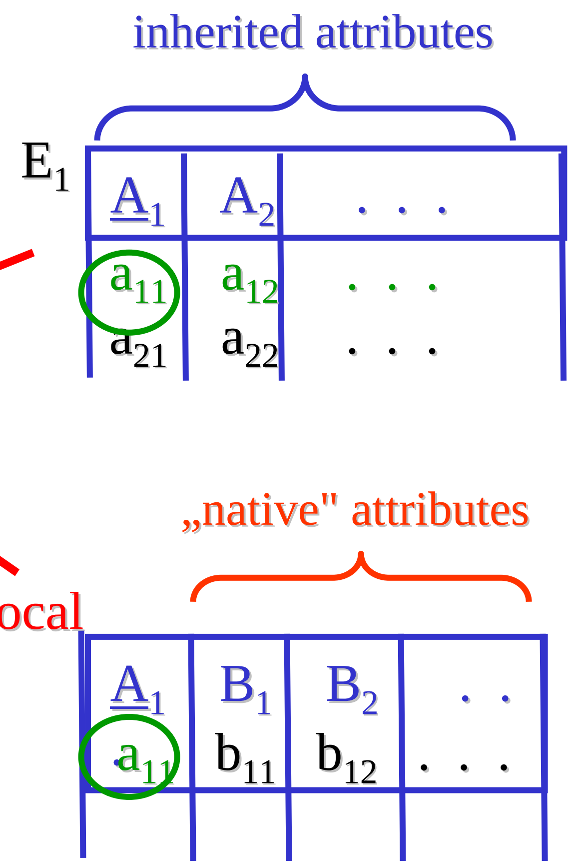



But in this case relation $E_2$ does no longer contain all attributes of $E_1$-entities !

way out: E2-population is completely realized by means of a view joining the inherited and the native attributes.

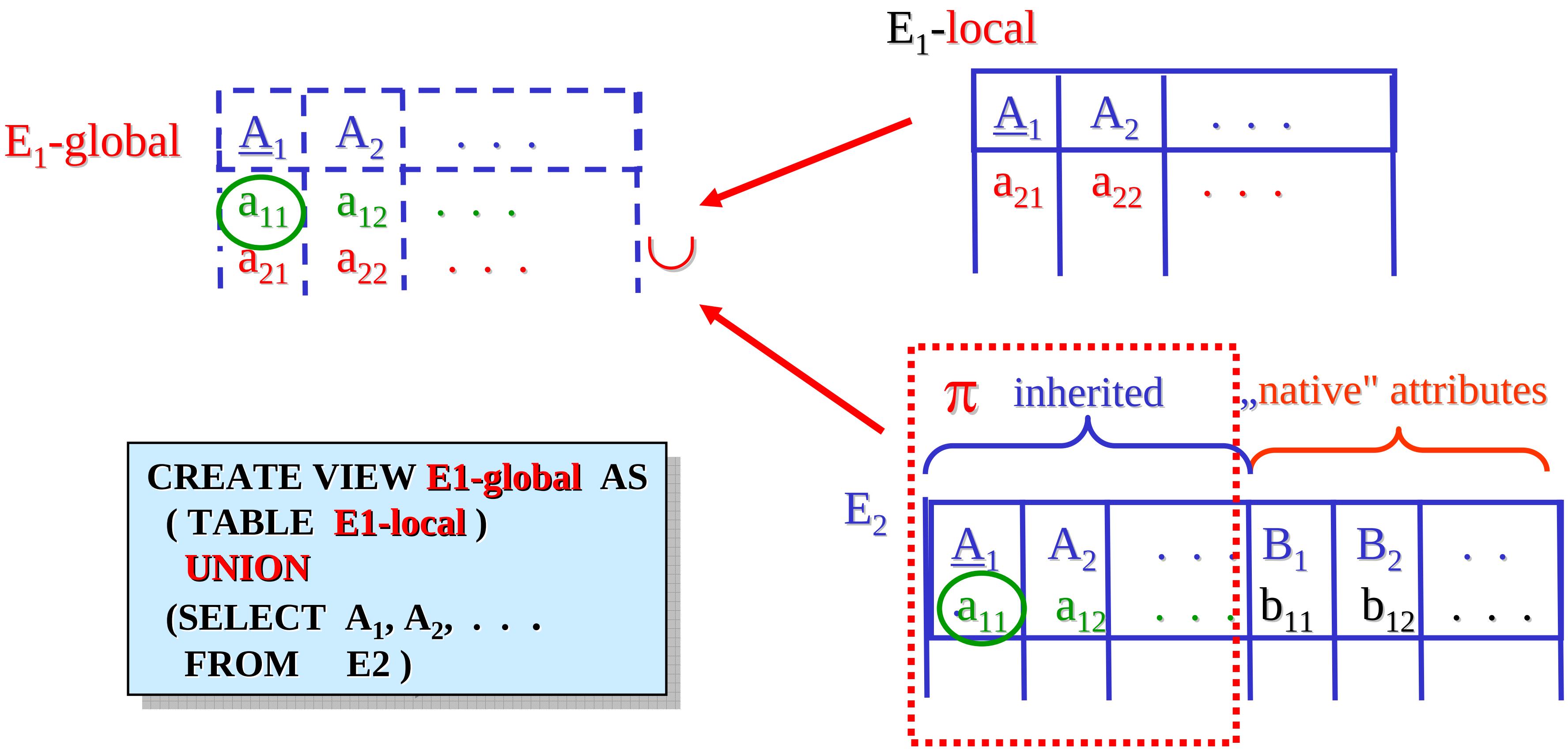



```
CREATE VIEW E2-global AS
  (SELECT  *
   FROM  E1  INNER JOIN  E2-local
                         ON  E1.A1 = E2-local.A1)
```

3rd alternative (also free of redundancies and using a view):

Distribute values of the inherited attributes to different relations – super types are reconstructed via views.




```
CREATE VIEW E1-global AS
 ( TABLE E1-local )
  UNION
 (SELECT A1, A2, . . .
  FROM   E2 )
```

**Which of the three alternatives is „the best" ?**

1. Relations $E_1$ and $E_2$, no views:

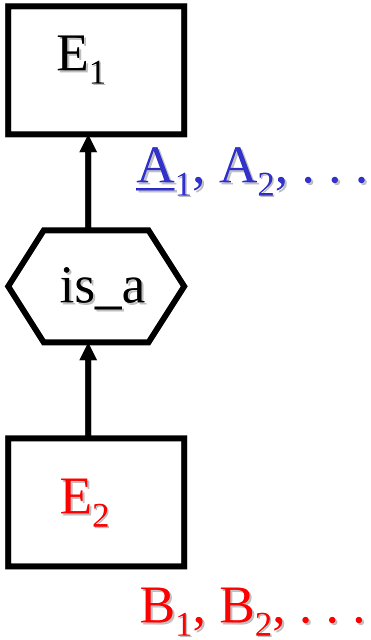
   + short access time (without any join of tables)
   – high requirements for space (due to redundant storage)

2. Relations $E_1$ and $E_2$-local, view $E_2$-global (JOIN):

   + Only key attribute values are stored redundantly.
   – Access to $E_2$-attributes is slower (due to join).

3. Relations $E_2$ and $E_1$-local, view $E_1$-global (PROJECT-UNION):

   + No duplication of any attribute values.
   – Access to $E_1$-attributes is slower (due to projection and union) .




**What happens if an $E_2$-entity is deleted ?**

- relational variant 1 (inherited attributes duplicated):
  Deletion from both relations is necessary.
- relational variant 2 (inherited and native A. separated):
  Deletion from both relations is necessary.
- relational variant 3 ($E_2$-attributes only in one relation):
  no propagation of deletions required

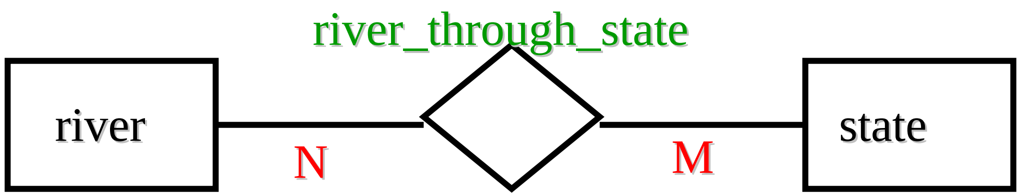
⇒ In variants 1 and 2: referential integrity constraints with delete cascade is required.

- For insertions and modifications: Changes in several relations may be necessary, too (depending on the chosen strategy).

- Deletion of instances of the super type $E_1$: Cascading deletion if the resp. instance is an $E_2$-instance, too (again referential integrity).

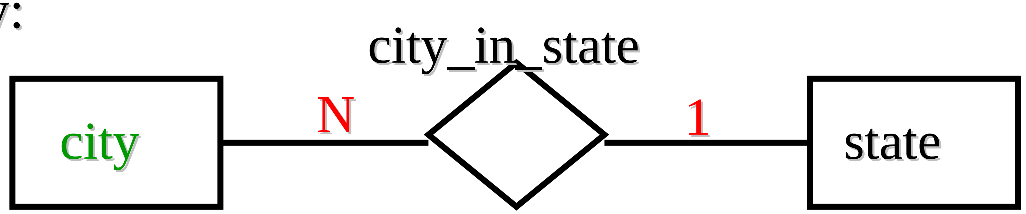
Each relationship type induces FOREIGN KEY-constraints as well:

- With N : M-functionality:




```
CREATE TABLE river_through_state
( River String REFERENCES river,
  State String REFERENCES state )
```

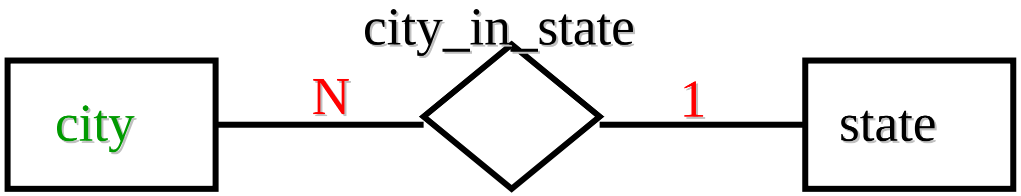
- With N : 1-functionality:




```
CREATE TABLE city
( . . . ,
  State String REFERENCES state )
```

- <u>But</u>: Uniqueness of the state in the city_in_state-relationship has not yet been expressed !

- An additional CHECK-constraint is required contraining the number of state instances:




```
CREATE TABLE  city
( . . . ,
 CHECK  COUNT (SELECT    State
               FROM      city S
               WHERE     S.Name = Name) =<  1
```

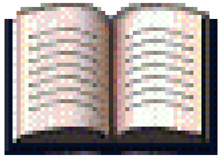
**Implicit universal quantifier ranging over each city row !**

Conceptual design

ER model

Mapping

Design documentation

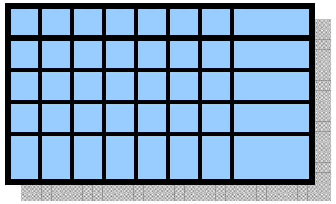


Logical design

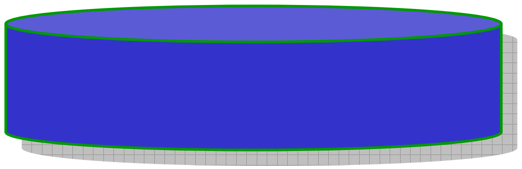
Relational model

Improvement

Logical structure
„on paper"



Physical design



**2nd step**
(after reaching relations):

Improve design by „fine tuning“ the structure of tables

- Example of „bad design“ (remaining after mapping from ER level):
  ‚City_in_state‘ and ‚capital_of‘ have been placed into a single table.

| City | State | Capital |
|---|---|---|
| Bonn | NW | Düsseldorf |
| Köln | NW | Düsseldorf |
| Essen | NW | Düsseldorf |
| . . . | | |
| Mainz | RP | Mainz |
| Trier | RP | Mainz |
| . . . | | |

Redundantly stored information:
Düsseldorf is the capital
of North Rhine-Westphalia.

- Obviously one topic (Which city is the capital of . . . ?) has been combined with another topic (In which state is a certain city situated ?) in such an „unlucky“ manner that considerable redundancies occur, resulting in waste of space.

What does „a topic“ mean ?

- An immediate consequence of such cases of storing multiple topics in one table is the occurrence of so-called anomalies when updating such tables:

Assume Köln (as largest city in NW) replaces Düsseldorf as capital:

**One fact changes, but multiple updates have to be made.**

| City | State | Capital |
|---|---|---|
| Bonn | NW | Düsseldorf |
| Köln | NW | Düsseldorf |
| Essen | NW | Düsseldorf |
| . . . | | |
| Mainz | RP | Mainz |
| Trier | RP | Mainz |
| . . . | | |

- Analoguos anomalies may happen due to insertions and deletions:
  - An instance of topic 1 disappears, as soon as it is no longer associated with any instance of topic 2.
  - A new instance of topic 1 can only be inserted, if it is combined with an instance of topic 2 (or null values are used).

**How to prevent such „defects“ (anomalies, redundancies) ?**

In the example, there is a simple remedy:

Separate the two topics into different relations!

| City | State | Capital |
|---|---|---|
| Bonn | NW | Düsseldorf |
| Köln | NW | Düsseldorf |
| Essen | NW | Düsseldorf |
| . . . | | |
| Mainz | RP | Mainz |
| Trier | RP | Mainz |
| . . . | | |

Decomposition

| City | State |
|---|---|
| Bonn | NW |
| Köln | NW |
| Neuss | NW |
| . . . | |

| State | Capital |
|---|---|
| NW | Düsseldorf |
| RP | Mainz |
| . . . | |

No redundancy, no anomaly !

- Already discovered by Codd before 1970: Functional relationships between attributes are of help for finding meaningful decompositions and for avoiding reduncancies!

- Resulting from this observation, Codd developed an elaborate theory of relational normal forms.

- Prerequisite: Designers identify such functional relationships during schema design (quite similar to identifying functionalities in the ER model) and express them as special integrity constraints.

**Functional dependency** (short: FD)

- Principle of functional dependency:
  - Let A and B be attributes of a relation R.
  - B depends functionally on A, if in each state of R each A-value always occurs in combination with the same, uniquely determined B-value.
  - symbolic notation: $A \rightarrow B$

- Each city lies in exactly one state: City → State
- Each state has exactly one capital: State → Capital
- <u>But also</u>: If a city is a capital then it is the capital of exactly one state:

  Capital → State
- Each city is associated with exactly one capital (namely the capital of its state):

  City → Capital

| City | State | Capital |
|---|---|---|
| Bonn | NW | Düsseldorf |
| Köln | NW | Düsseldorf |
| Essen | NW | Düsseldorf |
| . . . | | |
| Mainz | RP | Mainz |
| Trier | RP | Mainz |
| . . . | | |

- Decomposition separates FDs:

| City | State |
|---|---|

| State | Capital |
|---|---|

- One FD seems to be lost, however: City → Capital

- Thesis (claim) on which the normalization theory of Codd is based:

> **Attributes connected via an FD represent semantically significant topics of the application domain.**

- That is: Every FD identifies a topic – thus separating topics means separating FDs.
- But: Not every such topic is necessarily represented by an FD.
- Moreover: FD-connection is a sufficient, but not a necessary criterion for the existence of a ‚topic'.
- Basic idea of Codd's approach to normalization of relations:
  Decompose relations in such a way that „normally" each FD has a component relation of its own. But try to identify exceptions where several FDs may „coexist" in one and the same relation !

- There are FDs which are derivable from other FDs, already known. An important example are so-called transitive FDs:

$\alpha \rightarrow \beta$ is a transitive FD if there is an attribute set $\gamma$, such that $\alpha \rightarrow \gamma$ and $\gamma \rightarrow \beta$ are both FDs, but not $\gamma \rightarrow \alpha$.

- Every such transitive case leads to a (new) functional dependency.

- There are two other such inference rules for FDs, called the „Armstrong axioms“ (as they have been discovered by the Canadian scientist W. Armstrong).

| | |
|---|---|
| $\beta \subseteq \alpha \quad \Rightarrow \quad \alpha \rightarrow \beta$ | Every subset depends on its superset. |
| $\alpha \rightarrow \beta \quad \Rightarrow \quad \alpha\gamma \rightarrow \beta\gamma$ | Augmentation on both sides. |

- Not to be found in the literature, but quite useful: Special notion for FDs which are not transitive FDs:

$\alpha \rightarrow \beta$ is a direct FD if there is no attribute set $\gamma$, such that $\alpha \rightarrow \gamma$ and $\gamma \rightarrow \beta$ are both FDs, but not $\gamma \rightarrow \alpha$.

- Properly determining FDs and investigating their properties is the basis for each meaningful decomposition of relations into components free of redundancies.

> **„Codd's recepy"** (in short from):
> Redundancies can be safely avoided if FDs always originate from candidate keys of a relation.

- Codd defined various degrees of FD separation, called normal forms – the process of transforming a given schema into relations all of which exhibit a given normal form is called normalization.

- The most important normal form is the third normal form (short: 3NF) defined as follows:

> A relation is in 3rd normal form ⇔
> Each non-key attribute functionally depends **directly** on each candidate key of the relation .

- There are various other normal forms (1NF, 2NF, 4NF and others).

- Our example schema from the geographic domain originally was not in 3NF, as it still contains a transitive dependency pointing from a candidate key (City) to a non-key attribute (Capital):

$(\mathfrak{R}, \mathcal{F})$

| City | State | Capital |
|---|---|---|
| Bonn | NW | Düsseldorf |
| Köln | NW | Düsseldorf |
| Essen | NW | Düsseldorf |
| . . . | | |

~~3NF~~

$(\mathfrak{R}_1, \mathcal{F}_1)$ 3NF

| City | State |
|---|---|

$(\mathfrak{R}_2, \mathcal{F}_2)$ 3NF

| State | Capital |
|---|---|

- After decomposition, however, each of the resulting component schemas is in 3NF! Such a decomposition will always be possible.

- The „lost“ dependency City → Capital can be reconstructed by means of the transitivity axiom of Armstrong.