Introduction to Databases
(Winter Term 2020/2021)

Deductive Databases
(Summer Term 2020)

(c) Prof Dr. Wolfgang May
Universität Göttingen, Germany
may@informatik.uni-goettingen.de

Introduction to Databases (BSc):

2+1 SWS, 5 ECTS: Ch. 1-3, 5; overview of 4+6

3+1 SWS: Ch. 1-6

4+2 SWS: Ch. 1-8

Database Theory/Deductive Databases (MSc): Ch. 8-12

Chapter 1
Basic Notions

Context and Overview

- databases are used in ... economy, administration, research ...
- originally: storage of information
  relational model, SQL
- evolution: information systems, combining databases and applications
- today: Web-based information systems, electronic data exchange
  $\rightarrow$ new challenges, semistructured data, XML

APPLICATION PROGRAMS VS. DATABASES

(Application) Programs	Databases

Runtime Environment
• short-lived computation
	Persistent Storage + Access
	• long-lived model of an application domain
	  • schema
	  • data
	• temporary connections/access by application programs

APPLICATION PROGRAMS VS. DATABASES

(Application) Programs	Databases
Runtime Environment	Persistent Storage + Access
Programming Paradigms	
value-oriented	set-oriented, large amounts of data
variables	implicitly specified sets, iterators
procedural/imperative Pascal, C, C++, Java	declarative SQL
note: in both cases, object-orientation is added:	
Java: OO + imperative core	OQL: SQL + OO

APPLICATION PROGRAMS VS. DATABASES

(Application) Programs	Databases
Runtime Environment	Persistent Storage + Access
Operating Modes	
single-user	multiuser • user accounts
one-thread	concurrency

	• transactions
	• safety
	  • access control
	  • against physical failure
	• consistency, integrity

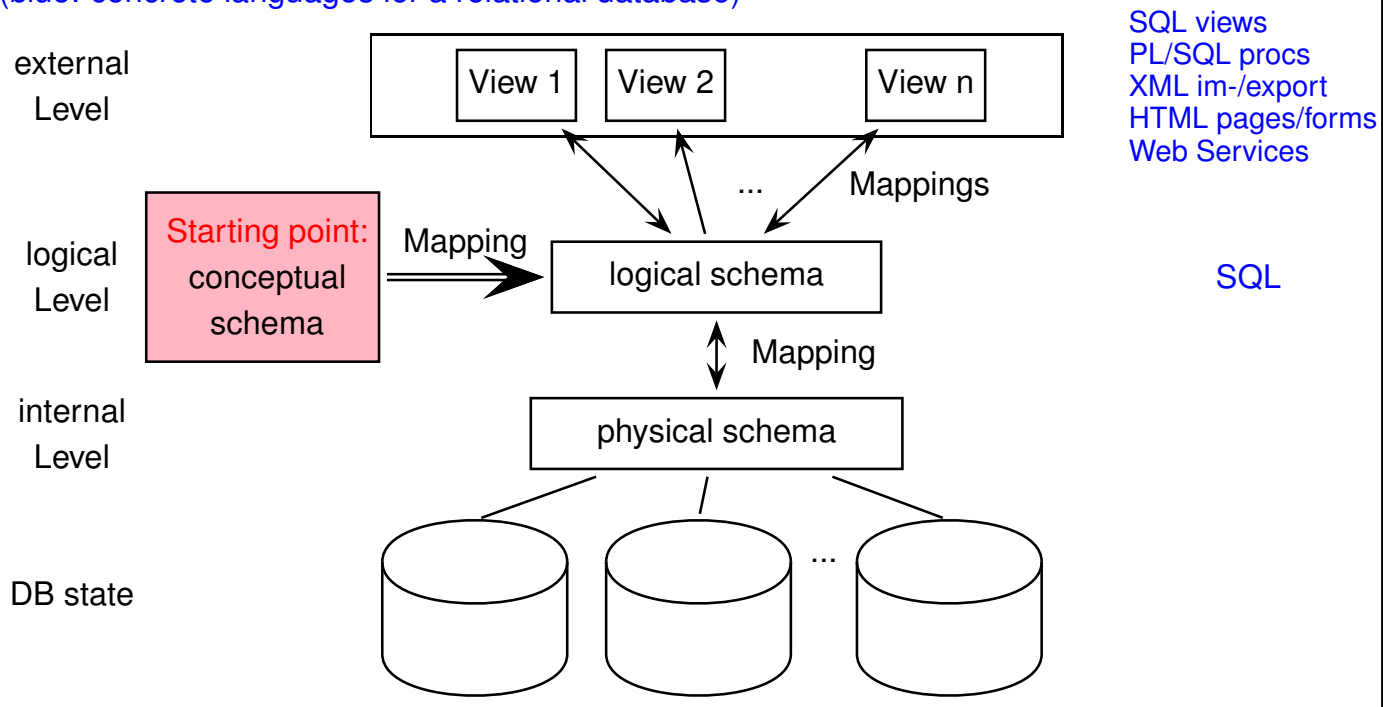
APPLICATION PROGRAMS VS. DATABASES

(Application) Programs	Databases
user-defined data structures	fixed data model user-defined schema internal storage aspects
small runtime data	large persistent data
program/algorithm	query
algorithms	internal algorithms
	transactions & safety

- A database system is a specialized data structure, with specialized behavior and -in contrast to most other data structures- specialized programming languages.
- (Note: the same holds for the XML data model.)

3-Level Architecture of a DBMS (ANSI/SPARC Standard, 1975)

(blue: concrete languages for a relational database)



- global model of the application domain: conceptual schema

Schema Levels

Conceptual schema: The conceptual schema defines the model of the world as represented in the database, using an *abstract* formalism: [intended to be stable]

- definition of all relevant *object types* and *relationship types*,
- including *integrity constraints*,
- independent from the implementation,
- changes only rarely after being defined once.

Logical schema: A mapping from the conceptual schema to a concrete data model.

Physical (internal) schema: Data structures for storing the data, and additional auxiliary structures for more efficient data handling (e.g., indexes).
[can be changed for optimizations]

Views/external schema/subschemata: Depending on the needs of special users, required object types and relationship types can be defined, derived from the ones that are defined in the *conceptual* model. [easily adaptable to users' needs]

Mappings:

- define how the objects of the logical level are mapped to the physical level.
- define how the objects of the external level are defined based on those of the logical level.

Data Independence

Independence of the three levels:

- levels connected by mappings,
- every level may use a different data model,
- every level can be changed without affecting the others.

logical data independence: Changes and restructurings in the conceptual schema can be hidden against the external schema (by appropriate redefinition of mappings).

physical data independence: Modifications in the internal schema (splitting a table, adding an index, etc.) do not effect the conceptual schema (only redefinition of the mappings).

Schema and State

On each level, there exist the notions of *schema* and *state*:

Database schema: the schema contains the *metadata* of the database, i.e., describes the concepts (e.g., object types and relationship types).

Database state: the state of a database (system) is given by the set of all data contained in the system. It represents the objects and relationships that hold in the application domain at a given timepoint.

With the time passing, a database passes through several database states.

- The admissible states are defined in terms of the conceptual schema (e.g., by integrity constraints),
- the database state itself is represented in the physical schema,
- users may access it through their views, using the external schema.

Data Dictionary: contains the definitions and mappings of the schemas.

Chapter 2
Data Models

A *data model* defines modeling (specification-) constructs which can be used for modeling an application domain (in general, both its (static) data structures and its (dynamic) behavior).

- definition of *data structures* (object types and relationship types),
- definition of *integrity constraints*,
- definition of *operations* and their effects.

A data model consists of

- a **D**ata **D**efinition **L**anguage (DDL) for defining the schema: object types, relationship types, and integrity constraints.
- a **D**ata **M**anipulation **L**anguage (DML) for processing *database states* (inserting and modifying data)

  Operations are *generic operations* (querying, inserting, modifying, and deleting objects or relationships), or *procedures* that are constructed from basic operations.

DATA MODELS

Kinds of Modeling:

- *conceptual modeling*: abstract model of the semantics of an application
- *logical modeling*: more formal model, similar to an *abstract datatype/API* that has actual implementations

Some prominent data models:

- Network Model (1964; CODASYL Standard 1971; “legacy”); Hierarchical Model
- Entity-Relationship-Model (1976, conceptual model, only static concepts) [this lecture]
- Unified Modeling Language – UML (∼1995, conceptual model) [Software Engineering] comprehensive formalism for specifying processes, based on the object-oriented model.
- Relational Model (1970; simple, but clear *logical* model) [this lecture]
- XML (since 1996; popular since 1998) [Semistructured Data and XML lecture]
- RDF data model (since 1997; popular since 200X); even more basic – only a single ternary relation – (subject, predicate, object) [Semantic Web]

2.1 Entity Relationship Model (ERM)

- purely *conceptual model*:
  Abstract description of the application domain in a graphical framework, which is then transformed into some logical data model (this lecture: relational model).
- This lecture uses the original "Chen Notation", named after Peter Pin-Shan Chen (born 1947 in Taichung, Taiwan; 1970-73 Harvard, 1974-78 MIT) who published it in 1976 in "The Entity-Relationship Model – Toward a Unified View of Data" in the *ACM Transactions on Database Systems* journal with min..max-Notation for cardinalities.
- some textbooks/lectures [e.g. the IKS lecture in our "Wirtschaftsinformatik" studies] and design tools use different notations (especially for the relationships and their cardinalities):
  - influenced by the earlier "Bachman Diagrams",
  - influenced by the later UML language (1990s);
  - most of them do not allow to model n-ary ($n > 2$) relationships directly.
  - information about the min/max-cardinalities is crucial for the mapping to the relational model.
- independent from what notation/tool you use: if you do it *correctly*, the result, i.e., the relational model obtained from the subsequent mapping step, will be the same.

2.1.1 Main Structural Concepts

The main structural concepts for describing a schema in the ERM are **Entities** and **Relationships**.

ENTITY TYPES

Entity type: An entity type represents a concept in the real world. It is given as a pair $(E, \{A_1, \ldots, A_n\})$, where E is the name and $\{A_1, \ldots, A_n\},\ n \geq 0$ are the attributes (literal-valued properties) of a type.

Attribute: a relevant property of entities of a given type. Each attribute can have *(literal) values* from a given *domain*.

Example 2.1
(Continent, {name, area})
(Country, {name, code, population, area}),
(City, {name, population, latitude, longitude, elevation}),
(Province, {name, area, population}), □

ENTITIES

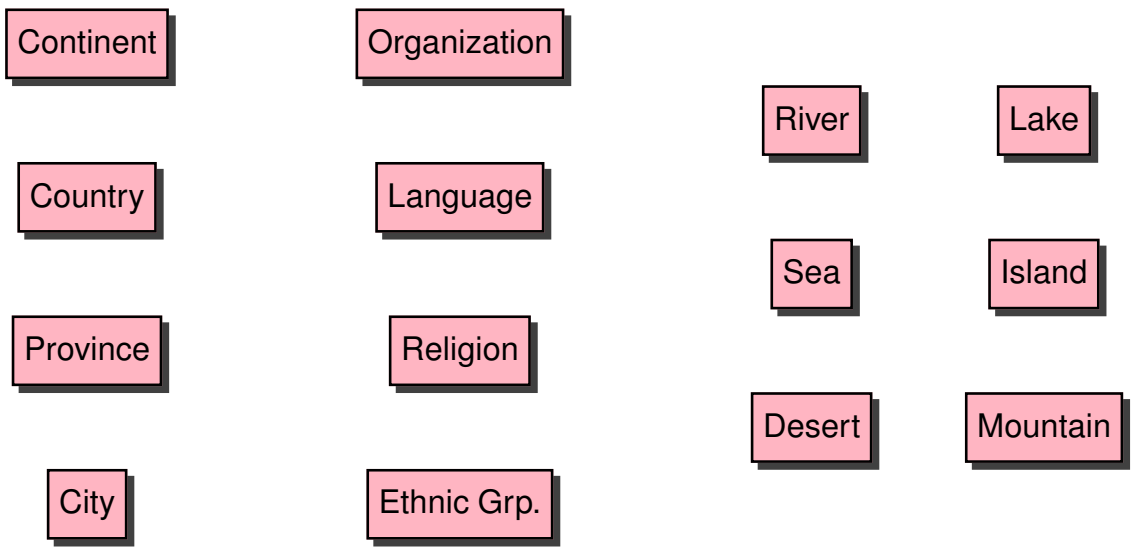
- An **entity set** e of an entity type E is a finite set of entities.
- each **entity** describes a real-world object. Thus, it must be of one of the defined entity types E. It assigns a value to each attribute that is declared for the entity type E.

Example 2.2
Entity set of the entity type (City, {name, population, latitude, longitude}):

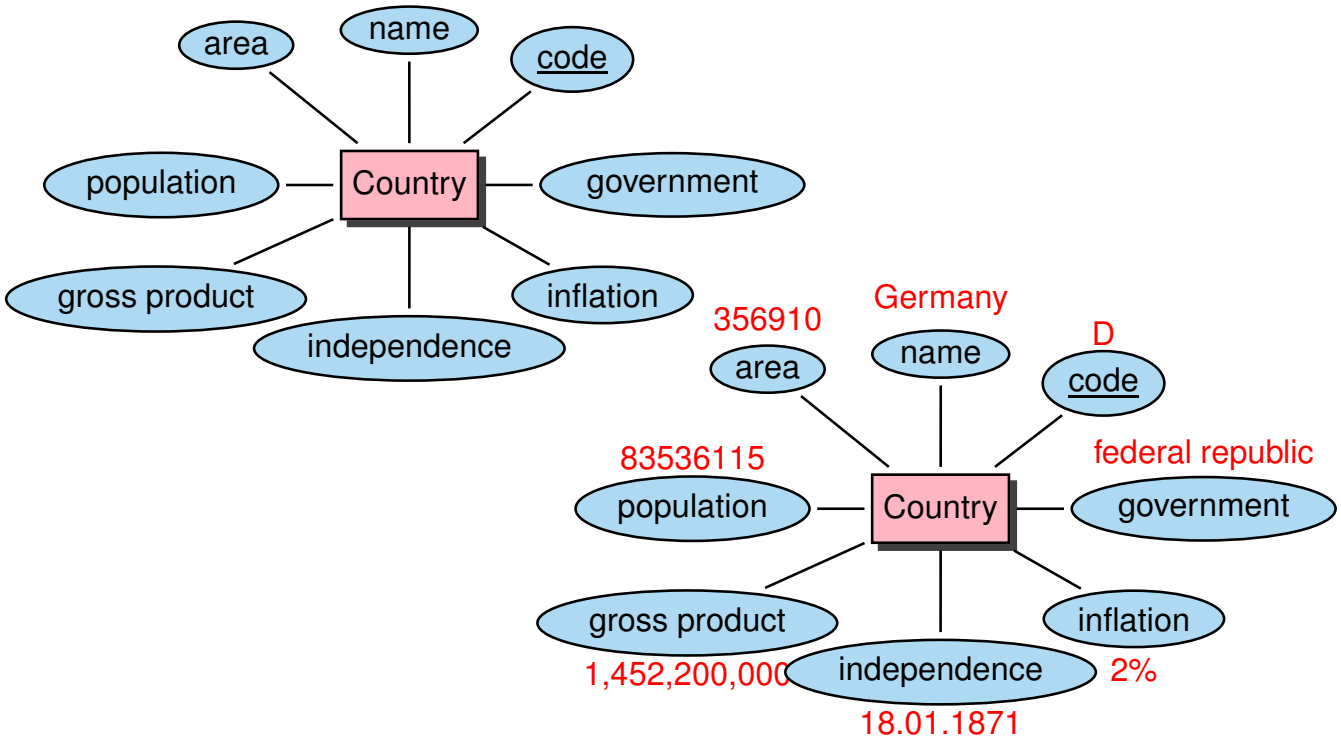
{(name: Aden, population: 250000, latitude: 13, longitude: 50),
(name: Kathmandu, population: 393494, latitude: 27.45, longitude: 85.25),
(name: Ulan Bator, population: 479500, latitude: 48, longitude: 107) } □

GRAPHICAL REPRESENTATION

- Entity types are represented as rectangles:



- Attributes are represented as ovals:



RELATIONSHIP TYPES

Relationship type: describes a concept of relationships between entities. It is given as a triple $(B, \{RO_1 : E_1, \ldots, RO_k : E_k\}, \{A_1, \ldots, A_n\})$, where B is the name, $\{RO_1, \ldots, RO_k\},\ k \geq 2$, is a list of *roles*, $\{E_1, \ldots, E_k\}$ is a list of entity types associated to the roles, and $\{A_1, \ldots, A_n\}, n \geq 0$ is the set of attributes of the relationship type.

In case that $k = 2$, the relationship type is called **binary**, otherwise n**-ary.**

Roles are pairwise different – the associated entity types are not necessarily pairwise distinct. In case that $E_i = E_j$ for $i \neq j$, there is a **recursive** relationship.

As long as there are no disambiguities, a role may be identified with the corresponding entity type. Roles are useful e.g. for annotating the semantic aspects of the reality.

Attributes describe relevant properties of relationships of a given type.

Example 2.3
(capital, {Country, City}, ∅),
(encompasses, {Continent, Country}, {percent}),
(belongsto, {Province, Country}, ∅),
(flowsinto, {tributary: River, main: River}, ∅) □

Relationship Types and Relationship Instances

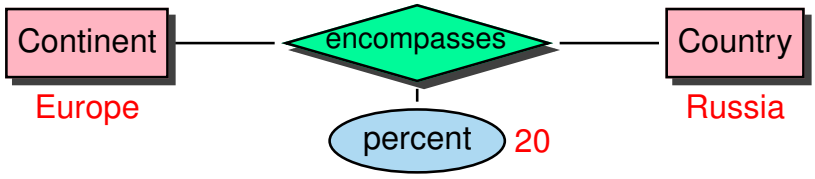
- A **relationship set** b of a relationship type B is a finite set of relationships.
- A **relationship (instance)** of a relationship type B is defined by the entities that are involved in the relationship, according to their associated roles. For each role, there is exactly one entity involved in the relationship, and every attribute is assigned a value.

(see examples next slide)

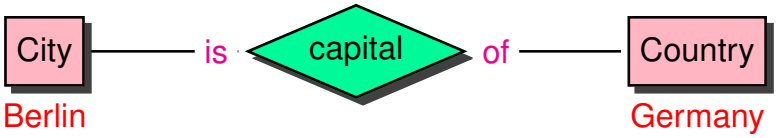
Relationships

recursive relationship type

relationship type with attributes

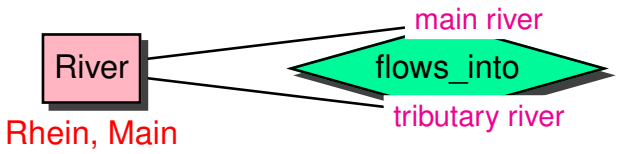


relationship type with roles

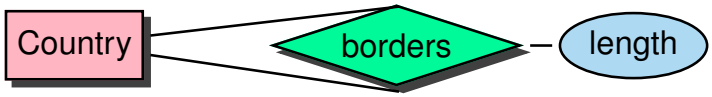


Recursive Relationship Types

- Non-symmetric recursive relationship types require the use of roles:



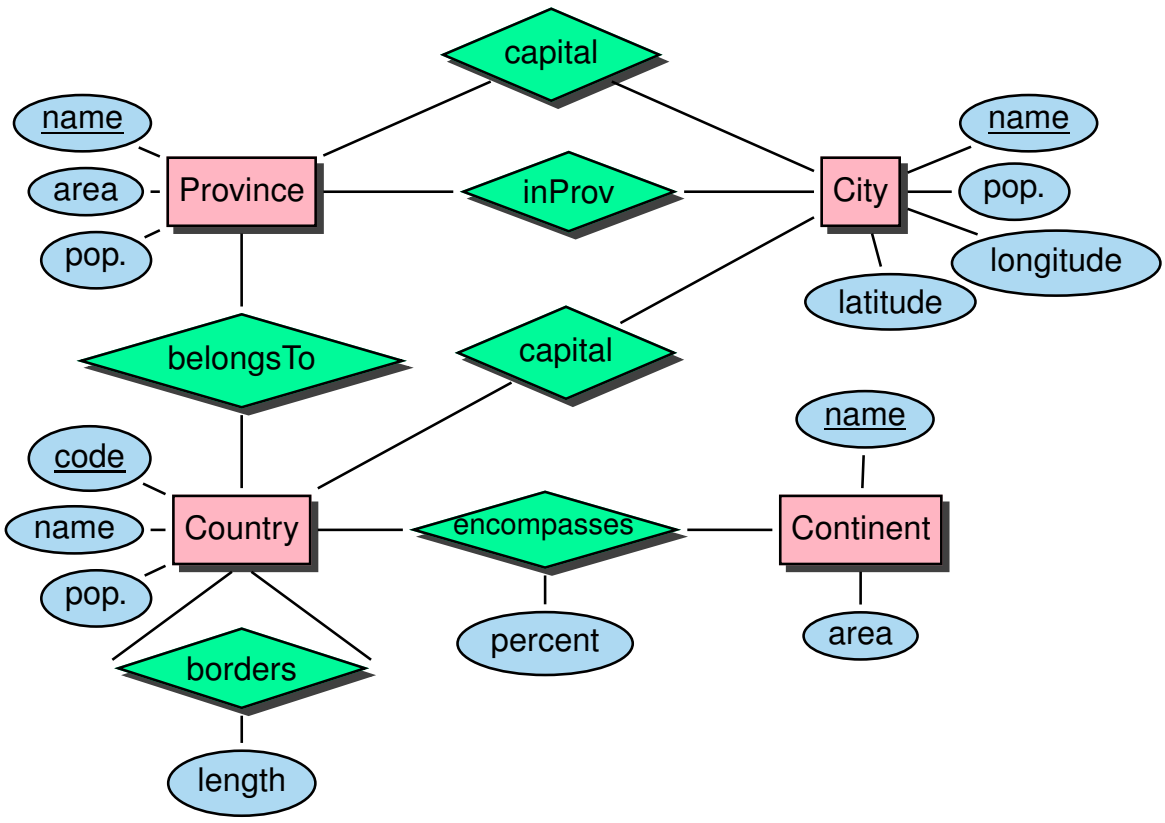
- Symmetric recursive relationship types are indicated by the absence of roles:



Aside – storage aspects:
For symmetric relationship types, it is sufficient to store them only in one direction:

- saves memory;
- define a “view” as symmetric hull over the relation;
- bidirectional storage: risk of inconsistencies, e.g. border(D,CH,334) and border(CH,D,335).

Example: ER Model of a geographical database



DATABASE STATES

A **(database) state** associates the entity types and relationship types of a given schema with an **entity set** and a **relationship set**, respectively.

(cf. examples above – can be represented graphical as a graph/network)

2.1.2 Integrity Constraints

There are additional constraints on the admissible *database states*.

Domains: Every attribute is assigned a domain which specifies the set of admissible values.

Keys: a *key* is a set of attributes of an entity type, whose values together allow for a unique identification of an entity amongst all entities of a given type (cf. *candidate keys, primary keys*).

Relationship Cardinalities: every relationship type is assigned a cardinality that specifies the minimal and maximal number of relationships in which an entity of a given type/role may be involved.

Referential Integrity: each entity which occurs in a relationship in any database state must also exist in the entity set of this state
(condition is trivial when represented as a graph, but crucial later in the relational model)

... to be described in detail on the following slides

KEYS

A *key* is a set of attributes of an *entity type*, whose values together allow for a unique identification of an entity amongst all entities of a given type (cf. *candidate keys, primary keys*).

For an entity type $(E, \{A_1, \ldots, A_n\})$ and an entity set e of E, a set $K \subseteq \{A_1, \ldots, A_n\}$ satisfies the **key constraint** if:

- K uniquely **identifies** any element $\mu \in e$, i.e., for all $\mu_1, \mu_2 \in e$, if μ_1 and μ_2 have the same values for all attributes in K, then $\mu_1 = \mu_2$.

Declaring a set of attributes to be a key thus states a condition on all admissible database states.

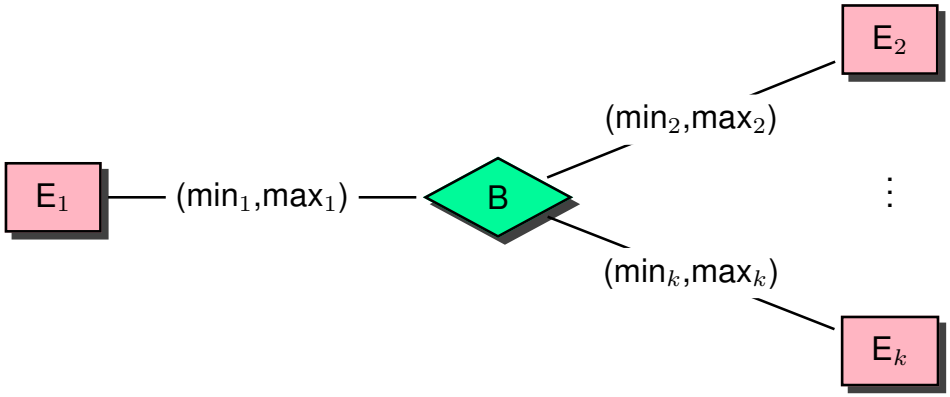
Graphically, key attributes are distinguished by underlining.

RELATIONSHIP CARDINALITIES

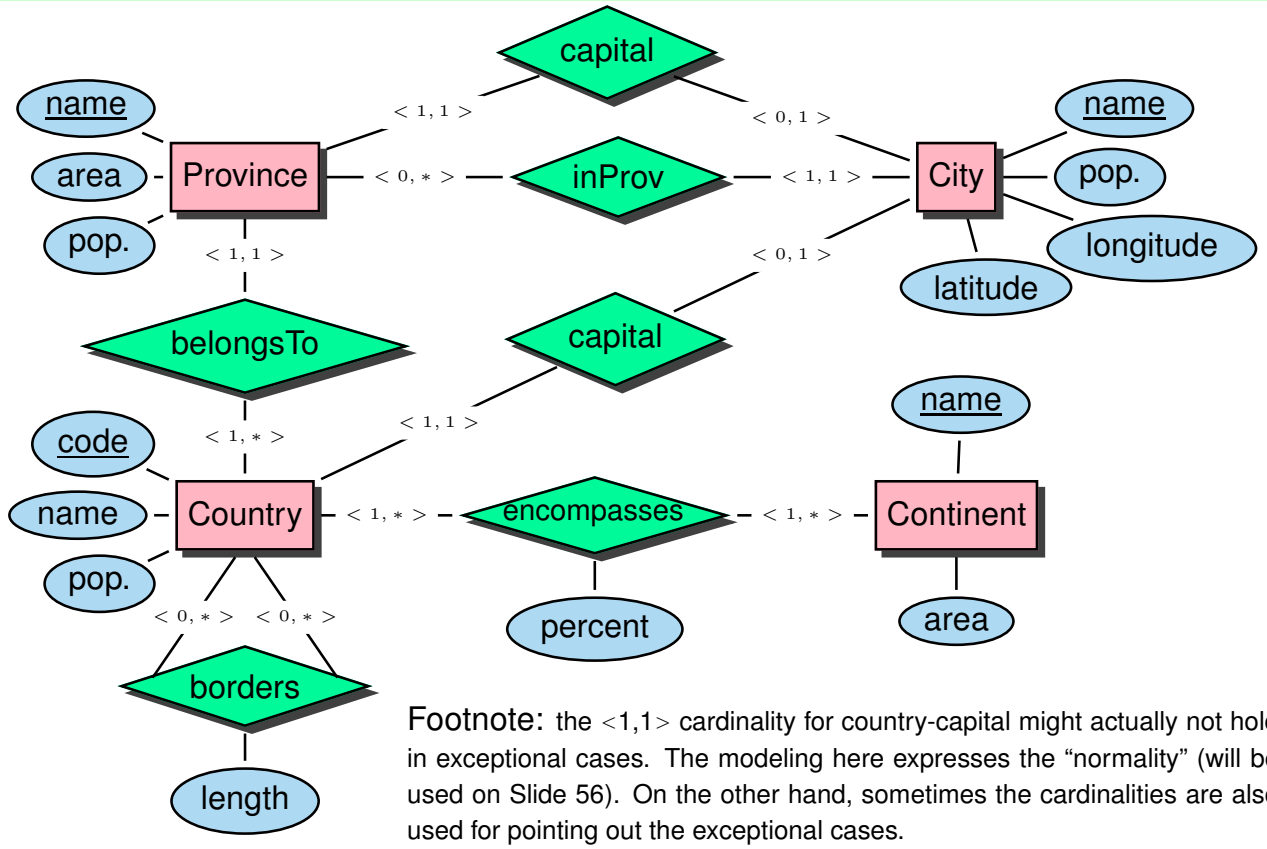
Every relationship type is assigned a cardinality that specifies the minimal and maximal number of relationships in which an entity of a given type/role may be involved.

The **cardinality** of a relationship type B wrt. one of its roles RO is an expression of the form (min, max) where $0 \leq min \leq max$, and $max = *$ means "arbitrary many".

A set b of relationships of relationship type B satisfies the cardinality (min, max) of a role RO if for all entities μ of the corresponding entity type E the following holds: there exist at least min and at most max relationships b in which μ is involved in the role RO.



Example: ER Model of a geographical database



Footnote: the <1,1> cardinality for country-capital might actually not hold in exceptional cases. The modeling here expresses the "normality" (will be used on Slide 56). On the other hand, sometimes the cardinalities are also used for pointing out the exceptional cases.

Comment on Minimal Cardinalities

- Conceptual modeling: minimal cardinality describes the allowed state of an up-and-running database:
  - 0 means the relationship is optional
  - 1 means the relationship is mandatory
- during initialization, and when new items are added, these may be temporarily violated (cf. country-capital <1,1>. How to add a new country?)

Additional Notions for Cardinalities

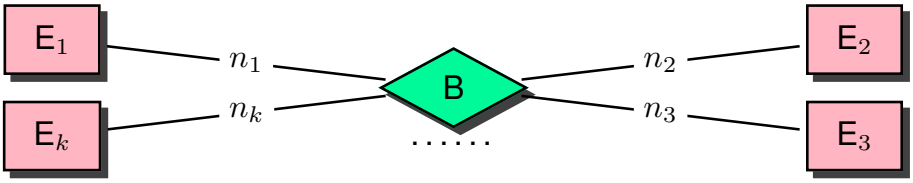
For *binary* relationships, the following notions are used:

- if $max_1 = max_2 = 1$, it is called a $1:1$-relationship.
  is_capital $\subseteq$ Country $\times$ City is a 1:1-relationship
- if $max_1 > 1, max_2 = 1$, it is called a $n:1$-relationship (functional relationship) from E_2 to E_1, and a $1:n$-relationship from E_1 to E_2.
  has_city $\subseteq$ Country $\times$ City is a 1:n-relationship
- Otherwise, it is called an $n:m$-relationship.
  borders $\subseteq$ Country $\times$ Country is an n:m-relationship

ASIDE: AN ALTERNATIVE NOTATION FOR CARDINALITIES

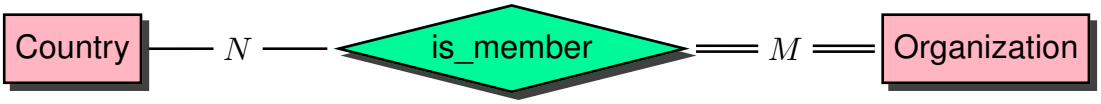
Indicates only the *maximum* cardinality: 1,2,3, ... N, M, ...
and is to be read the other way round:

- Each combination $(e_1, \ldots, e_{i-1}, e_{i+2}, \ldots, e_k)$ (e_j of type E_j) is in relation with at most n_i entities of type E_i:

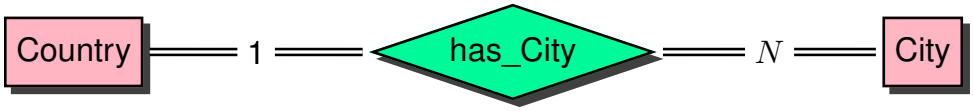


Mandatory relationships can be indicated by double lines:

- Each country is a member of arbitrary many organizations (maybe none); each organization has at least 1, and arbitrary many countries as members (n:m):



- Each country has at least one, and arbitrary many cities, each city belongs to exactly one country (1:n):



REFERENTIAL INTEGRITY

Each entity which occurs in a relationship in any database state must also exist in the entity set of this state.

For a relationship type B with relationship set b, a role RO of B that is connected to an entity type E with entity set e, b and e **satisfy the referential integrity** wrt. RO, if for every entity μ that is associated with some $\nu \in b$ under the role RO, $\mu \in e$ holds.

Note:

- referential integrity is inherent to the ER Model, thus, it is not necessary to care for it.
- there are data models (e.g., the *relational model* (which is described later) where referential integrity must be enforced explicitly).
  (postpone the discussion to the relational model)

2.1.3 Further Concepts

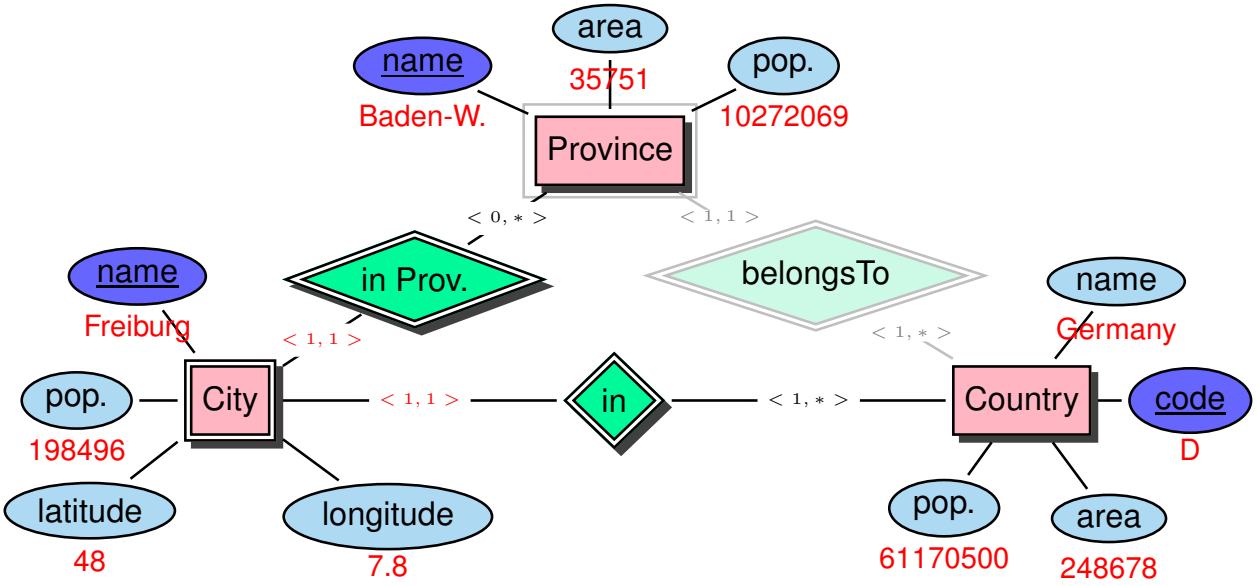
Weak Entity types

A weak entity type is an entity type without a key.

Thus entities of such types must be identified by the help of another entity (see the following figure).

- Weak entity types must be involved in at least one $n:1$-relationship with a strong entity type (where the strong entity type stands on the 1-side).
  - this relationship is called an *identifying relationship*,
  - the corresponding entity type is called an *identifying entity type*.
- They usually have a **local** key, i.e., a set of attributes that can be extended by the primary keys of the corresponding strong entity type to provide a key for the weak entity type (*key inheritance*).
  (cases where they do not have a local key are rare, but do exist; usually resulting from *reification*, cf. Slide 38.)
- Note that weak entity types and their identifying relationship types have a special notation.

Weak Entity Types



There is also a Freiburg/CH
and Freiburg/Elbe, LowerSaxony (Niedersachsen)

(Note: Province is also itself a weak entity type since several countries have provinces with the same name (e.g., Western, Distrito Federal, Amazonas))

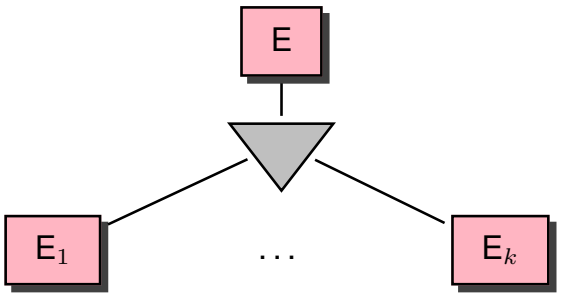
EXTENSIONS OF THE ERM: MULTIVALUED AND COMPLEX ATTRIBUTES

Attributes can be

- set-valued or multi-valued,
- structured

EXTENSIONS OF THE ERM: GENERALIZATION/SPECIALIZATION

- covers the general idea of a class hierarchy between entity types.

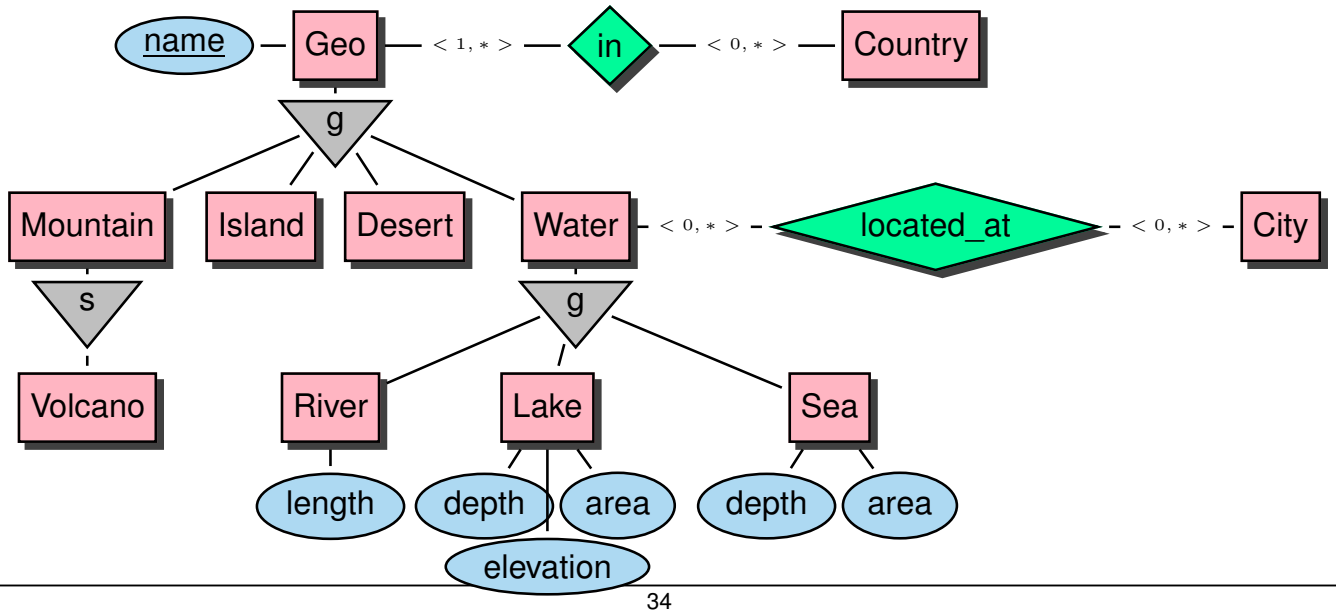


E is called **supertype**, E_i are **subtypes** for $1 \leq i \leq k$. Each entity of a subtype s also an entity of the supertype.

- The common attributes and relationships are assigned to the more general type.
- The attributes and relationships of the supertype are also applicable to the subtypes (which may define further attributes and relationships).

Generalization/Specialization

- Geographical things such as rivers, lakes, seas, mountains, deserts, and islands (no lowlands, highlands, savannas, fens, etc). All such geographical things have in common that they have names and theat they are involved in *in*-relationships with countries.
- Rivers, lakes, and seas are *waters*. These can e.g. be involved in *located-at* relationships with cities.



Generalization/Specialization

Integrity Constraints (cf. UML)

- Common integrity constraints **ISA:** ISA is satisfied in a database state if the entity sets of the subtypes are subsets of the entity sets of the supertype,
- optional integrity constraint **Disjointness:** if the entity sets of the subtypes are disjoint,
- optional integrity constraint **Covering:** if the union of the entity sets of the subtypes cover the entity set of the supertype.

Intuition Annotations

- Generalization 

  Bottom-up: from the subclasses, the superclass is “discovered” as a general concept.
- Specialization 

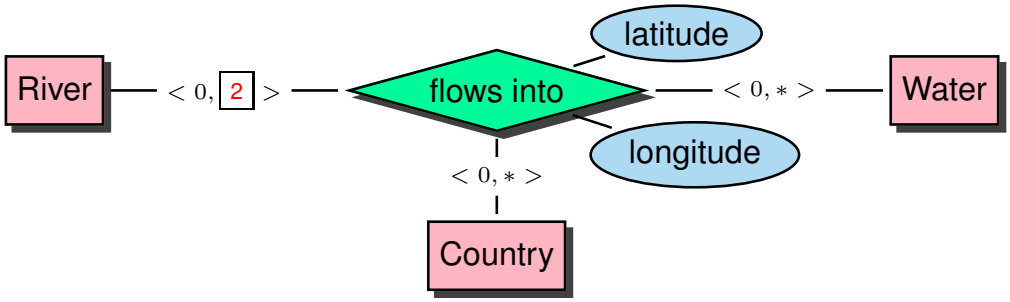
  Top-Down: from the superclass, subclasses are “discovered” as restricted concepts.
- generalization usually leads to “covering”, and in most cases also to disjointness.
- specialization usually leads to non-covering.

EXTENSIONS OF THE ERM: AGGREGATION

The ERM does not allow to define relationship types that involve relationship types (note that attributes of relationship types are allowed).

- This restriction can be overcome by defining artificial entity types for “the relationship”.

A river flows (finally) into a sea/lake/river; more detailed, such a *relationship instance* is related to one or two countries:

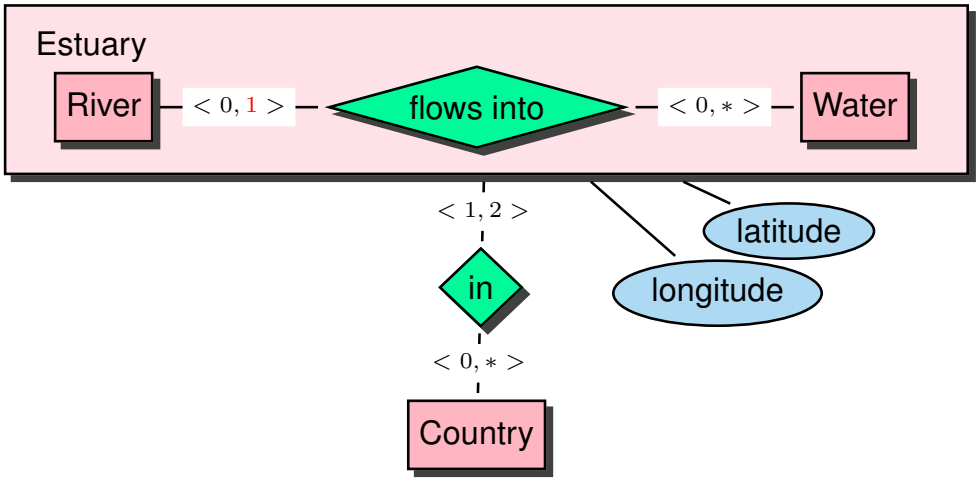


This representation is ambiguous: A river could flow into two waters!?
(at different latitude/longitudes?)

Aggregation

- originally introduced in *J. Smith, D. Smith: Database Abstractions: Aggregation. In: Comm. of the ACM. Vol. 20, Nr. 6, 1977, pp. 405-413*

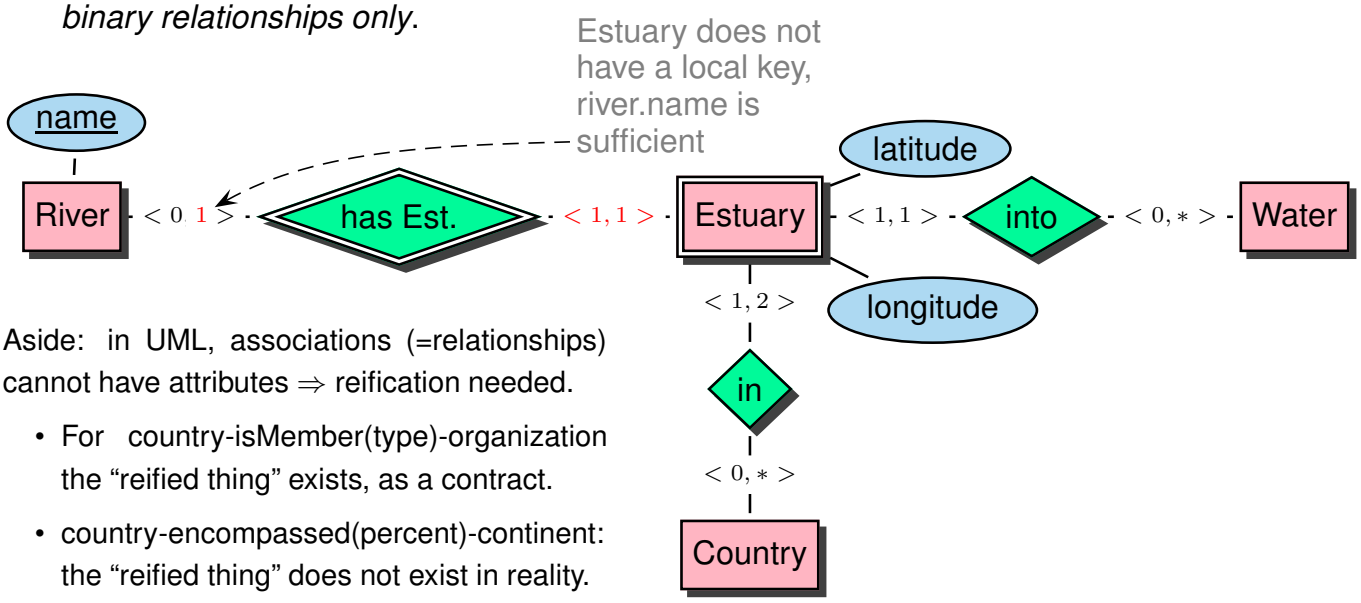
Using an “aggregation entity type”, this information can be specified much clearer by introducing an *aggregate type estuary* for the “river flows into another water” relationship:



The cardinalities allow for expressing a more detailed semantics than with the plain ternary relationship type.

General Modeling Strategy: Reification

- Since the 1990s, this modeling strategy is called Reification ("turning something into a thing"), and applied in several modeling approaches (ERM, UML, XML, RDF) (UML: [Software Engineering Lecture] Association classes)
- Reification can replace ER-specific modeling concepts like n-ary relationship types or aggregation entity types by introducing new (usually weak) entity types, and then using *binary relationships only*.



Aside: in UML, associations (=relationships) cannot have attributes ⇒ reification needed.

- For country-isMember(type)-organization the "reified thing" exists, as a contract.
- country-encompassed(percent)-continent: the "reified thing" does not exist in reality.

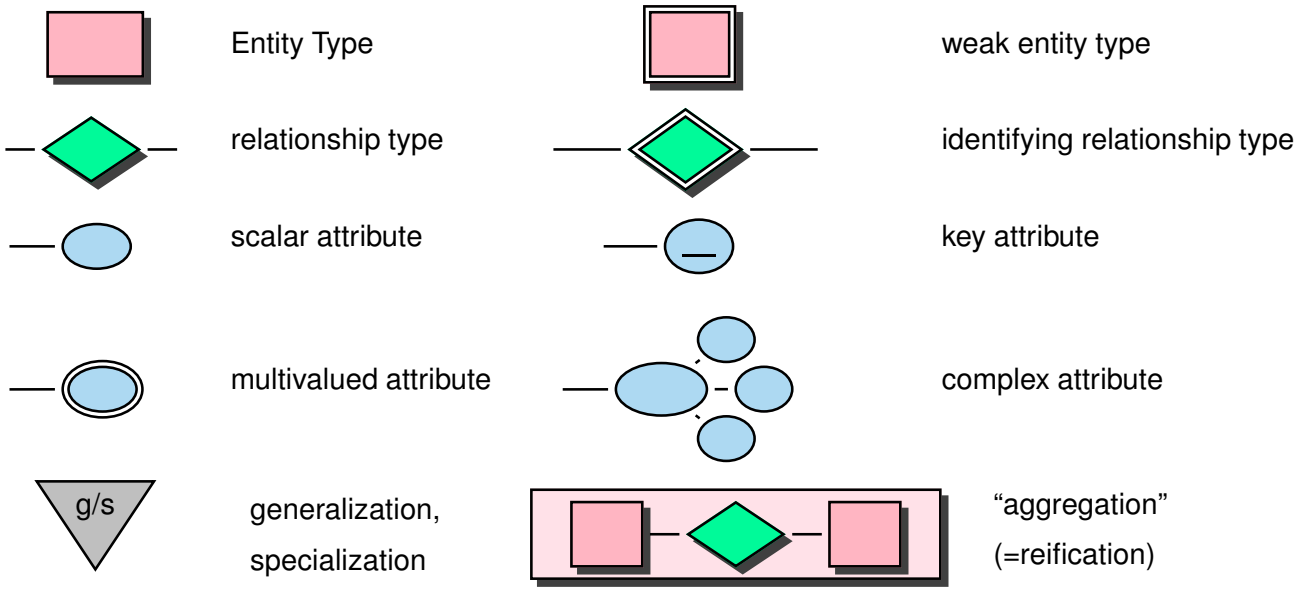
2.1.4 Discussion ERM

- With the structuring concepts of the ERM and its extensions, the *static* aspects of a relevant excerpt of the real world can be modeled semantically adequate in a natural way.
- The graphical representation is also understandable for non-computer-scientists.
- The ERM is useful
  - in the early stages of the design of the database (i.e., when designing the conceptual schema) when discussions with the potential users take place.
  - for documentation (!)
- The ERM can easily be transformed into the data models of existing, real-world database systems (especially, into the relational model – as will be shown in the sequel).
- There are no relevant DBMS that use the ERM directly. They are subsumed by **object-relational** and **object-oriented** DBMS (and more recently also by RDF-DBMS).

DISCUSSION ERM (CONT'D)

- There is a more complex and more expressive language:
  UML (Unified Modeling Language):
  - static aspects are described in more detail than in the ERM, using notions of a fully object-oriented model,
  - dynamic aspects are also described graphically,
  - coarser granularities for describing *information systems* and *workflows* are provided.

SUMMARY: GRAPHICAL NOTATION OF ER CONSTRUCTS



Convention: names of entity types start with a capital letter, names of relationship types and attributes start with non-capital letters.

2.1.5 Some Exercises

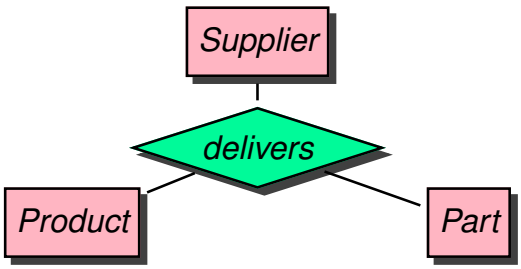
Exercise 2.1
Consider a binary relationship type and the cardinalities $(0,1)$ *and* $(1,*)$*. Investigate all possible ways how to assign these relationship cardinalities to the relationship type. For each variant, give a nontrivial state that satisfies them, and a state that violates them.* □

Exercise 2.2
Discuss ER schemata for the following scenario:

- *All students work on projects. For this, they need tools.* □

Some Exercises (Cont'd)

Exercise 2.3
Consider a ternary relationship type between the entity types supplier*,* product*, and* part *(where suppliers deliver parts for a product).*



- *Check whether this situation can be represented by using only binary relationship types.*
  - *Under which conditions is it possible?*
  - *Can such situations be described by the relation cardinalities?*
- *Show that for an ER schema consisting of a ternary relationship there is always an equivalent ER-Schema that consists of three binary relationship types* and an additional entity type*.* □

DEVELOPMENT OF A DATABASE APPLICATION

(cf. 3-Level-Architecture, Slide 6)

Conceptual Design: structuring of the requirements for the representation of the relevant excerpt of the real world:

- independent from the database system to be used (phys. level),
- independent from the detailed views of the users (external schema).

results in the **conceptual schema**, in general an ER schema (or specified in UML).

... but this cannot be “used” in a real database.

Implementation Design: convert into the actual, *logical* schema of the logical level in a logical model (Relational Model),

(process to be continued then on Slide 49)