



**ifis**

Institut für Informationssysteme  
Technische Universität Braunschweig

# Relational Database Systems I

**Wolf-Tilo Balke**

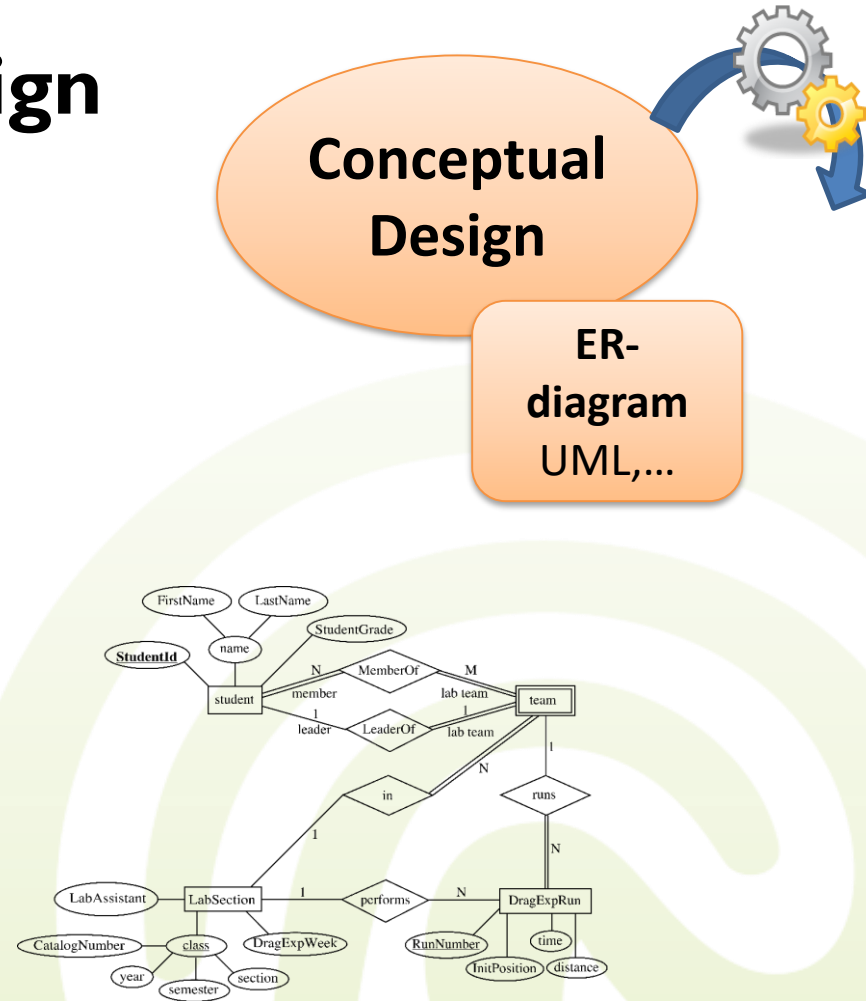
**Niklas Kiehne, Enrique Pinto Dominguez**

Institut für Informationssysteme  
Technische Universität Braunschweig  
<http://www.ifis.cs.tu-bs.de>



# 2 Data Modeling I

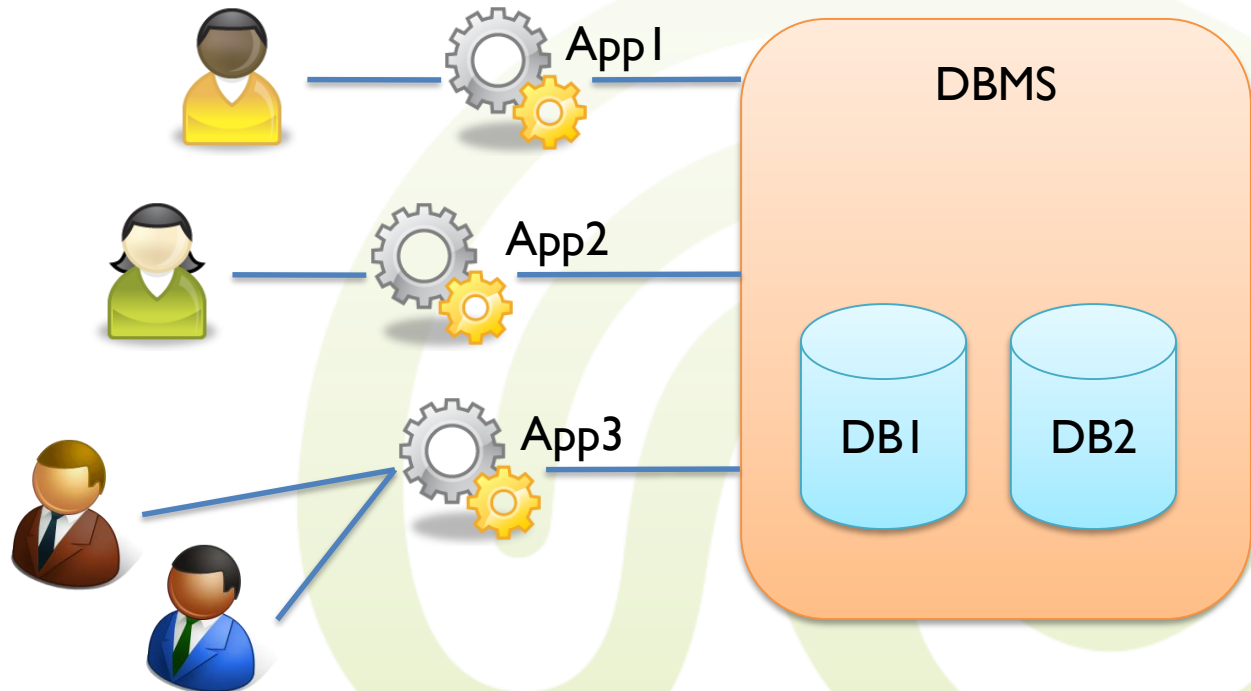
- **Phases of DB Design**
- Data Models
- Basic ER Modeling
  - Chen Notation
  - Mathematical Model
- Example





# 2.1 Database Applications

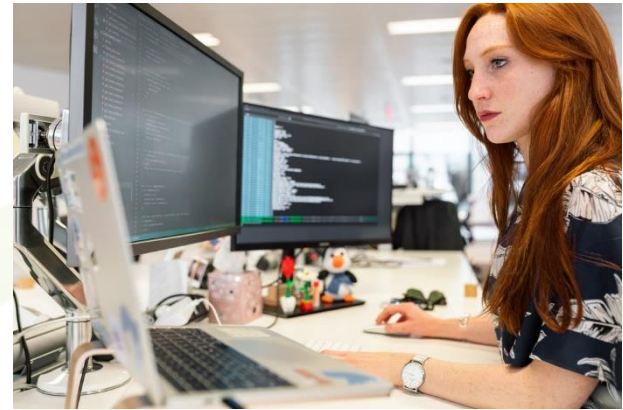
- **Database applications** consist of
  - **database instances** with their respective **DBMS**
  - associated **application programs** interfacing with the users





## 2.1 Database Applications

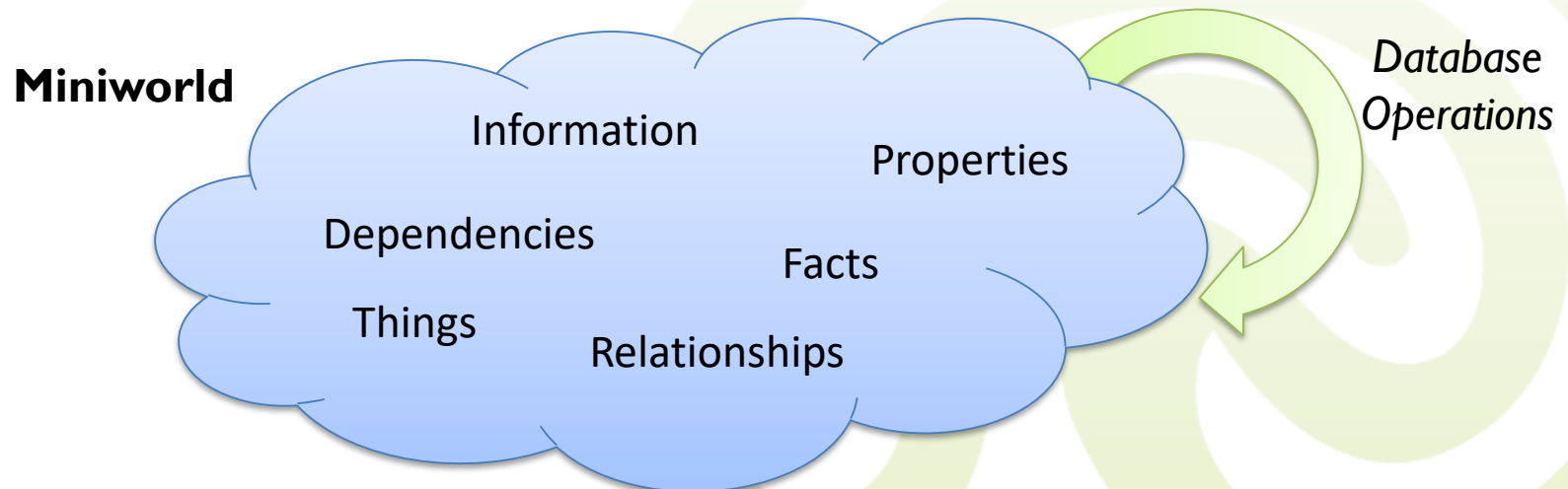
- Planning and developing application programs traditionally is a **software engineering** problem
  - Requirements Engineering
  - Conceptual Design
  - Application Design
  - ...
- Software engineers and **data engineers** cooperate tightly in planning the need, use and flow of data
  - **Data Modeling**
  - **Database Design**





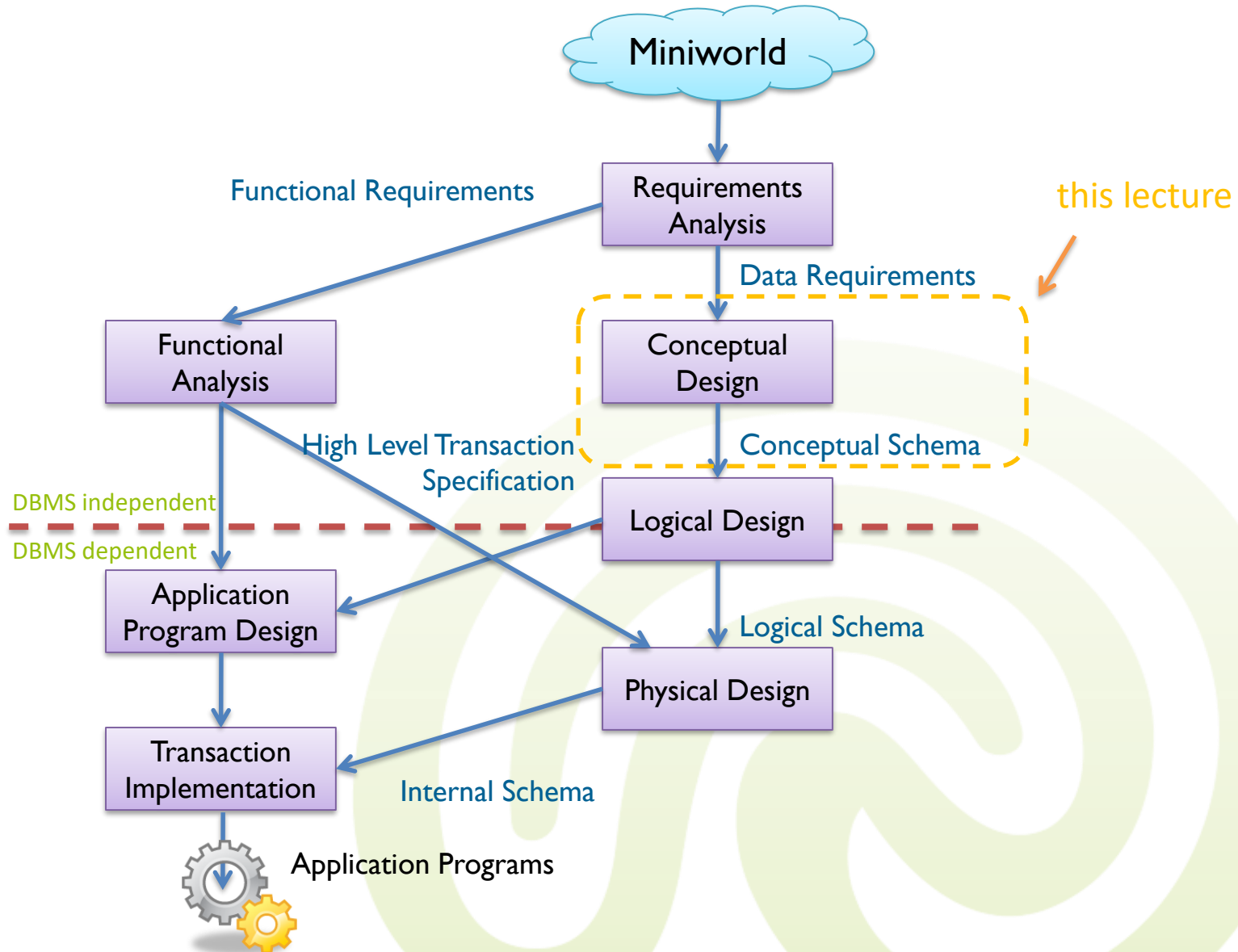
## 2.1 Universe of Discourse

- DB Design models a **miniworld** (also called universe of discourse) into a formal representation
  - restricted view on the real world with respect to the problems that the current application should solve





# 2.1 Phases of DB Design





## 2.1 Phases of DB Design

- **Requirements Analysis**
  - database designers interview prospective **users** and **stakeholders**
  - **Data Requirements** describe what kind of data is needed
  - **Functional Requirements** describe the operations performed on the data
- **Functional Analysis**
  - concentrates on describing **high-level** user operations and transactions
    - does not yet contain implementation details



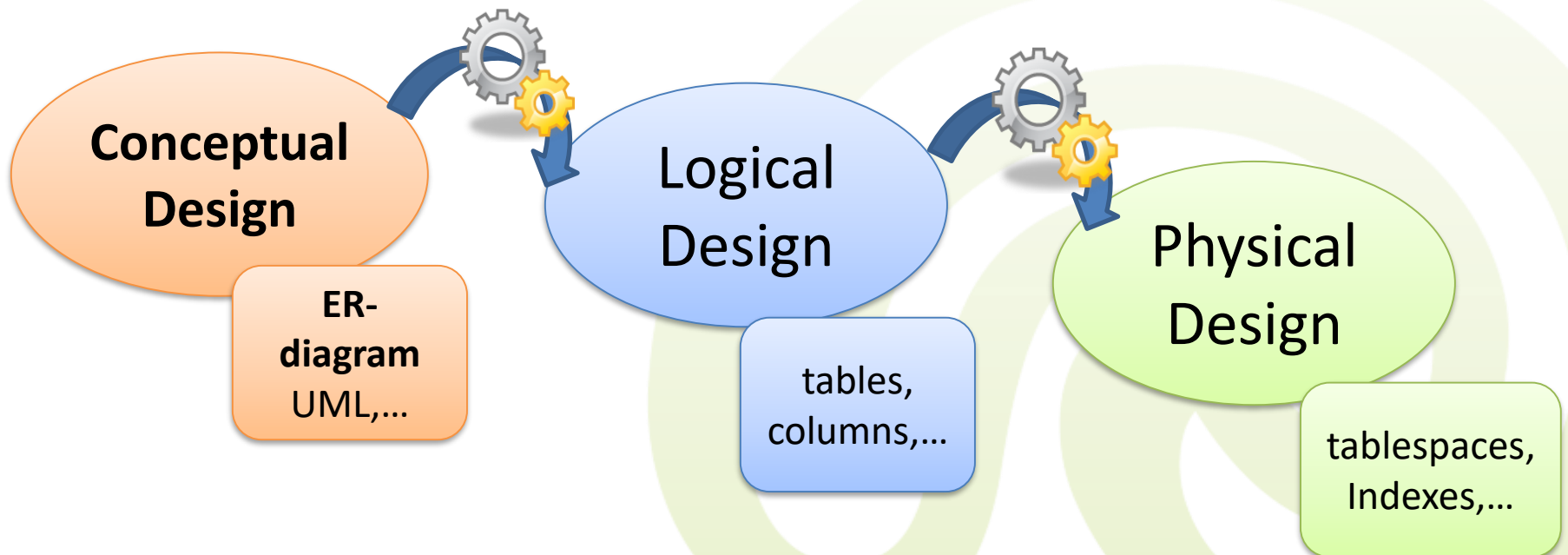
# 2.1 Phases of DB Design

- **Conceptual Design**
  - transforms Data Requirements to **conceptual model**
  - describes high-level data entities, relationships, constraints, etc.
    - does not contain any implementation details
    - independent of used software and hardware
    - only loosely depending on chosen data model
- **Logical Design**
  - maps the conceptual data model to the logical data model used by the DBMS
    - e.g. relational model, hierarchical model
    - technology independent conceptual model is adapted to the used DBMS software
- **Physical Design**
  - creates internal structures needed to efficiently store/manage data
    - e.g. table spaces, indexes, access paths
    - depends on used hardware and DBMS software



## 2.1 Conceptual Design

- Modeling the data involves three design phases
  - result of one phase is input of the next phase
  - often, automatic transition is possible with some additional designer feedback





# 2 Data Modeling I

- Phases of DB Design
- **Data Models**
- Basic ER Modeling
  - Chen Notation
  - Mathematical Model
- Example





## 2.2 Data Semantics

- In databases, the data's specific **semantics** are very important
  - what is described?
  - what values are reasonable/correct?
  - what data belongs together?
  - what data is often/rarely accessed?





## 2.2 Data Semantics

- Example: Describe the *age* of a person
  - semantic definition:  
*The number of years elapsed since a person's birthday.*
  - integer data type
  - always:  $0 \leq \text{age} \leq 150$
  - connected to the person's name, passport id, etc.
  - may often be retrieved, but should be protected
  - ...





## 2.2 Data Models

- A **data model** is an abstract model that describes how data is represented, accessed, and reasoned about
  - e.g. network model, relational model, object-oriented model
  - **warning:** The term *data model* is ambiguous
    - a data model **theory** is a formal description of how data may be structured and accessed, and is independent of a specific software or hardware
    - a data model **instance** or **schema** applies a data model theory to create an instance for some particular application (e.g., data models in MySQL Workbench designer refer to a logical model adapted to the MySQL database)



## 2.2 Data Models

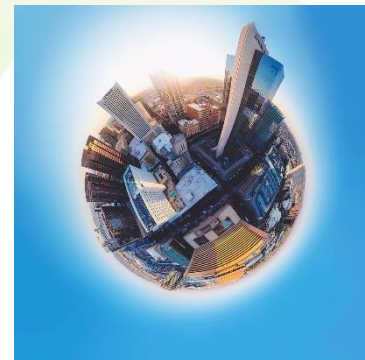
- A **data model** consists of three parts
  - Structure
    - **data structures** are used to create databases representing the modeled objects
  - Integrity
    - rules expressing the **constraints** placed on these data structures to ensure structural integrity
  - Manipulation
    - operators that can be applied to the data structures, to **update** and **query** the data contained in the database





## 2.2 Generic Data Models

- **Generic data models** are generalizations of conventional data models
  - definition of standardized general relation types, together with the kinds of things that may be related by such a relation type
  - Think of: “Pseudocode data model”
    - Simple description of the data requirements of the miniworld independent of formal data model





## 2.2 Generic Data Models

- **Example:** A generic data model may define relation types for describing structures, such as
  - **classification relation** – as a binary relation between an individual thing and a kind of thing (i.e. a *class*)
    - e.g. Dolphin *is\_a* Animal, Cat *is\_a* Animal  
is\_a: (Dolphin, Animal), (Cat, Animal), (Snowball, Cat)
  - **part-whole relation** – as a binary relation between two things: one with the part role and the other with the whole role
    - e.g. Wheel *is\_part\_of* Car, Branch *is\_part\_of* Tree  
is\_part\_of: (Wheel, Car), (Branch, Tree)



## 2.2 Data Models

- Different categories of formal data models exist
  - **conceptual** data models (**high-level**)
    - represent structure in a way that is close to the users' perception of data
      - e.g., the relational model, network models, etc.
  - **representational** or **logical** data models
    - represent structure in a way that is still perceivable for users but that is also close to the physical organization of data on the computer
  - **physical** data models (**low-level**)
    - represent structure that describe the details of how data is stored from the computer



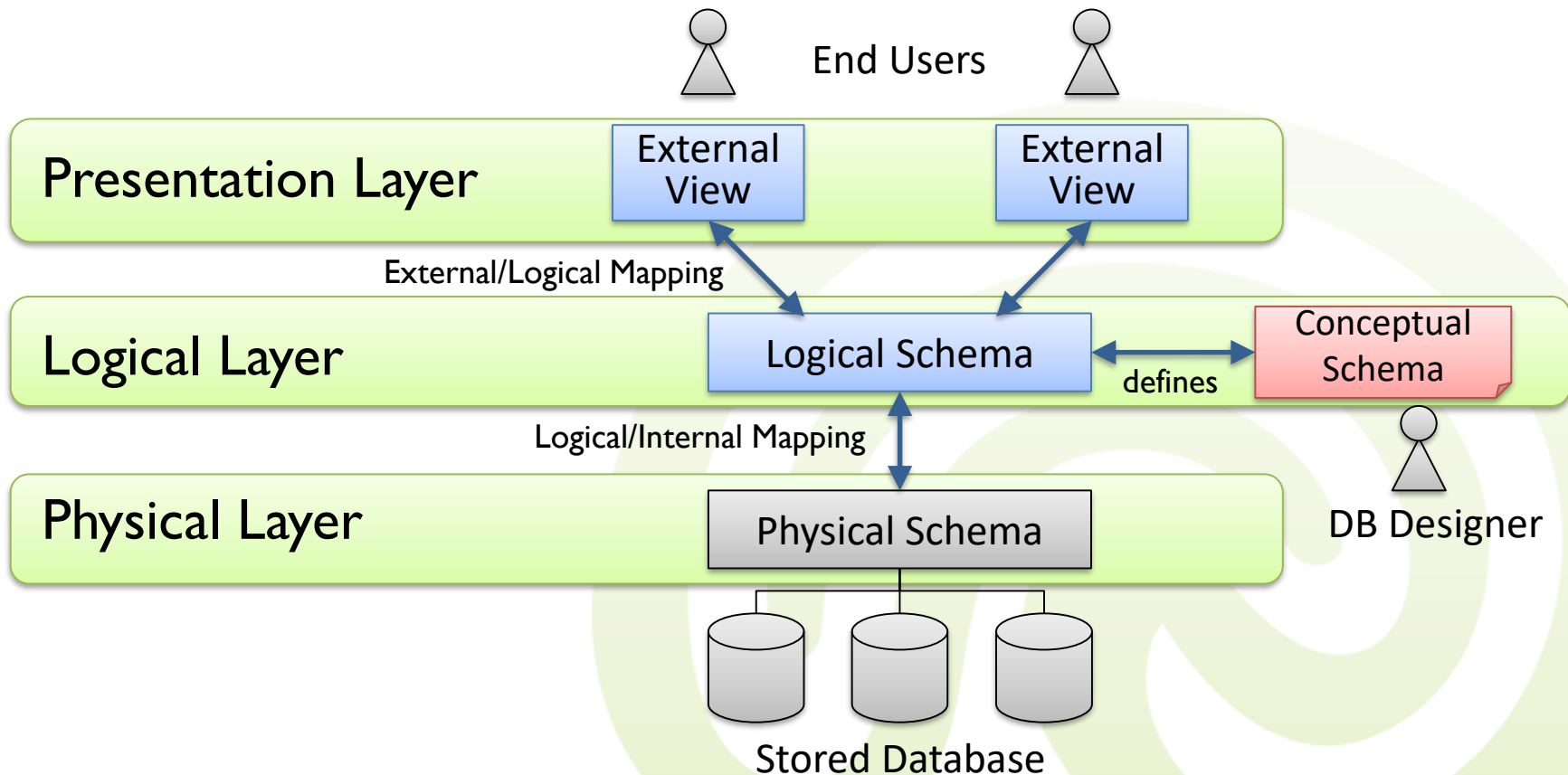
## 2.2 Data Models

- Concrete instances of data models are called **schemas**
  - a **conceptual schema** describes the data semantics of a certain domain
    - what facts or propositions hold in this domain?
  - a **logical schema** describes the data semantics, as needed by a particular data manipulation technology
    - e.g. tables and columns, object-oriented classes, XML elements
  - a **physical schema** describes the physical means by which the data is stored
    - e.g. partitions, tablespaces, indexes



## 2.2 Three-layer Architecture

- Example: Three-layer Architecture
  - Also called ANSI-SPARC Architecture





## 2.2 Three-layer Architecture

- ANSI-SPARC Architecture
  - Careful: A lot of ambiguous naming is going on!
  - the **logical layer** is often referred to as the **conceptual layer**
    - usually **logical** or **representational** data model
      - e.g., lower level ER schemas
    - but often based on a **conceptual schema design** in a high-level data model
      - e.g., high level Extended ER schemas
  - **external views**
    - typically implemented using a **logical** data model
    - but often based on a **conceptual schema design** in a high-level data model



## 2.2 Three-layer Architecture

- Why do we need layers?
  - they provide **independence**
  - **physical independence**
    - storage design can be altered without affecting logical or conceptual schemas
    - e.g. regardless on which hard drive a person's age is stored, it remains the same data
  - **logical independence**
    - logical design can be altered without affecting the data semantics
    - e.g. it does not matter whether a person's age is directly stored or computed from the person's birth date



## 2.2 Data Models

- Which data model do we want to use?
  - Conceptual Model: **Entity-Type-Centric Approach**
    - Model the miniworld entity types, their properties, and relationships
  - Logical Model: **Relational Model**
    - Analogy: *Index cards*
      - Similarly **structured** index cards for the same entity type
      - All data (properties, relationships to other cards) about a single entity on a single card
      - Each single card can be uniquely identified by (a subset) of its properties
      - “**What do we want to write on our index cards?**”



## 2.2 Data Models

### – Physical Model:

- How do we want to store and access our logical model physically?
- *Index card analogy:*
  - **How do we write the content on our index cards?**
  - **How do we organize or sort our cards?**
  - Are there additional indexes next to the box?
  - Do we use a simple box, or a fancy card flywheel?





# 2 Data Modeling I

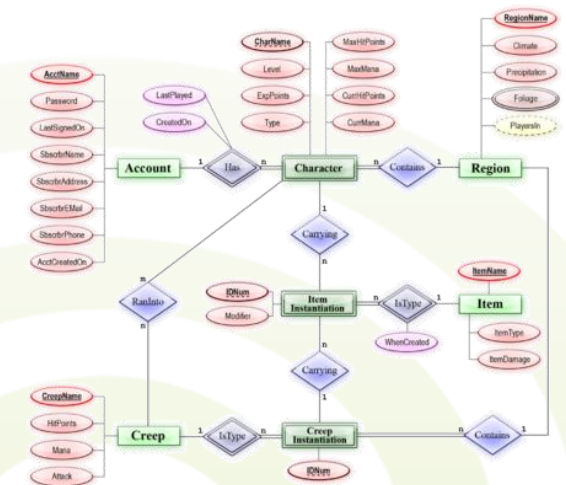
- Phases of DB Design
- Data Models
- **Basic ER Modeling**
  - Chen Notation
  - Mathematical Model
- Example





## 2.3 ER Modeling

- Traditional approach to **Conceptual Modeling**
  - **Entity-Relationship Models (ER-Models)**
    - also known as Entity-Relationship Diagrams (ERD)
    - introduced in 1976 by **Peter Chen**
    - graphical representation
- **Top-Down-Approach** for modeling
  - entities and attributes
  - relationships
  - constraints
- Some derivatives became popular
  - ER Crow's Foot Notation (Bachman Notation)
  - ER Baker Notation
  - later: Unified Modeling Language (UML)

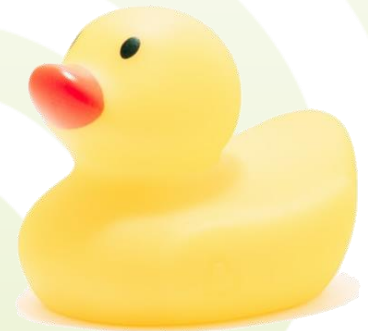
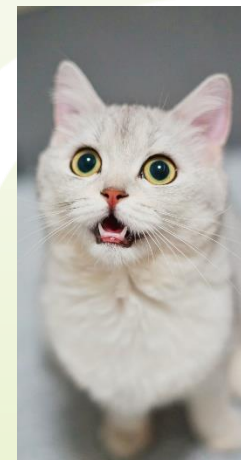




## 2.3 ER – Entities

- **Entities**

- an entity represents a *thing* in the real world with an independent existence
  - an entity has an own identity and represents just one thing
- e.g. *a car, a savings account, my neighbor's house, the cat Snowflake, a product*





## 2.3 ER – Attributes

- **Attributes**
  - a **property** of an entity, entity type or a relationship type
  - e.g. *name* of an employee, *color* of a car, *balance* of an account, *location* of a house
  - attributes can be classified as being:
    - **simple** or **composite**
    - **single-valued** or **multi-valued**
    - **stored** or **derived**
    - e.g. name of a cat is simple, single-valued, and stored



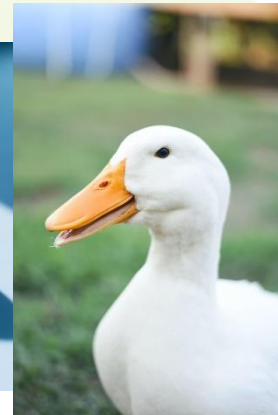
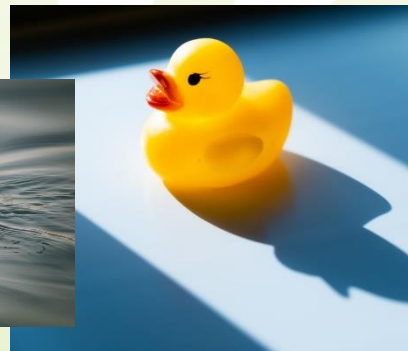
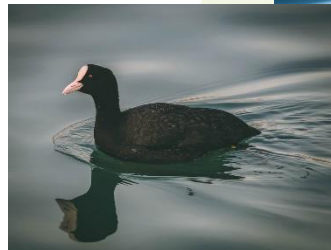
## 2.3 ER – Entity Types

- **Entity types**
  - sets of entities sharing the same characteristics or attributes
    - each entity within the set has its own attribute values
  - each entity type is described by its name and attributes
    - each entity is an **instance** of an entity type
  - describes the so called **schema** or **intension** of a set of similar entities



## 2.3 ER – Entity Sets

- **Entity Set** (*of a given entity type*)
  - collection of all stored entities of a given entity type
  - entity sets often have the same name as the entity type
    - *Duck* may refer to the entity type as well as to the set of all *Duck* entities (sometimes also plural for the set: *Ducks*)
  - also called the **extension** of an entity type (or **instance**)

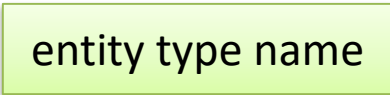




## 2.3 ER Diagrams

- ER diagrams represent **entity types** and **relationships** among them, not single entities
- Graphical Representation

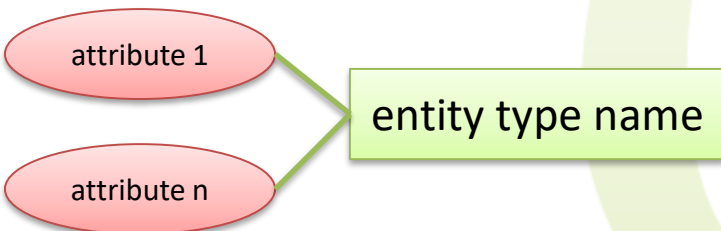
### – entity type



entity type name

- Rectangle labeled with the name of the entity
- Usually, name starts with capital letters

### – attributes



attribute 1

attribute n

entity type name

- Oval labeled with the name of the attribute
- Usually, name starts with lower case letters



## 2.3 ER Diagrams

- Textual Representation
  - entity types
    - written: `entity_type_name(attribute_1, ..., attribute_n)`
  - entity
    - written: `(value of attribute_1, ..., value of attribute_n)`

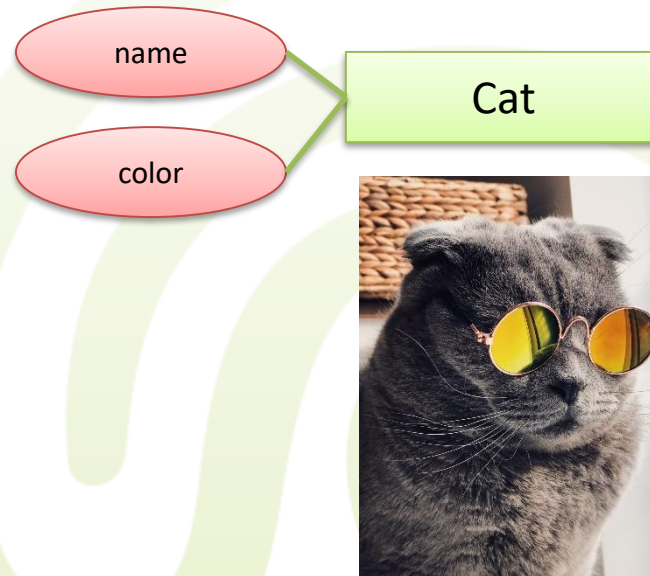
- Example

- **Entity Type** *Cat*

- `Cat(name, color)`

- **Entity Set** *Cats*

- `(Fluffy, black-white)`
    - `(Snowflake, white)`
    - `(Captain Hook, red)`
    - `(Garfield, orange)`





## 2.3 ER – Composite Attributes

- **Simple Attribute:**

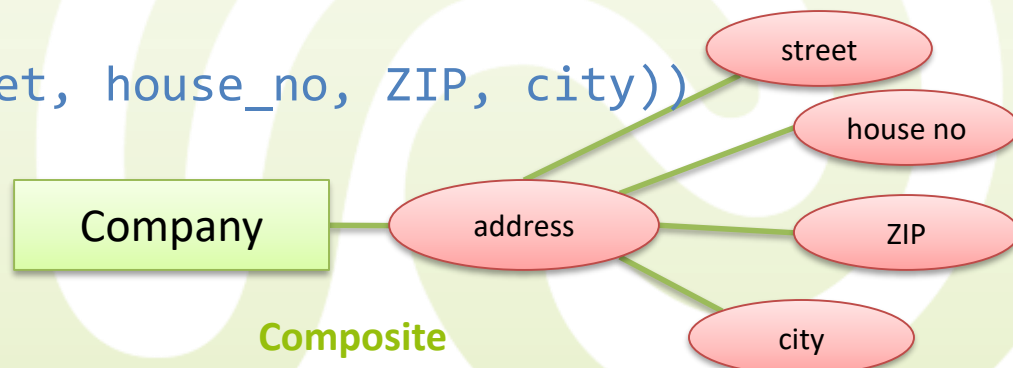
- attribute composed of a single component with an independent existence
- e.g. *name* of a cat, *salary* of an employee
  - `Cat(name)`, `Employee(salary)`

- **Composite Attribute:**

- Attribute composed of multiple components, each with an independent existence
  - graphically represented by connecting sub-attributes to main attribute
  - textually represented by grouping sub-attributes in ()
- e.g. *address* attribute of a company (is composed of *street*, *house number*, *ZIP*, and *city*)
  - `Company(address(street, house_no, ZIP, city))`



Simple



Composite



## 2.3 ER Multi-Valued Attributes

- **Single-Valued Attribute**
  - attribute holding a **single** value for each occurrence of an entity type
  - e.g. *name* of a cat, *registration number* of a student
- **Multi-Valued Attributes (lists)**
  - attribute holding (possibly) **multiple** values for each occurrence of an entity type.
    - graphically indicated by a double-bordered oval
    - textually represented by enclosing in {}
  - e.g. *telephone number* of a student
    - `Student({telephone_no})`
  - Careful here: Do you really want to model something as an multi-value attribute? Or should it be an own entity type instead?
    - For a student, are phone numbers a good multi-valued attribute? Are courses of studies good multi-valued attributes?



Single Valued



Multi-Valued



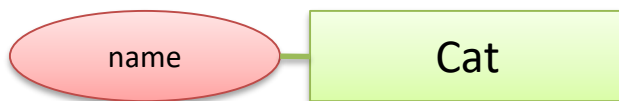
## 2.3 ER – Derived Attributes

- **Stored Attribute**

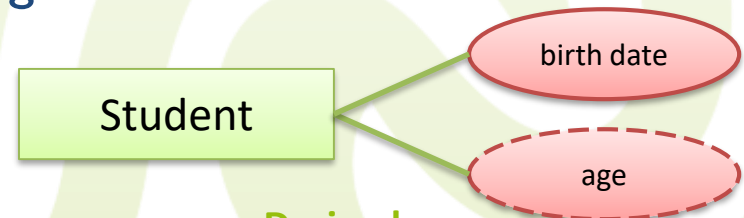
- the attribute is directly stored in the database

- **Derived Attribute**

- the attribute is (usually) not stored in the DB but derived from an other, stored attribute
  - On a logical schema, it's a design decision if an attribute should really be derived or stored (redundantly)
  - Redundant storage might lead to better performance, but requires dealing with consistency of updates
- indicated by dashed oval
- e.g. *age* can be derived from *birth date*, *average grade* can be derived by aggregating all stored *grades*



Stored



Derived



## 2.3 ER – Keys

- Entities are **only** described by attribute values
  - two entities with identical values cannot be distinguished
    - Later, we might introduce OIDs, row IDs, etc. to fix this problem in a logical schema
- Entities (usually) must be distinguishable
- Identification of entities with **key attributes**
  - value combination of key attributes is **unique** within **all possible extensions** of the entity types
  - key attributes are indicated by underlining the attribute name



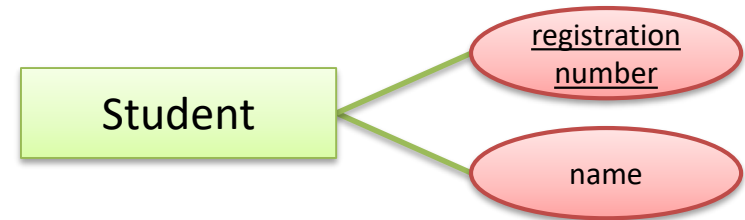


## 2.3 ER – Keys

- Key attribute examples

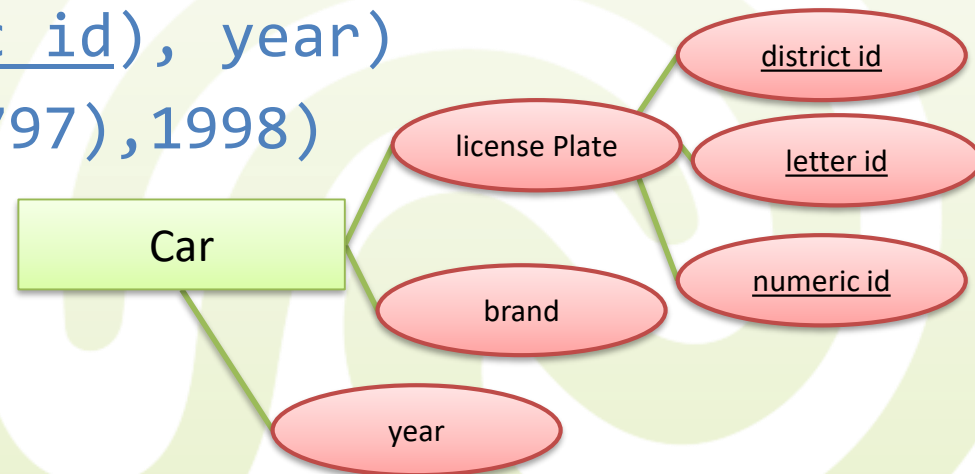
- single key attribute

- Student(registration number, name)
- (432451, Hans Müller)



- composite key (multiple key attributes)

- Car(brand, license\_plate(district id, letter id, numeric id), year)
- (Mercedes, (BS, CL, 797), 1998)
- please note that each key attribute itself does not need to be unique!

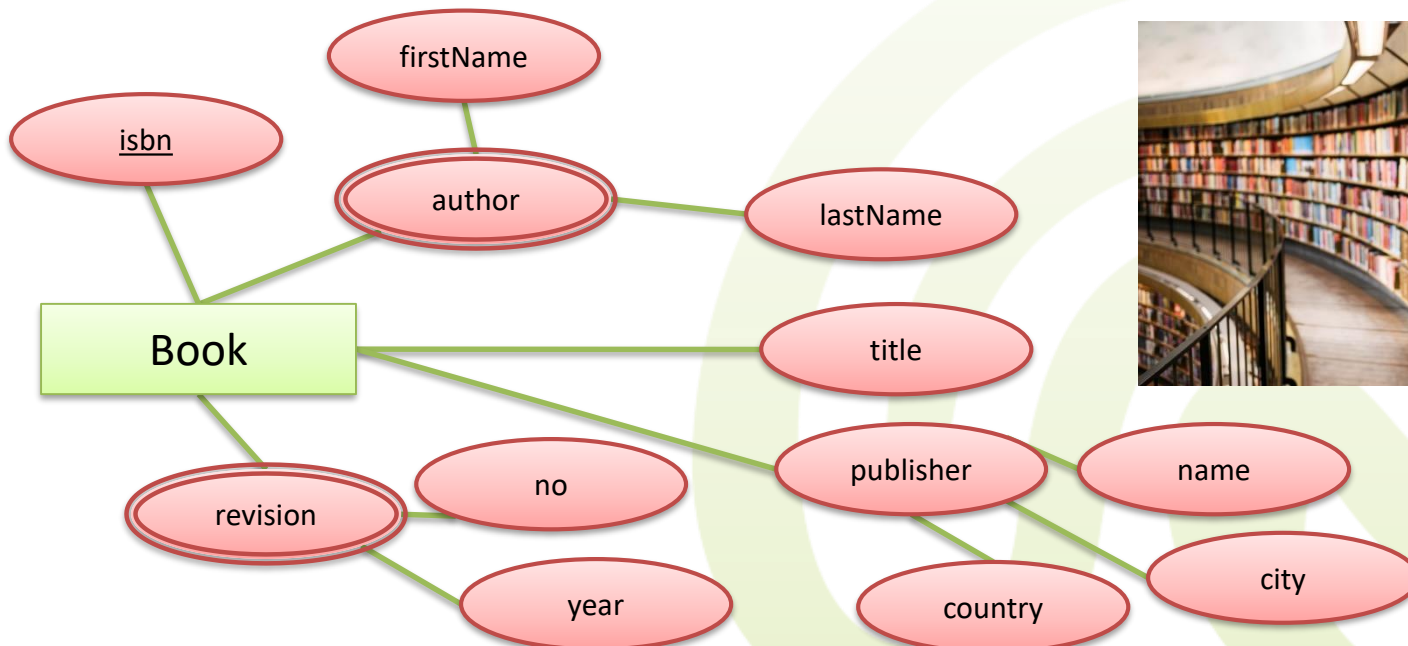




## 2.3 ER Modeling

- **Sample Entity Type**

- `Book(isbn, {author(firstName, lastName)}, title, publisher(name, city, country), {revision(no, year)})`
- `(0321204484, {(Ramez, Elmasri), (Shamkant, Navathe)}, Fundamentals of Database Systems, (Pearson, Boston, US), {(4, 2004), (2, 1994)})`

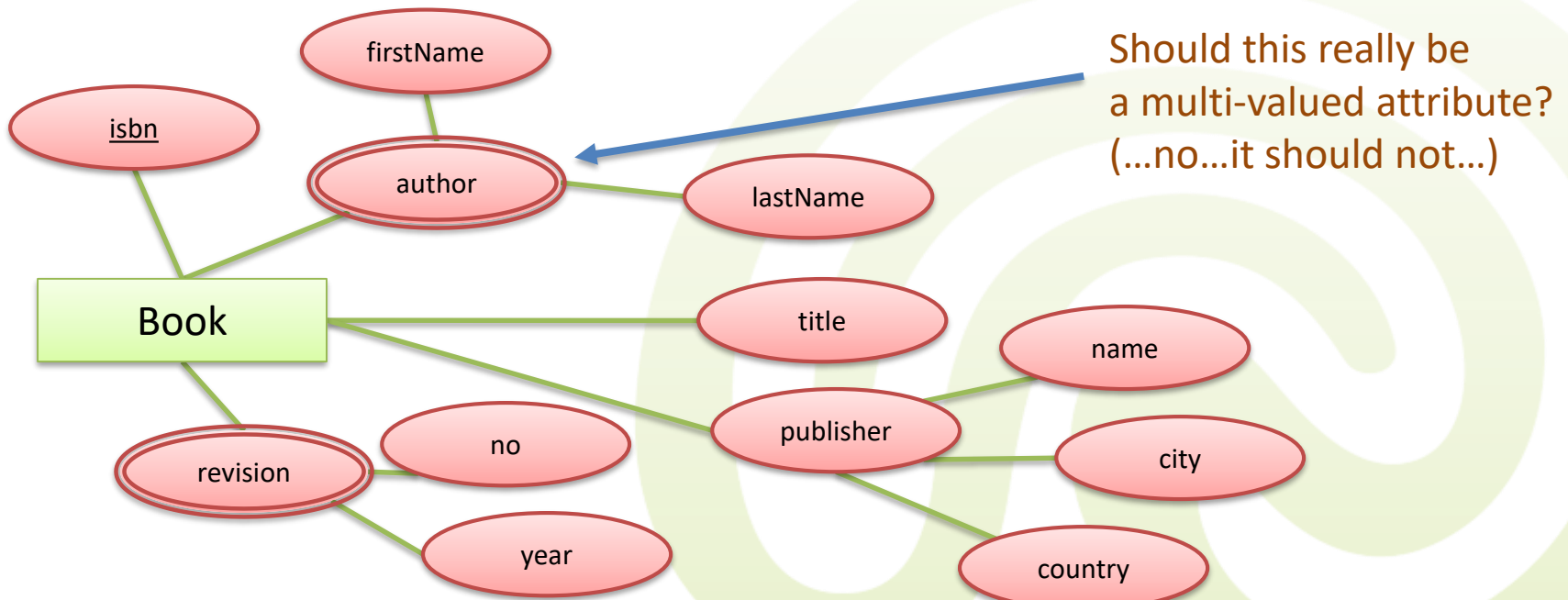




## 2.3 ER Modeling

- **Sample Entity Type**

- `Book(isbn, {author(firstName, lastName)}, title, publisher(name, city, country), {revision(no, year)})`
- `(0321204484, {(Ramez, Elmasri), (Shamkant, Navathe)}, Fundamentals of Database Systems, (Pearson, Boston, US), {(4, 2004), (2, 1994)})`





## 2.3 ER – Domains

- Attributes cannot have arbitrary values: they are restricted by the attribute **value sets (domains)**
  - *zip codes* may be restricted to integer values between 0 and 99999
  - *names* may be restricted to character strings with maximum length of 120
  - domains are not displayed in ER diagrams
  - usually, popular **data types** are used to describe domains in data modeling
    - e.g. integer, float, string



## 2.3 ER – Domains

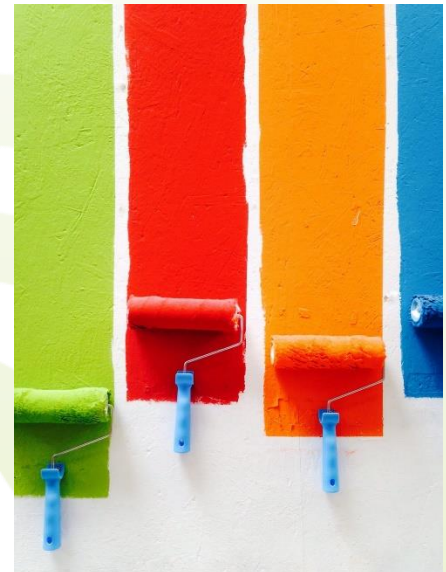
- Commonly used data types

Name	Syntax	description
integer	integer	32/64-Bit signed integer values between $-2^{31/64}$ and $2^{31/64}$
double	double	64-Bit floating point values of approximate precision
numeric	numeric( $p, s$ )	A number with $p$ digit before the decimal and $s$ digitals after the decimal (exact precision)
character	char( $x$ )	A textual string of the exact length $x$
varying character	varchar( $x$ )	A textual string of the maximum length $x$
date	date	Stores year, month, and day
time	time	Stores hour, minute, and second values



## 2.3 ER – Domains

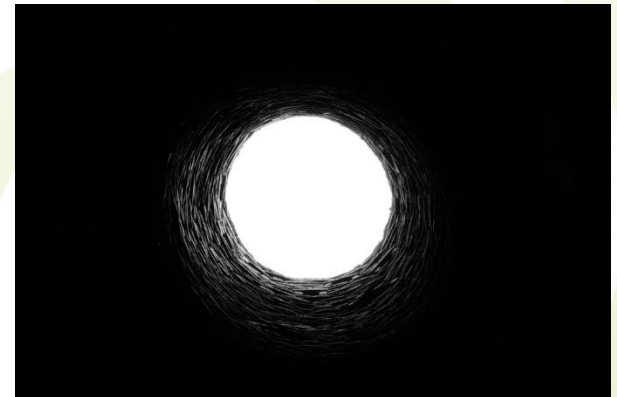
- Using data types for modeling domains is actually a crutch
  - Some modern programming language are better in this way!
  - the original intention of domains was modeling all valid values for an attribute
    - `color: {Red, Blue, Green, Yellow}`
  - using data types is very coarse and more a convenient solution
    - `color: varchar(6) ???`
  - to compensate for the lacking precision, often **restrictions** are used
    - `color: varchar(6) restricted to {Red, Blue, Green, Yellow}`





## 2.3 ER – NULL Values

- Sometimes, an attribute value is **not known** or an attribute does **not apply** for an entity
  - this is denoted by the special value **NULL**
    - so called **NULL-value**
  - e.g. attribute *university\_degree* of Entity *Heinz Müller* may be NULL, if he does not have a degree
  - NULL is usually always allowed for any domain or data type unless explicitly excluded





## 2.3 ER – NULL Values

- What does it mean when you encounter a NULL-value?
  - attribute is not applicable
    - e.g. attribute *maiden name* when you don't have one
  - value is not known
  - value will be filled in later
  - value is not important for the current entity
  - value was just forgotten to set
- Actually there are more than 30 possible interpretations...





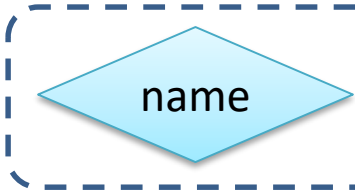
## 2.3 ER – Relationships

- Entities are not enough to model a miniworld
  - the power to model dependencies and relationships is needed
- In ER, there can be **relationships** between entities
  - each relationship **instance** has a **degree**
    - i.e. the number of entities it relates to
  - a relationship instance may have attributes





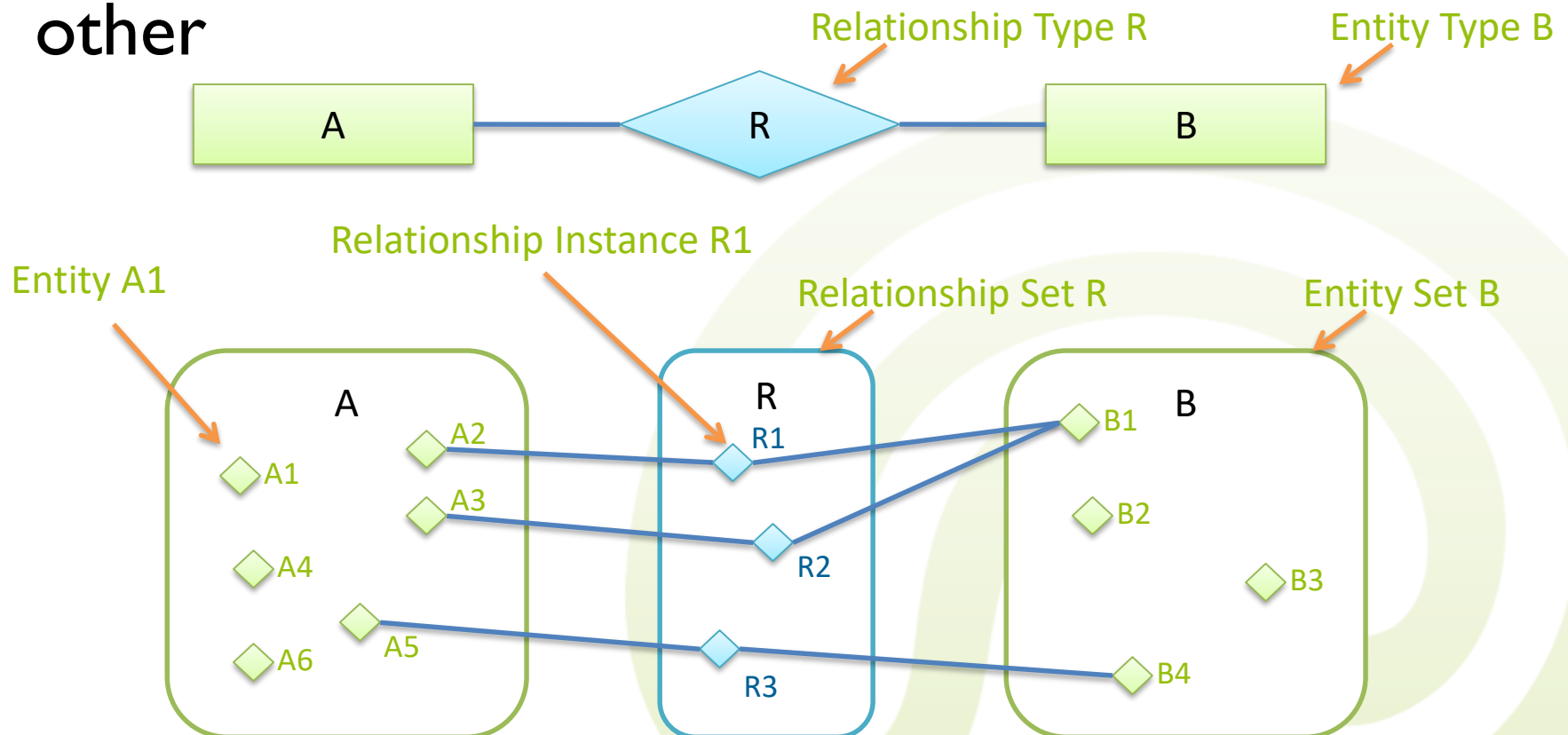
## 2.3 ER – Relationships

- Similar to entities, ERDs do not model individual relationships, but **relationship types**
  - **Relationship type**
    - named set of all similar relationships with the same attributes and relating to the same entity types
- 
  - Diamond labeled with the name of the relationship type
  - Usually, name starts with lower-case letters
- **Relationship set**
    - set of all **relationship instances** of a certain relationship type



## 2.3 ER – Relationships

- **Relationships** relate **entities** within the **entity sets** involved in the **relationship type** to each other





## 2.3 ER – Relationships

- **Example:**
  - there is an *ownership* relation between cats and persons

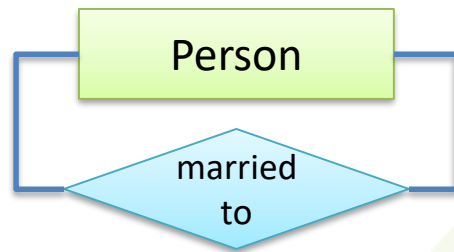


- but more modeling detail is needed
  - does every person own a cat? Does every cat have an owner?
  - can a cat have multiple owners or a person own multiple cats?
  - since when does a person own some cat?
  - who owns whom?



## 2.3 ER – Relationship Cardinality

- Additionally, **restrictions** on the combinations of entities participating in an entity set are needed
  - e.g. relationship type *married to*



- unless living in Utah, a restriction should be modeled that each person can only be married to a single person at a time
  - i.e. each person entity may only appear once in the “married to” relationship set
- **cardinality** annotations are used for this
- relationship types referring to just one entity type are called recursive



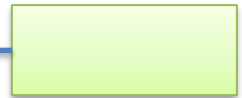
## 2.3 ER – Relationship Cardinality

- **Cardinality** annotations

- one cardinality annotation per entity type / relationship end

- minimum and maximum constrains possible written as (min, max)

cardinality



- Common Cardinality Expressions

- **(1, 1)**: each entity is bound exactly once
    - **(0, \*)**: each entity may participate arbitrarily often in the relationship
    - **(2, \*)**: each entity may participate arbitrarily often in the relationship, but at least twice

- Convention you might see outside this lecture

- no annotation is usually interpreted as (0, \*)
    - if only one symbol / number  $s$  is used, this is interpreted as (0,  $s$ )  
     $*$  = (0, \*); 4 = (0, 4)
    - sometimes, N or M are used instead of \*



## 2.3 ER – Relationship Cardinality

- Cardinalities express how often a specific entity may appear within a relationship set
  - Please note: There are other notations which look similar but use different semantics (e.g., UML)

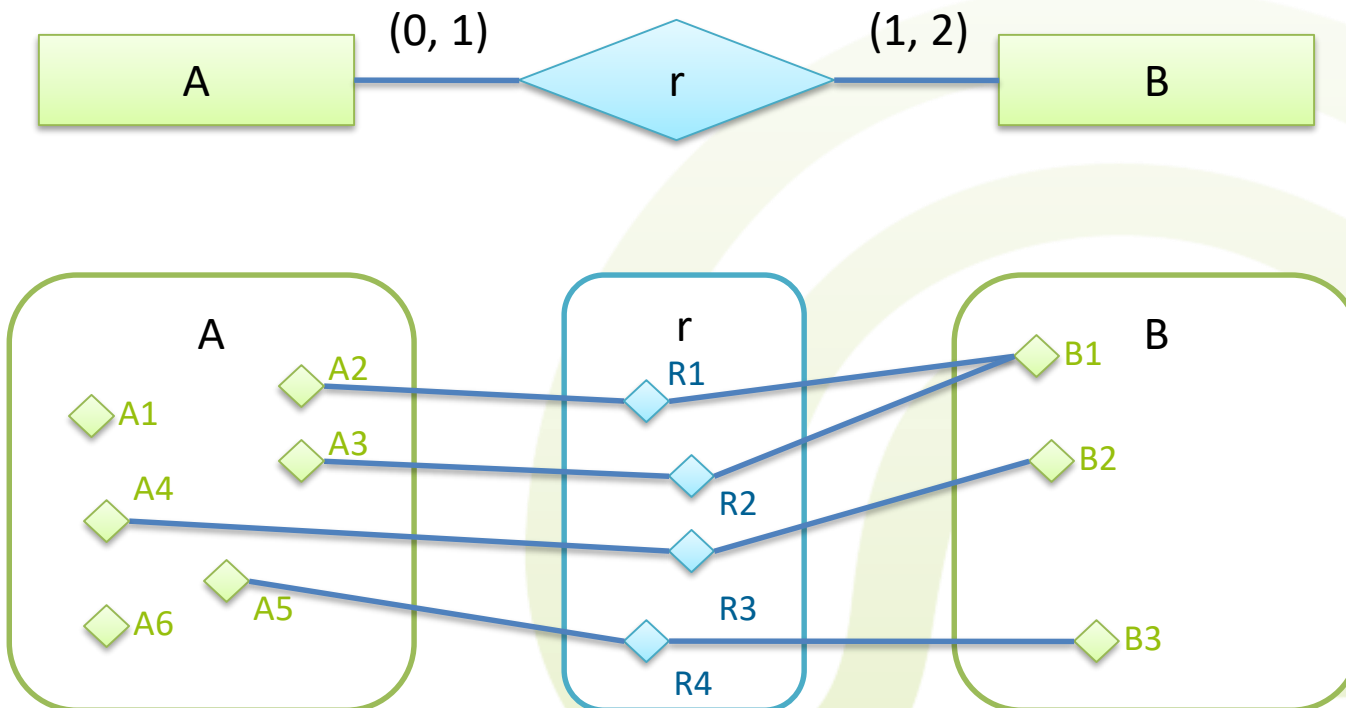


- a specific entity of type A may appear *up to once* in the relationship set, an entity of type B appears *at least once* and *at most twice*
  - this means: Up to two entities of type A may relate to one entity of type B. Some entities in A are not related to any in B. All entities in B are related to at least one in A.



## 2.3 ER – Relationships

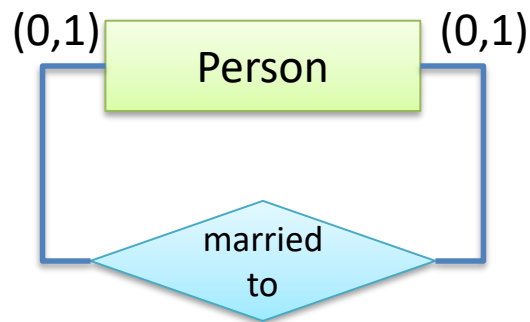
- To each entity of type B, one or two entities of type A are related*





## 2.3 ER – Relationship Cardinality

- Example
  - *Each person can only be married to one other person.*

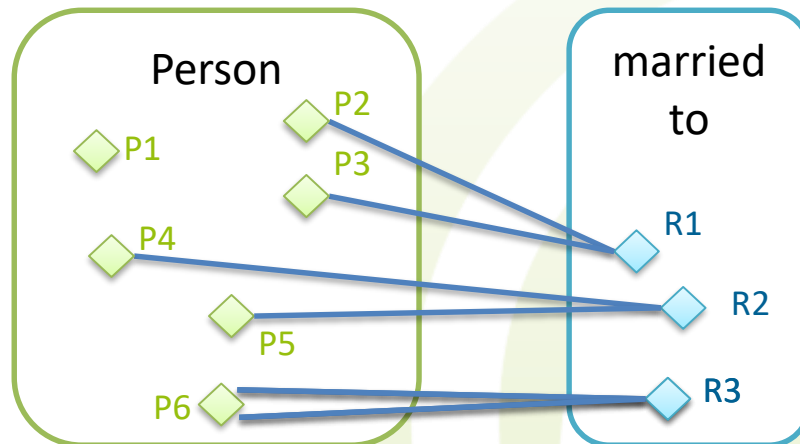
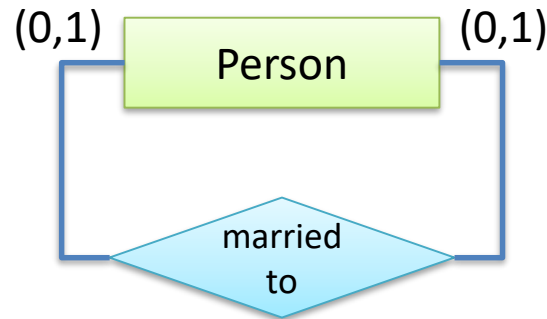


- each entity can only appear in one instance of the *married to* entity set
  - Still, could be married to oneself





## 2.3 ER – Relationships





## 2.3 ER – Relationship Cardinality

- Example

- *A cat has up to 4 owners, but at least one. A person may own any number of cats.*



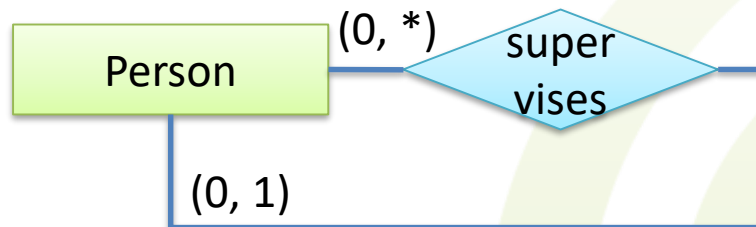


## 2.3 ER – Relationship Cardinality

- Example

- A person may supervise any other number of persons.

- Wolf-Tilo Balke supervises his WiMis
- His WiMis supervise their research HiWis
- His WiMis' research HiWis do not supervise anyone



- A person is supervised by at most one person



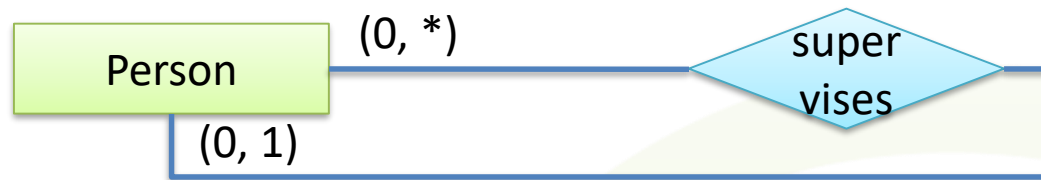
## 2.3 ER – Relationship Cardinality

- Cardinalities for **binary** relationship types can be classified into common, more general **cardinality types**
  - these cardinality types are also often found in other modeling paradigms
    - **One-To-One (1:1)** – each entity of the first type can only relate to exactly one entity of the other type
    - **One-To-Many (1:N)** – each entity of the first type can relate to multiple entities of the other type
    - **Many-To-One (N:1)** – multiple entities of the first type can relate to exactly one entity of the second type
    - **Many-To-Many (N:M)** – any number of entities of first type may relate to any number of entities of second type (no restrictions)
  - As we will see later, these will have a direct impact on the logical database schema

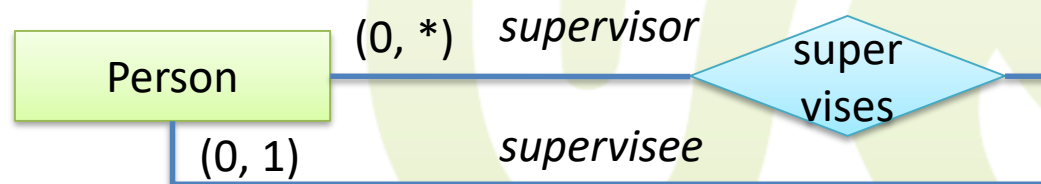


## 2.3 ER – Relationship Roles

- Often, it is beneficial to clarify the **role** of an entity within a relationship
  - e.g. relationship *supervises*



- what is meant? Who is the supervisor? Who is the supervised person?
- roles can be annotated on the relationship lines
  - Careful! These are only labels for clarification, nothing more!





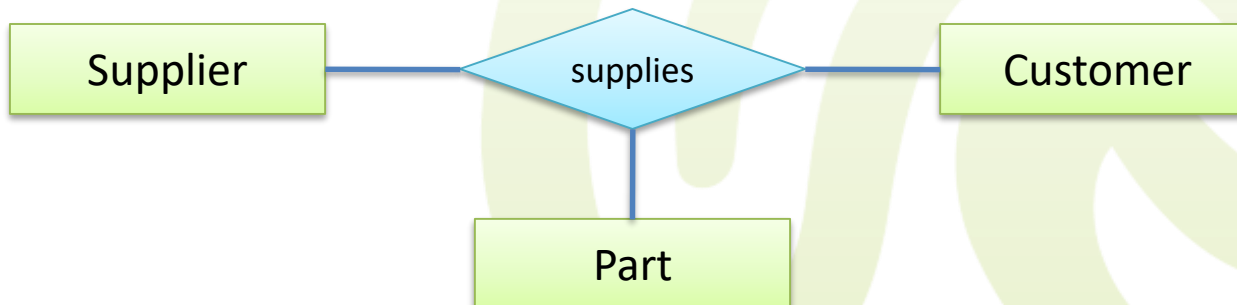
## 2.3 ER – Relationship Degree

- Relationship instances involve multiple entities
  - the number of entities in each relationship instance is called **relationship degree**

- degree = 2 – Binary Relation



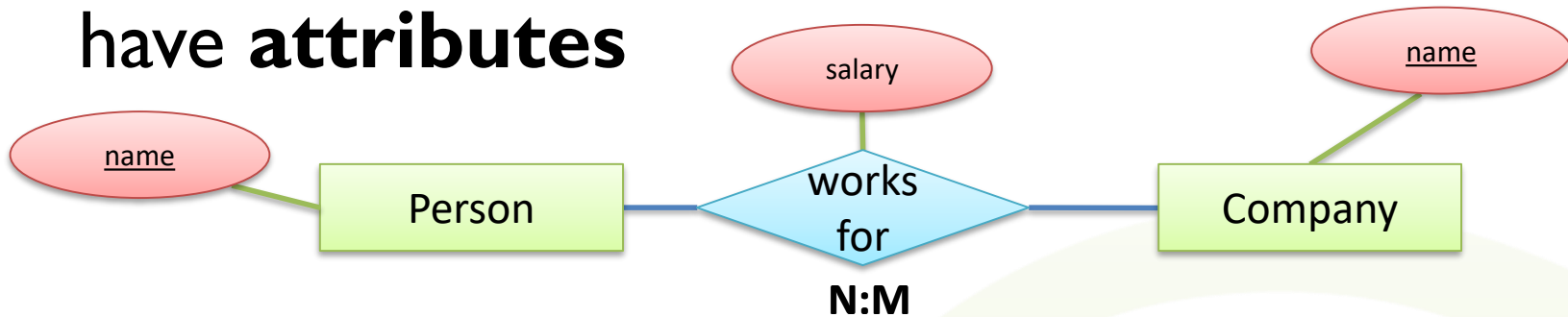
- degree = 3 – Ternary Relation





## 2.3 ER – Relationship Attributes

- Similar to entities, relationship types may even have **attributes**



- Later, when designing the logical schema:
  - for 1:1 relationships, the relationship attribute may be migrated to any of the participating attributes
  - for 1:N relationships, the attribute may be only migrated to the entity type on the N-side
  - for N:M relationships, no migration is possible



## 2.3 ER – Total Participation

- To express that all entities of an entity type appear in a certain relationship set, the concept of **total participation** can be used
  - the entity type which is totally participating is indicated by a double line
  - e.g. *each driver's license must belong to exactly one person.*
    - *There are no unassigned licenses*





## 2.3 ER – Weak Entities

- Each entity needs to be identifiable by a set of **key attributes**
- Entities that exist **independently** of the context are called **strong entities**
  - a person exists whether it is married or not
- In contrast, there may be entities without a unique key called **weak entities**





## 2.3 ER – Weak Entities

- **Weak entities** are identified by being related to strong entities
  - the strong entities *own and define* the weak entities
    - the weak one cannot exist without the strong ones
  - the relationships relating the strong to the weak are called **identifying relationships**
    - weak entities are **totally participating** in that relationship
  - weak entities have **partial keys** which are unique within the identifying relationship sets of their strong entities
    - to be unique, the weak entity instance has to borrow the key values of the respective strong entity instances



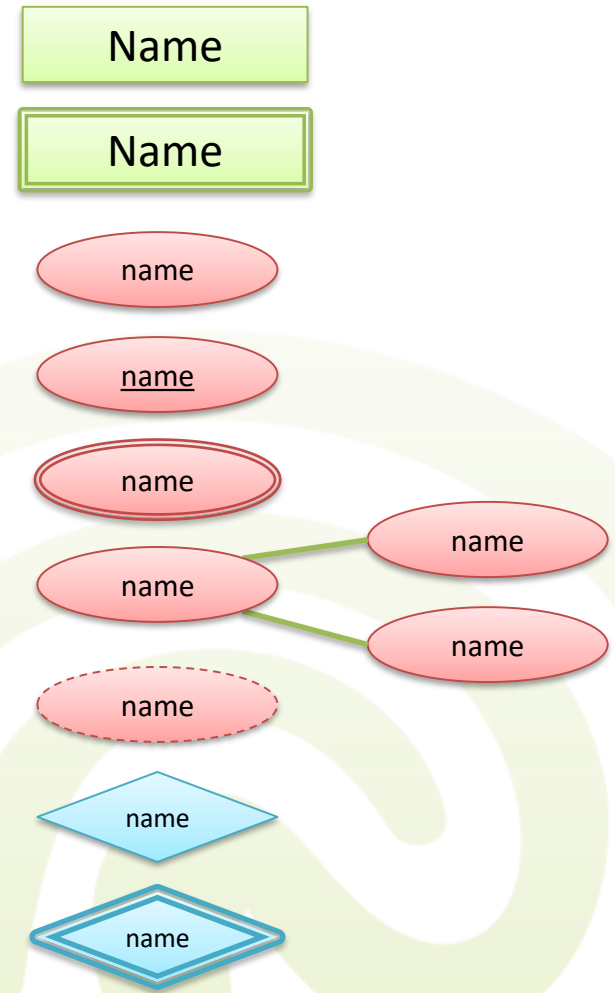
- [illegible]

- [illegible]



## 2.3 ER – Overview

- Entity Type
- Weak Entity Type
- Attribute
- Key Attribute
- Multi-valued Attribute
- Composite Attribute
- Derived Attribute
- Relationship Type
- Identifying Relationship Type





## 2.3 ER – Overview

- Total participation of E2 in R



- Cardinality

- an instance of E1 may relate to multiple instances of E2



- Specific cardinality with min and max

- an instance of E1 may relate to multiple instances of E2





## 2.3 Schema Modelling

- **Problems:** Persons designing a schema for the **same domain** will often come up with **very different schemas**
  - each schema can be a correct representation of the domain
  - but merging and mapping them is difficult due to their differences
  - **exchanging** and **integrating data** between organizations with incompatible schemas is tough





## 2.3 Schema Modelling

- often **different levels of abstraction** are used
  - the semantic expressiveness of schemas is different
  - e.g. one schema may contain *Cows* and *Dolphins* while another only contains the higher-level concept *Animals*
- **extending** a schema is often necessary
  - e.g. when the focus changes or new information about the domain becomes available
  - schemas limit what can be expressed about a domain
  - adjustments may result in a complete re-modeling of a schema



# Quick Exercise

## Exercise

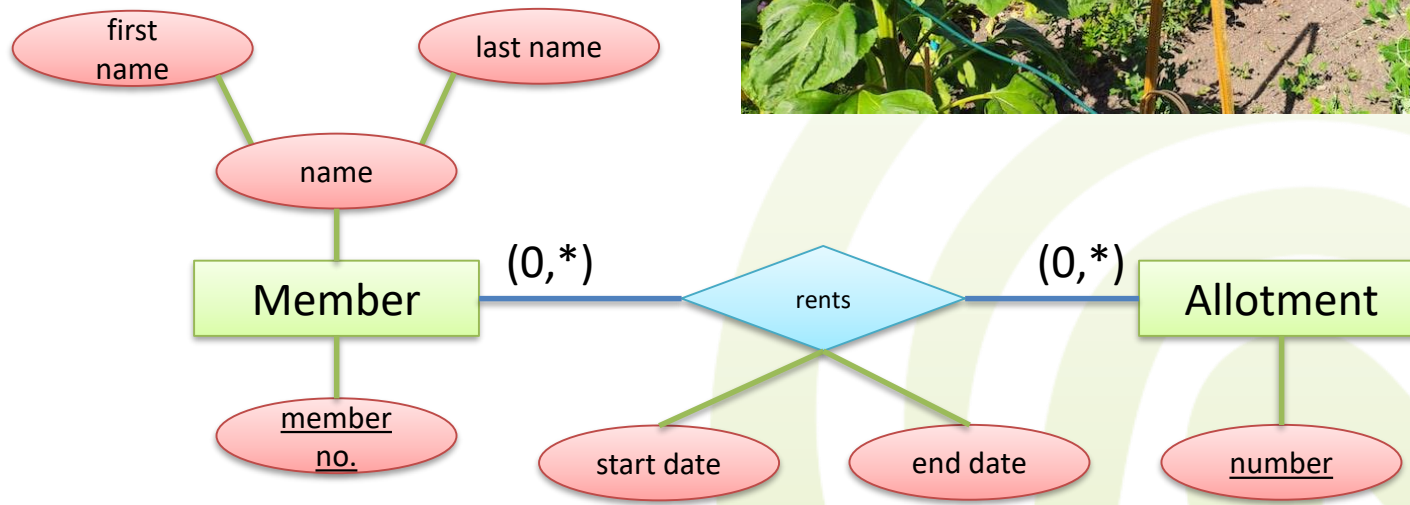
- We want to build a database for an **allotment club** (German: **Kleingartenverein**)
  - In our database, we have club **members**
  - Each member has a real **name**, which consists of a first name and a last name. Also, each member has a unique **member number**.
  - There are **garden patches** with unique numbers. Each member can rent as many of them as they like.
    - Some allotments are even rented by multiple members!
  - For each member who rents an allotment (German: Kleingarten), the start and end **date** must be stored.





# Quick Exercise

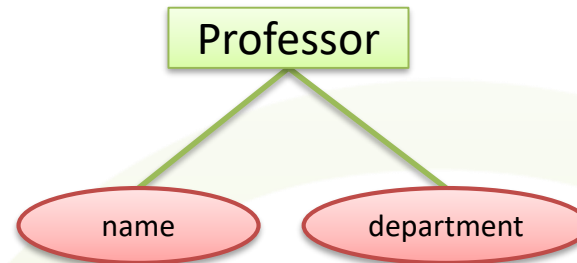
## Exercise





# 2 Data Modeling I

- Phases of DB Design
- Data Models
- Basic ER Modeling
  - Chen Notation
  - Mathematical Model
- **Example**





## 2.4 Example

*Detour*

- We want to model a simple university database
  - *In our database, we have students. They have a name, a registration number, and a course of study.*
  - *The university offers lectures. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have a unique ID*
  - *Each year, some of these lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may register for that lecture.*
  - *Professors have a name and are member of a specific department.*





## 2.4 Example

*Detour*

- How to start? What to do?
  - find the basic **entity types**
  - find the **attributes** of entities
    - decide to which entity an attribute should be assigned
    - which attributes are key attributes?
    - some attributes are better modeled as own entities, which ones?
  - define the **relationship types**
    - which role do entities play?
    - do relationships require additional entity types?
    - are the relationships total? Identifying? Are weak entities involved?
    - what are the cardinalities of the relationship type?



## 2.4 Example

*Detour*

- Which are our entity types?
  - In our database, we have **students**. They have a name, a registration number and a course of study.
  - The university offers **lectures**. Each lecture may be part of some course of study in a certain semester. Lectures may have other lectures as prerequisites. They have a title, provide a specific number of credits and have a unique ID
  - Each year, some of these lectures are offered by a **professor** at a certain day at a fixed time in a specific room. Students may register for that lecture.
  - Professors have a name and are member of a specific department.





## 2.4 Example

*Detour*

Student

Lecture

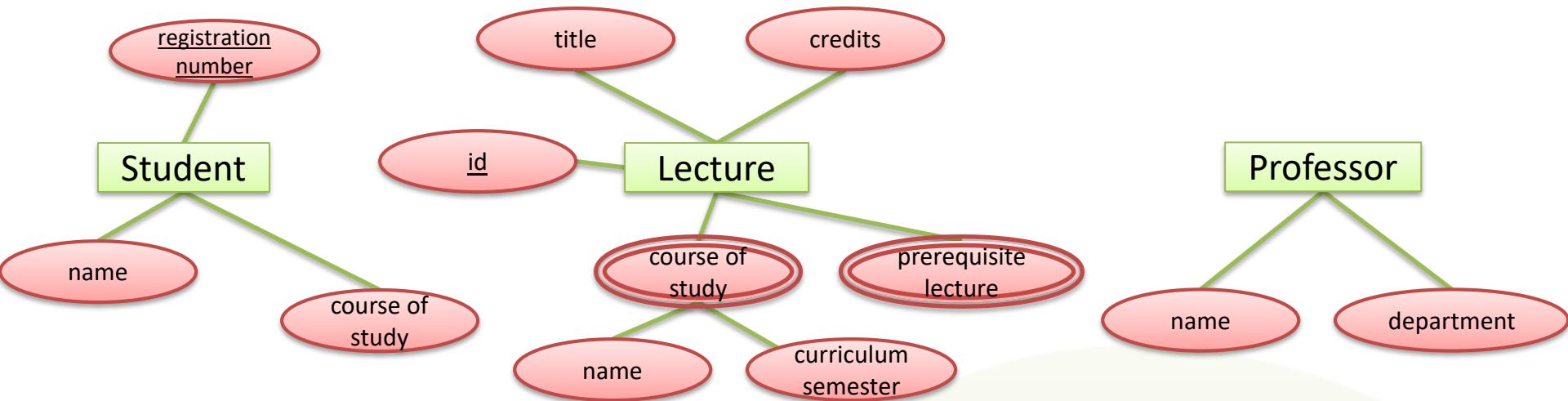
Professor

- What attributes are there?
  - In our database, we have students. They have a **name**, a **registration number** and a **course of study**.
  - The university offers lectures. Each lecture may be part of some **course of study** in a certain **semester**. Lectures may have other lectures as **prerequisites**. They have a **title**, provide a specific number of **credits** and have **unique ID**
  - Professors have a **name** and are member of a specific **department**.



## 2.4 Example

*Detour*

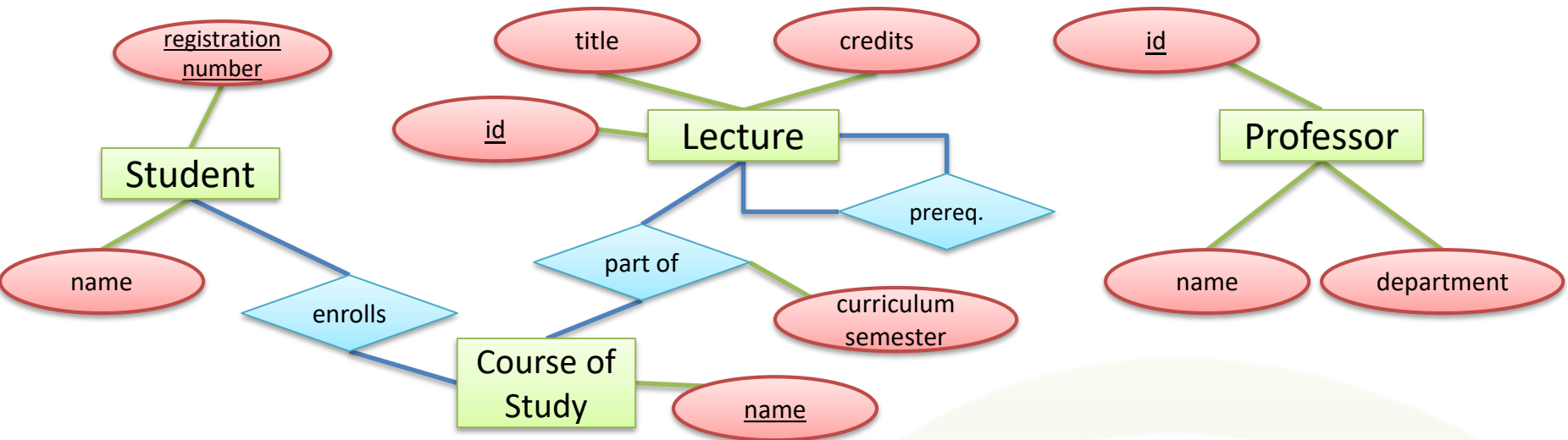


- First try...
  - this model is **really crappy!**
  - *course of study* does not seem to be an attribute
    - used by student and lecture. Even worse, lecture refers to a course of study in a specific curriculum semester.
    - use additional entity type with relationships!
  - *prerequisite lecture* also is not a good attribute
    - prerequisite lectures are also lectures. Use a relationship instead!
  - *professor* does not have key attributes



## 2.4 Example

*Detour*

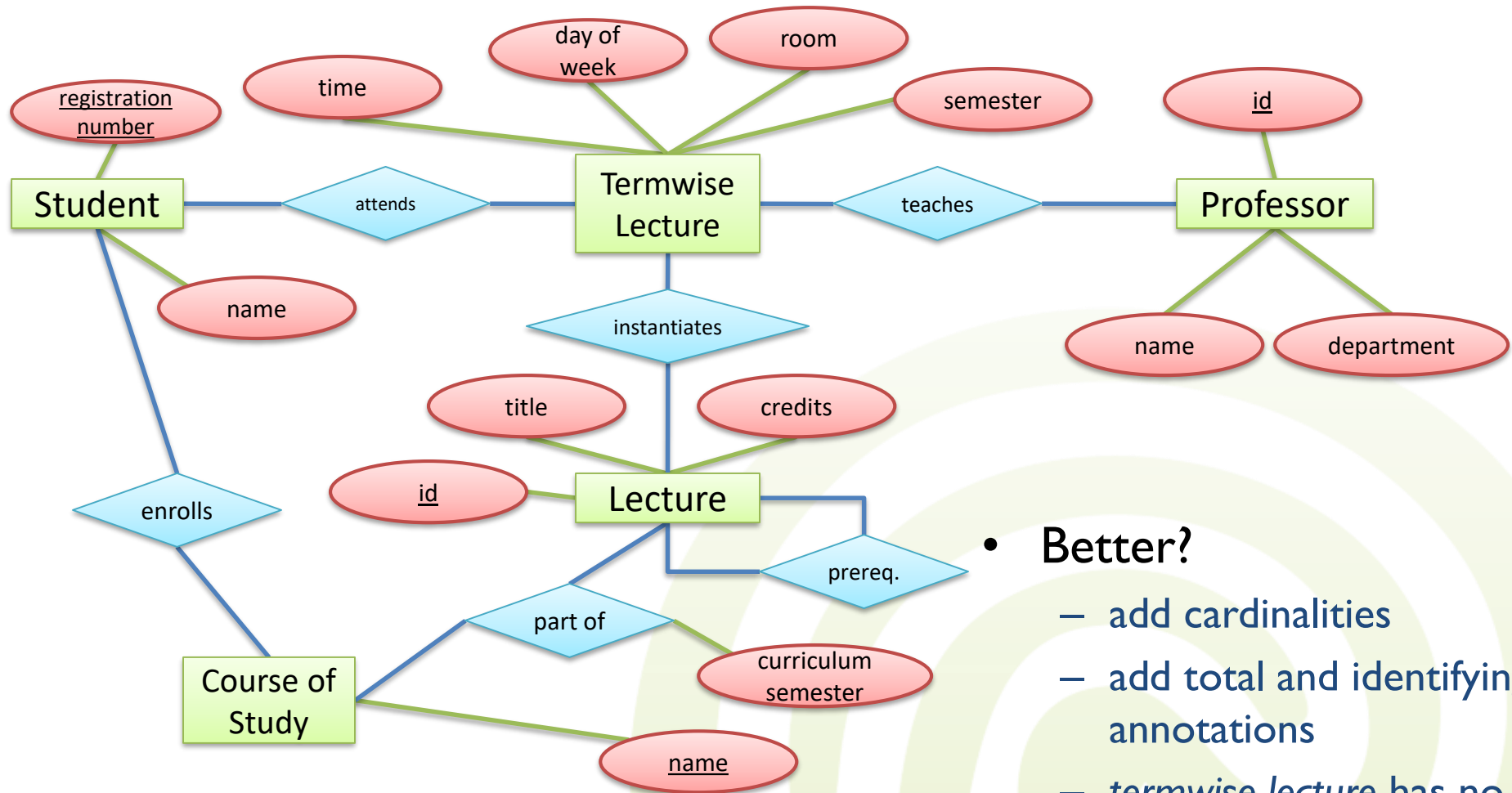


- Second try
  - professor uses a **surrogate** key now
    - key is automatically generated and has no meaning beside unique identification (**but must be present!**)
  - course of study is an entity type now
- Which entity types are additionally related?
  - Each year, some lectures of the pool of all lectures are offered by a professor at a certain day at a fixed time in a specific room. Students may attend that lecture.



## 2.4 Example

*Detour*

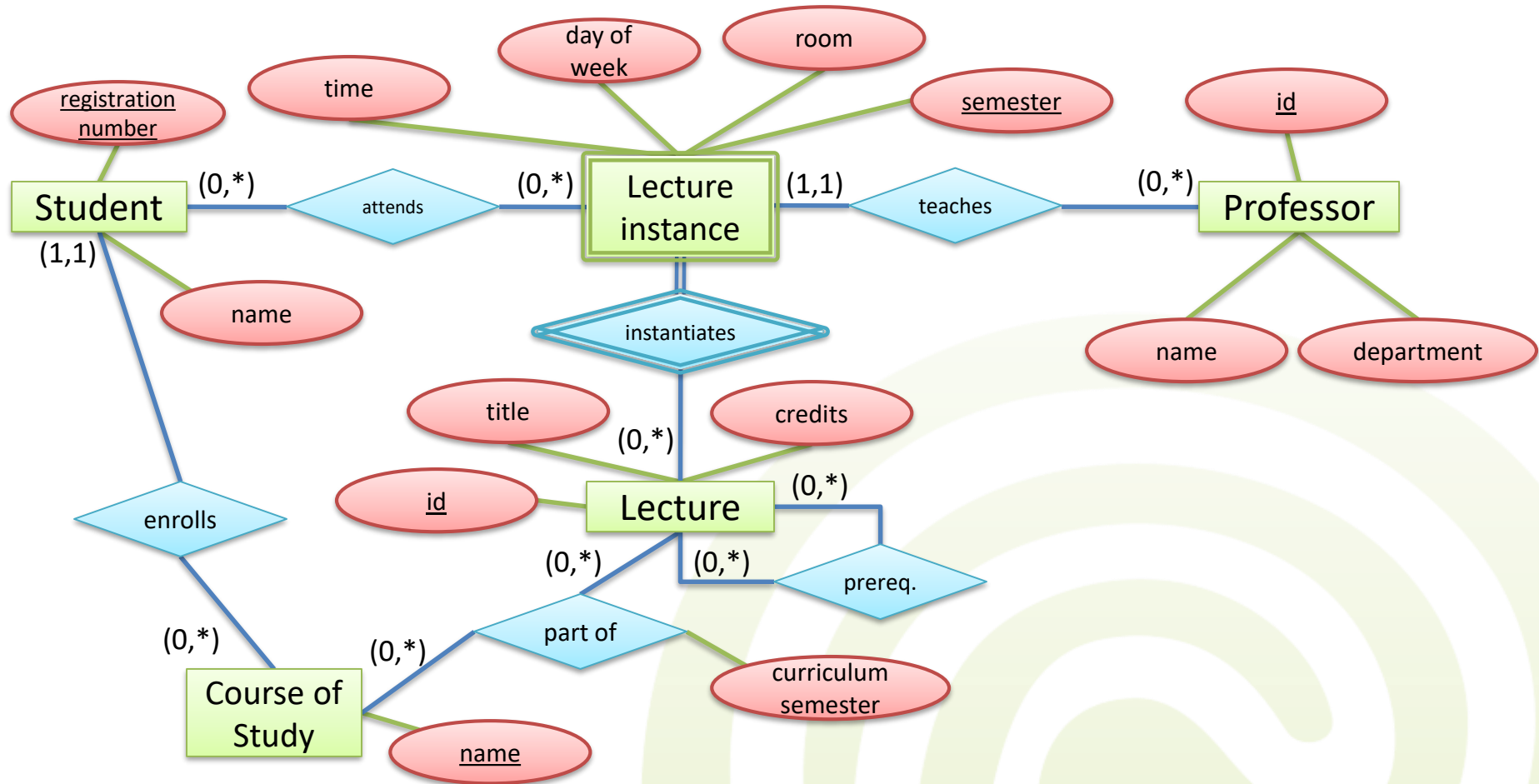


- **Better?**
  - add cardinalities
  - add total and identifying annotations
  - *termwise lecture* has no key



## 2.4 Example

*Detour*





## 2.4 Example

*Detour*

- In general, modeling is not that simple
- Many possible ways of modeling the same miniworld
  - some are more elegant, some are less elegant, but all may be **valid**!
- Models alone are not enough, they need to be documented
  - what do the attributes mean?
  - what do the relationships mean?



# Next week

- Alternative ER Notations
- Extended ER
  - Inheritance
  - Complex Relationships
- Taxonomies & Ontologies
- UML

