# Introduction to Databases
# (Winter Term 2004/2005)

(c) Prof Dr. Wolfgang May
Universität Göttingen, Germany

may@informatik.uni-goettingen.de

Wahlpflichtvorlesung Informatik mit Übungen
2+1 SWS, 4 ECTS-Punkte



# 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 | Persistent Storage + Access |
| • short-lived computation | • long-lived model of an application domain<br>  • schema<br>  • data<br>• temporary connections/access by application programs |



## Application Programs vs. Databases

| (Application) Programs | Databases |
|---|---|
| Runtime Environment | Persistent Storage + Access |
| **Programming Paradigms** | |
| value-oriented | set-oriented,<br>large amounts of data |
| variables | implicitly specified sets, iterators |
| procedural/imperative<br>Pascal, C, C++, Java | declarative<br>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<br>• user accounts |
| one-thread | concurrency |
| | • transactions<br>• safety<br>    • access control<br>    • against physical failure<br>• consistency, integrity |

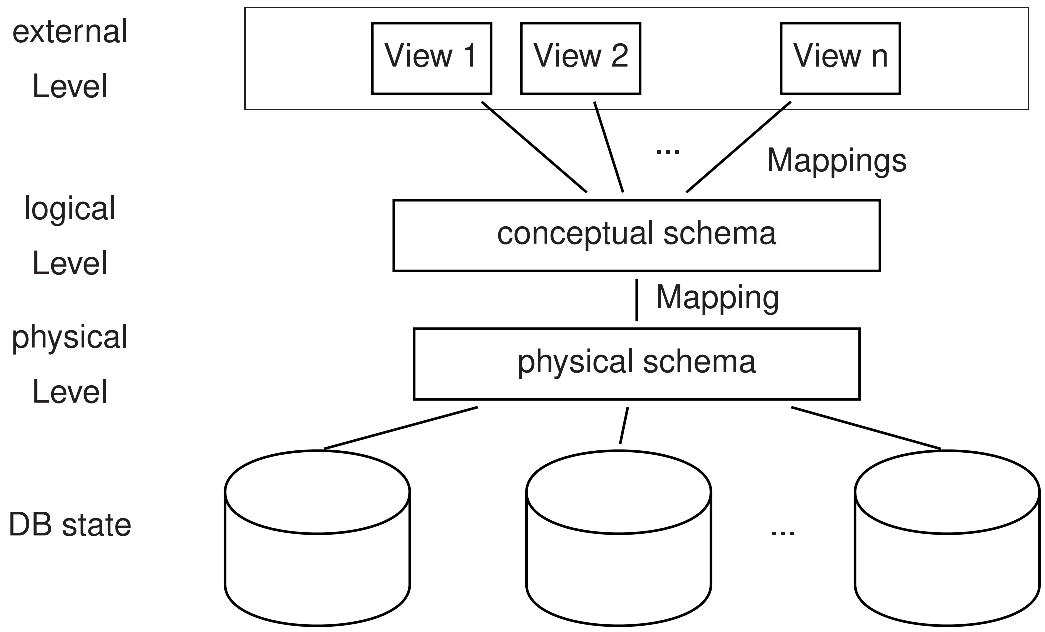


## Application Programs vs. Databases

| (Application) Programs | Databases |
|---|---|
| user-defined data structures | fixed data model<br>user-defined schema<br>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)




- 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:

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

**Physical (internal) schema:** Data structures for storing the data, and additional auxiliary structures for more efficient data handling (e.g., indexes).

**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.

**Mappings:**

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

## 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.



## 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).

# 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

... there are different data models and modeling languages ...

- Entity-Relationship-Model (only static concepts)
- Unified Modeling Language (UML)
  comprehensive formalism for specifying processes, based on the object-oriented model
- Relational Model (very basic model)

# 2.1 Entity Relationship Model (ERM)

## 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 (value properties) of a type.

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

**Example 2.1**
*(Continent,* $\{$*name, area*$\}$*)*
*(City,* $\{$*name, population, longitude, latitude*$\}$*),*
*(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 $E$.

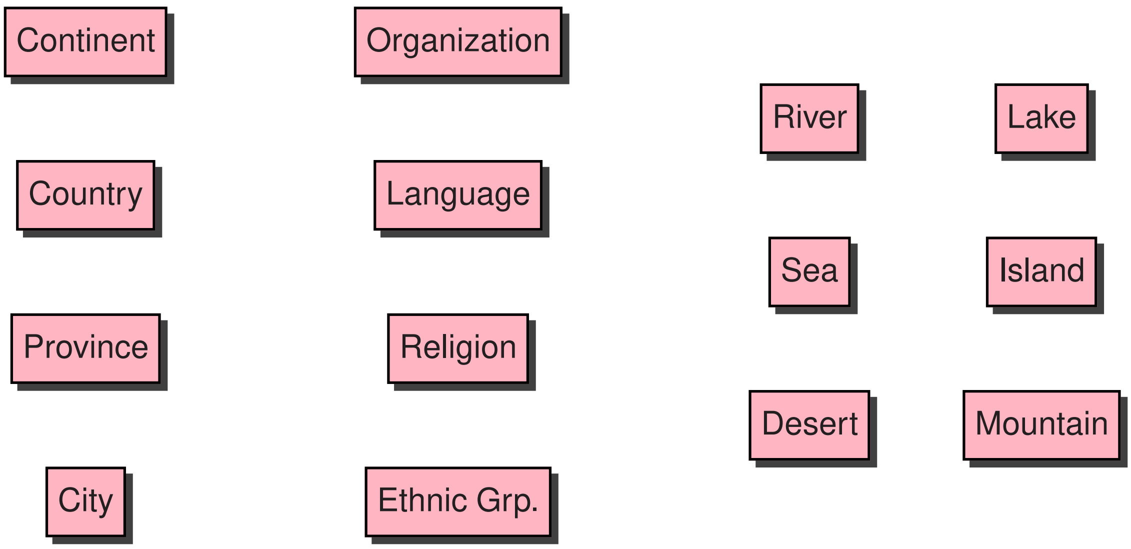
**Example 2.2**
*Entity set of the entity type (City, {name, population, longitude, latitude}):*

*{(name: Aden, population: 250000, longitude: 50, latitude: 13),*
*(name: Katmandu, population: 393494, longitude: 85,25, latitude: 27,45),*
*(name: Ulan_Bator, population: 479500, longitude: 107, latitude: 48)}* □

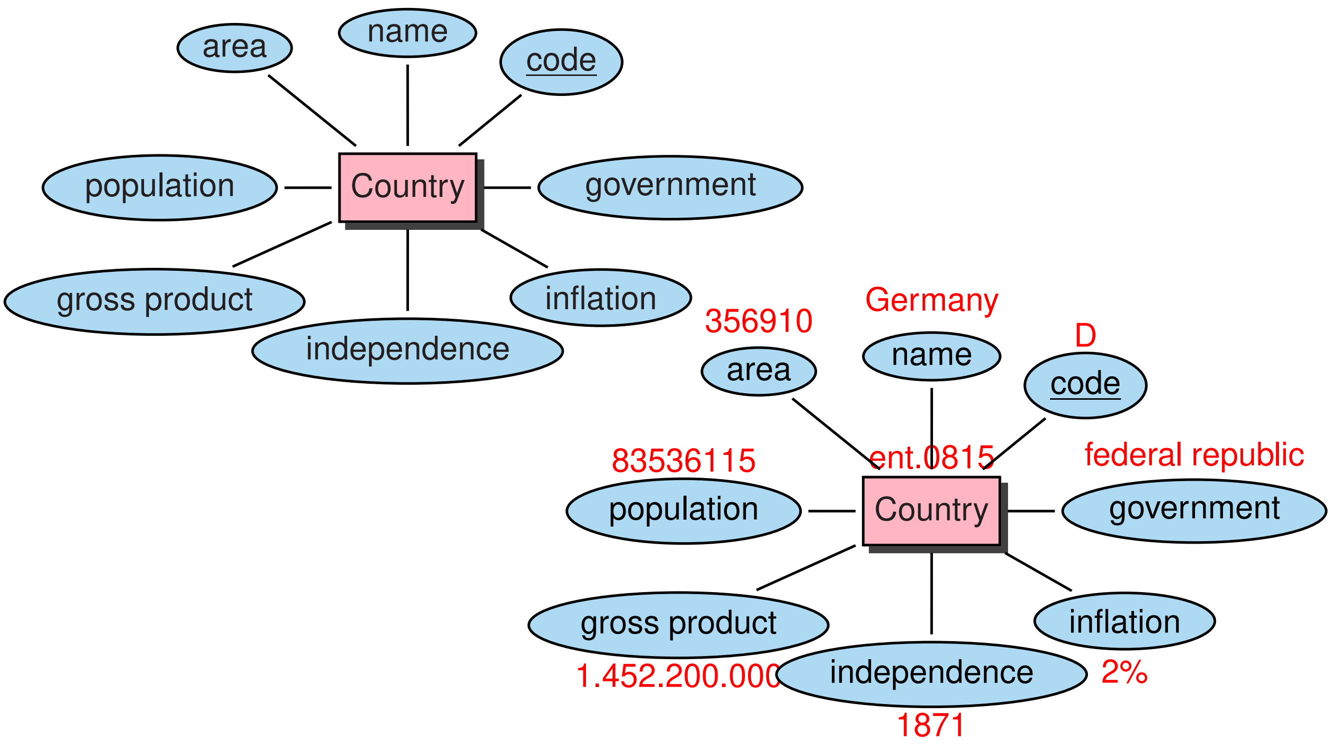


## GRAPHICAL REPRESENTATION

- 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,* $\{$*City, Country*$\}$, $\emptyset$*),*
*(encompasses,* $\{$*Continent, Country*$\}$, $\{$*percent*$\}$*),*
*(belongsto,* $\{$*Province, Country*$\}$, $\emptyset$*),*
*(flowsinto,* $\{$*tributary: River, main: River*$\}$, $\emptyset$*)* □

# Relationship Types (Cont'd)

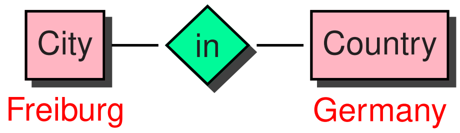
- A **relationship set** $b$ of a relationship type $B$ is a finite set of relationships.
- A **relationship** of a relationship type $B$ is defined by the entities that are involved in the relationship, according to their associated roles. For each role, there is exactly one entity involved in the relationship,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

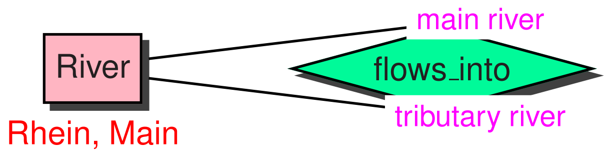
(to be extended with examples)



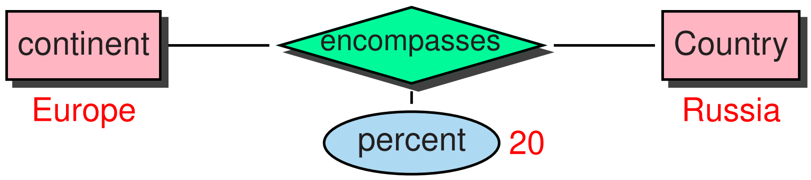
# Relationships

recursive relationship

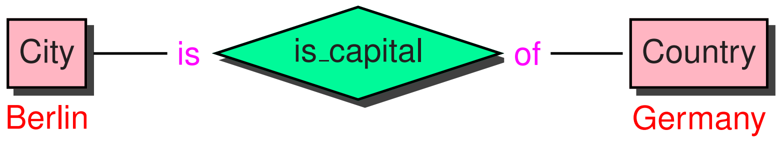


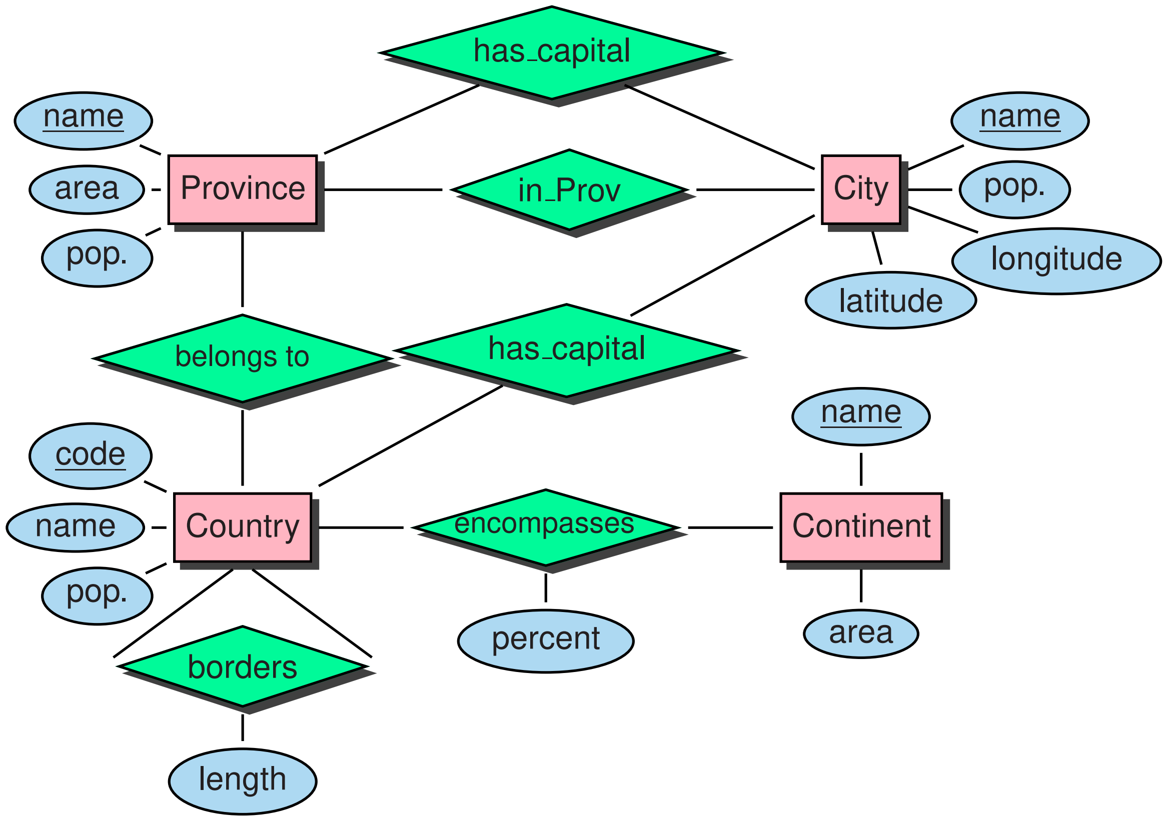

relationship with attributes




relationship with roles



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 all amongst all entities of a given type (cf. *candidate keys, primary keys*).

**Relationship Complexities:** every relationship type is assigned a complexity 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 all 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 $\mu_1$ and $\mu_2$ 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.

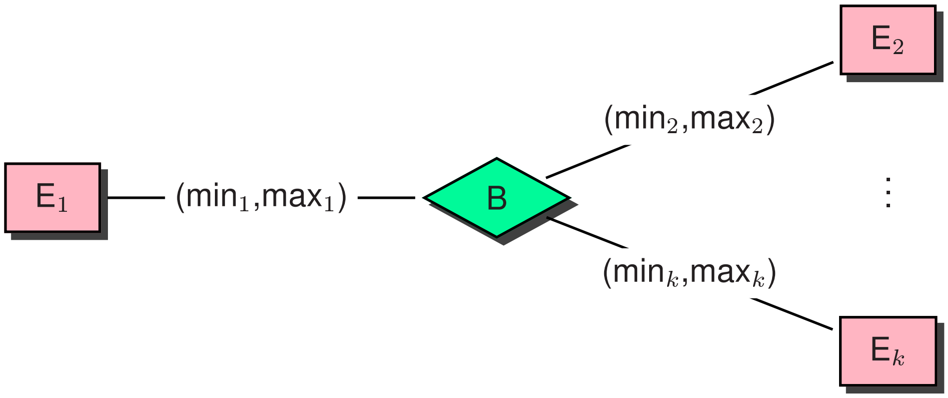
Graphically, key attributes are distinguished by underlining.

## Relationship Complexities

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

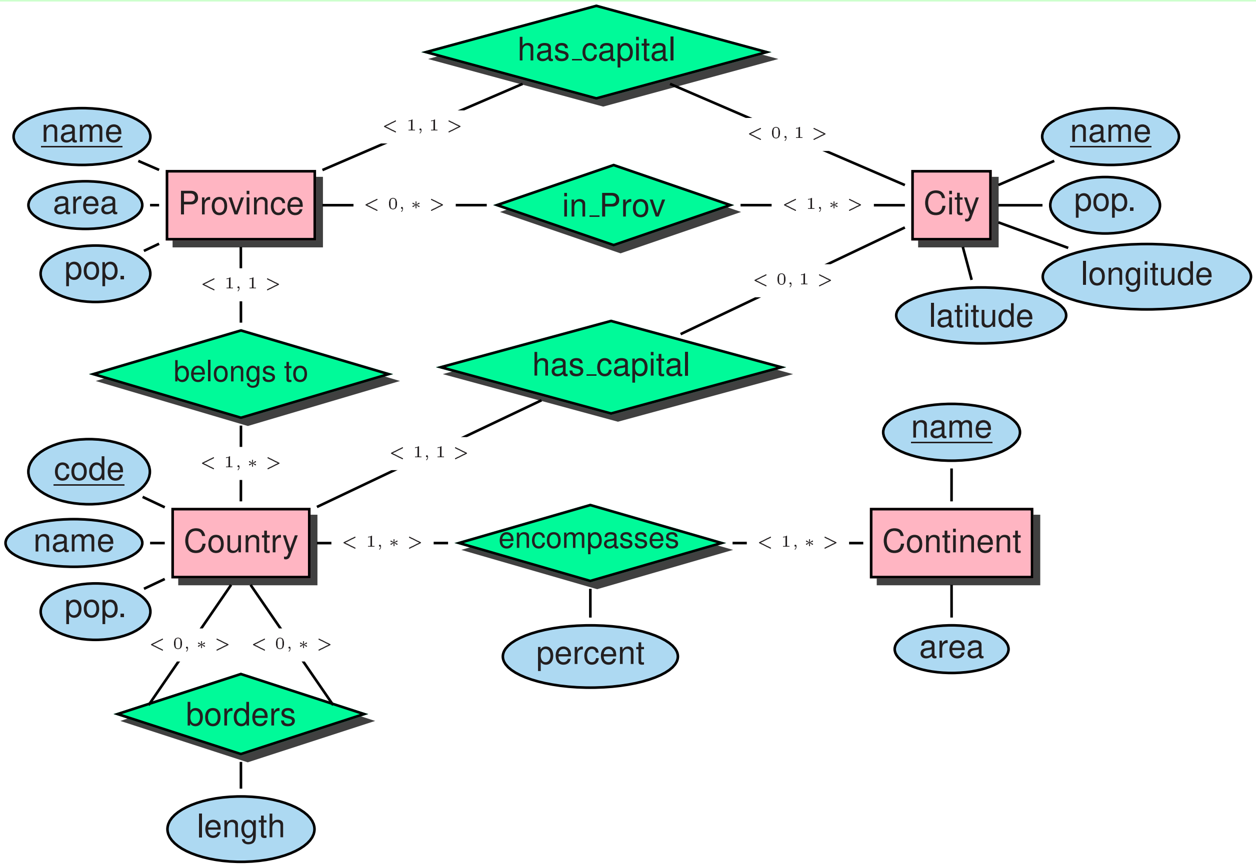
The **complexity degree** of a relationship type $B$ wrt. one of its roles $RO$ is an expression of the form $(min, max)$ where $0 \leq min \leq max$, and $max = *$ means "arbitrary many".

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






## Example: ER Model of a geographical database



## Additional Notions for Complexity Degrees

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

- if $max_1 = max_2 = 1$, it is called a $1:1$-relationship.
  is_capital(country: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(country:city) is a 1:n-relationship
- Otherwise, it is called an $n:m$-relationship.
  borders(country:country) is an n:m-relationship



## 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 $\mu$ 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

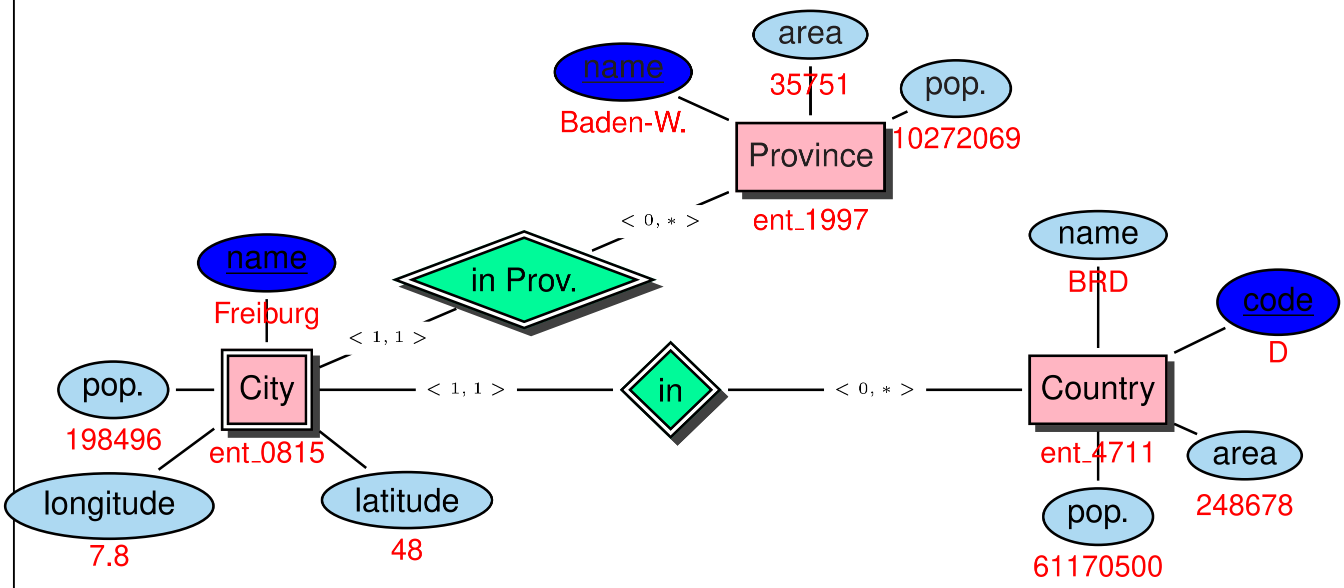
A weak entity type is an entity type without a key.

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 must 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*).
- 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, LowerSaxonia (Niedersachsen)

## Extensions of the ERM: Attributes

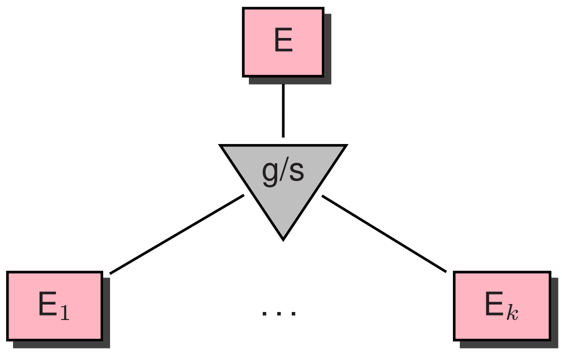
Attributes can be

- set-valued or multi-valued,
- structured




## Extensions of the ERM: Generalization/Specialization

Abstractions that allow for grouping entities of different, but related types to a more general type.

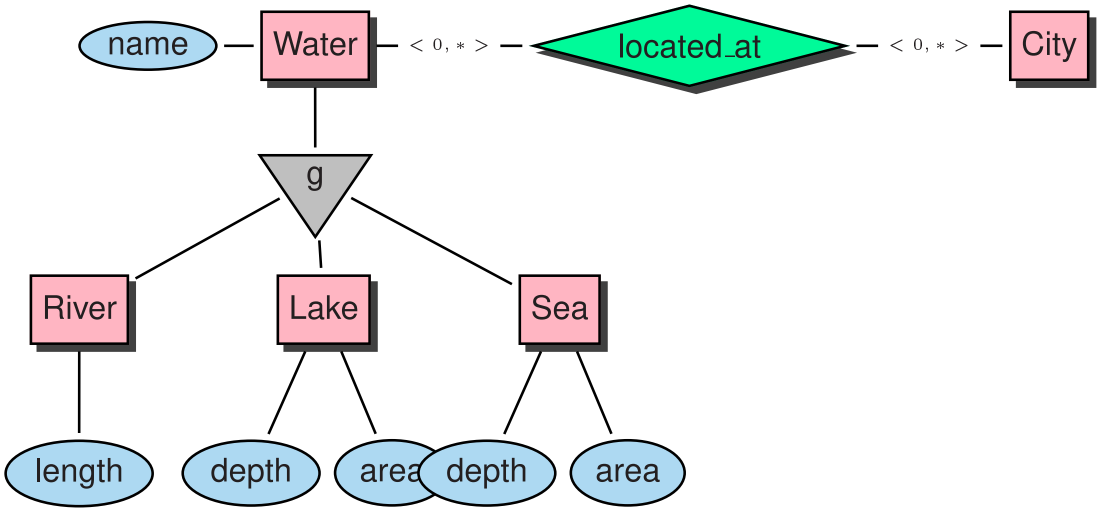



$E$ is called **supertype**, $E_i$ are **subtypes** for $1 \leq i \leq k$. Each entity of a subtype s also an entity of the supertype.

- 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

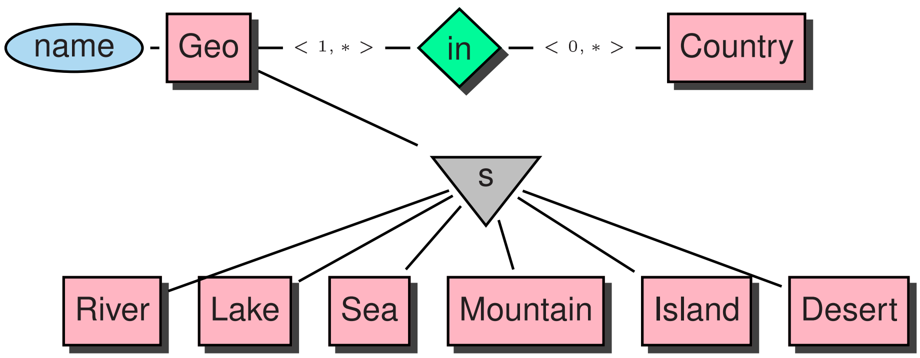
- Generalization: rivers, lakes, and seas are *waters*. These can e.g. be involved in *located-at* relationships with cities:






## GENERALIZATION/SPECIALIZATION

- Specialization: MONDIAL does not describe all geographical things, but only rivers, lakes, seas, mountains, deserts, and islands (no lowlands, highlands, savannas, fens, etc). All such geographical things have in common that they are involved in *in*-relationships with countries:



# GENERALIZATION/SPECIALIZATION

### Integrity Constraints

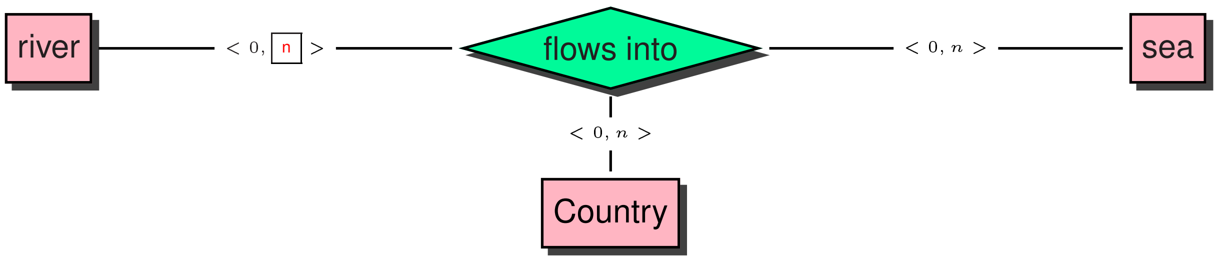
- Common integrity constraints **ISA:** ISA is satisfied in a database state if the entity sets of the subtypes are subsets of the entity sets of the supertype,
- optional integrity constraint **Disjointness:** if the entity sets of the subtypes are disjoint,
- optional integrity constraint **Covering:**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



# 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 is overcome by defining artificial entity types.

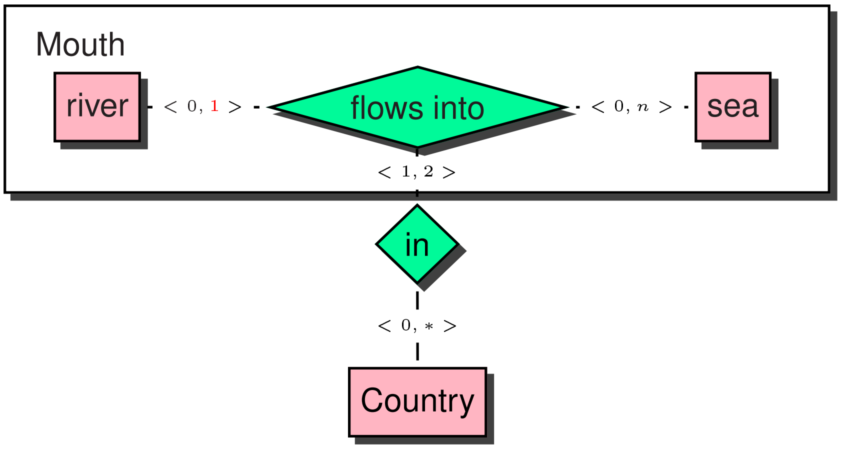
A river flows into a sea/lake/river; more detailed, this point can be described by giving one or two countries.




This representation is ambiguous: A river could flow in two seas!?

## Aggregation

Using aggregation, this information can be specified much clearer by introducing an *aggregate type mouth*:




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



## 2.1.4 Discussion ERM

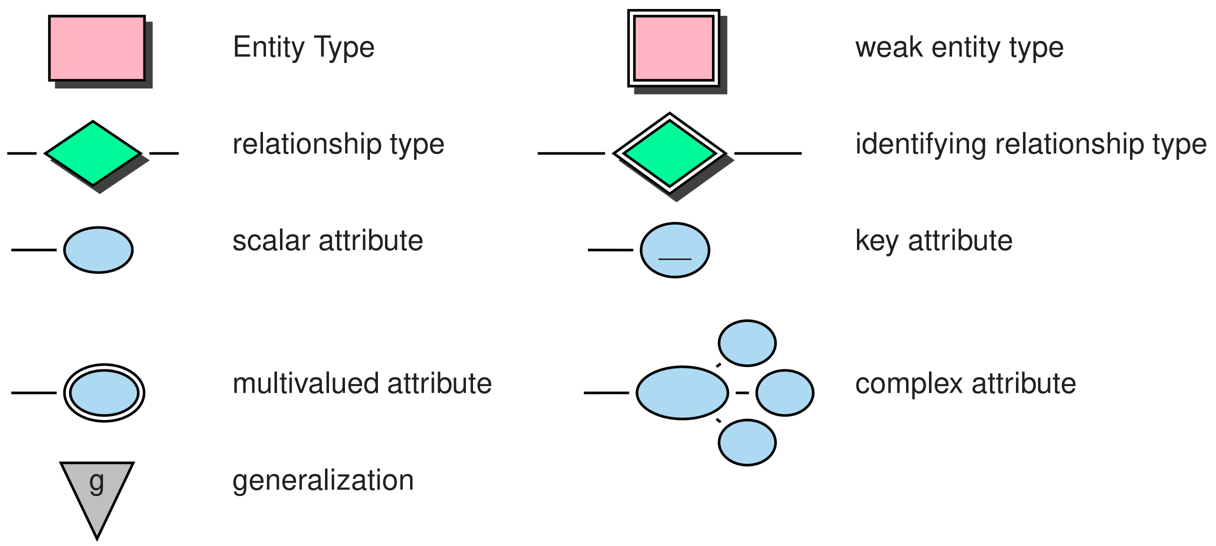
- With the structuring concepts of the ERM and its extensions, the *static* aspects of a relevant excerpt of the real world can be modeled semantically adequate in a natural way.
- The graphical representation is also understandable for non-computer-scientists.
- The ERM is useful
  - in the early stages 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 recently also by XML-based DBMS).

## Discussion ERM (cont'd)

- There is a more complex and more expressible 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 complexities $(0, 1)$ and $(1, *)$. Investigate all possible ways how to assign these relationship complexities to the relationship type. For each variant, give a nontrivial state that satisfies them, and a state that violates them.* □

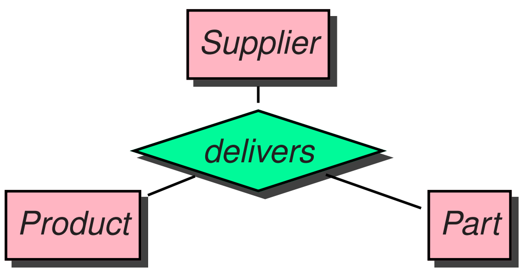
**Exercise 2.2**
*Discuss ER schemata for the following scenario:*

- *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 complexities?*
- *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. □

# 3-Level Architecture of a DBMS (ANSI/SPARC)