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

Common Mistakes in Modelling

Summary

- Mistake: **No Primary Key**
  - Just don't do that outside of simple examples
  - (same is true for cardinalities)
- Mistake: **No Relation Symbol or Name**

- Mistake: **Modelling Functionality as Data**
  - "Users can write reviews for a book, which they can also delete."

Common Mistakes in Modelling

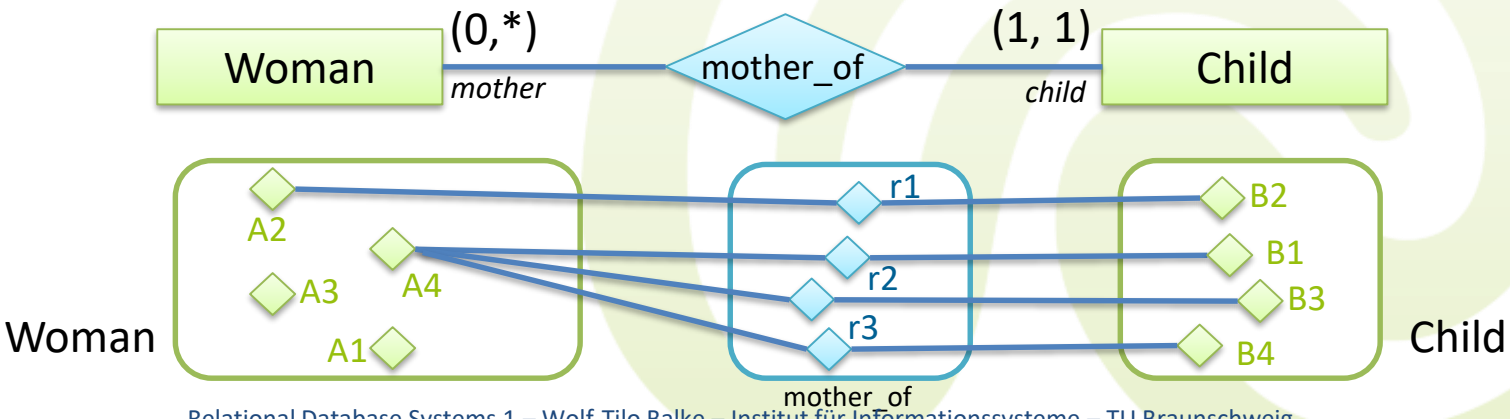
Summary

- Mistake: **Primary Key does not make sense**
  - Intuition: **An attribute / set of attributes which uniquely identify an entity**
  - Additional soft constraints
    - Should feel "natural"
    - Should be minimal
    - Should be easy to handle
  - Example: Modeling a book in a book store
    - Good: Book(name, author, year, <u>isbn</u>, summary, price)
    - Less good: Book(<u>name, author, year</u>, isbn, summary, price)
      - More natural, but more complex. Only valid if it is guaranteed that a given author writes only a single book with the same name in a year. Depends on task if this makes sense .
    - Not good or even invalid:
      - Book(name, author, year, isbn, <u>summary</u>, price)
      - Book(name, author, year, isbn, summary, <u>price</u>)
  - **Weak Entities** always have composite key
    - One component is the primary key of the strong entity, the second component is with the weak entity and is only unique within the set of weak entities belonging to the same strong one

Common Mistakes in Modelling

Summary

- Mistake: **Confusing Cardinality Notations**
  - Warning: There are several conflicting ways of writing cardinalities!
    - Whichever way you choose, **BE CONSISTENT!**
  - **Classic ER Style (default in this lecture)**:
    - Read cardinalities in diagram below as
      - "each entity of *woman* participates in any number of relationships of *mother_of*, while each entity in *child* must participate in exactly one"
    - This notation is needed for non-trivial n-ary relationsships (n>2)

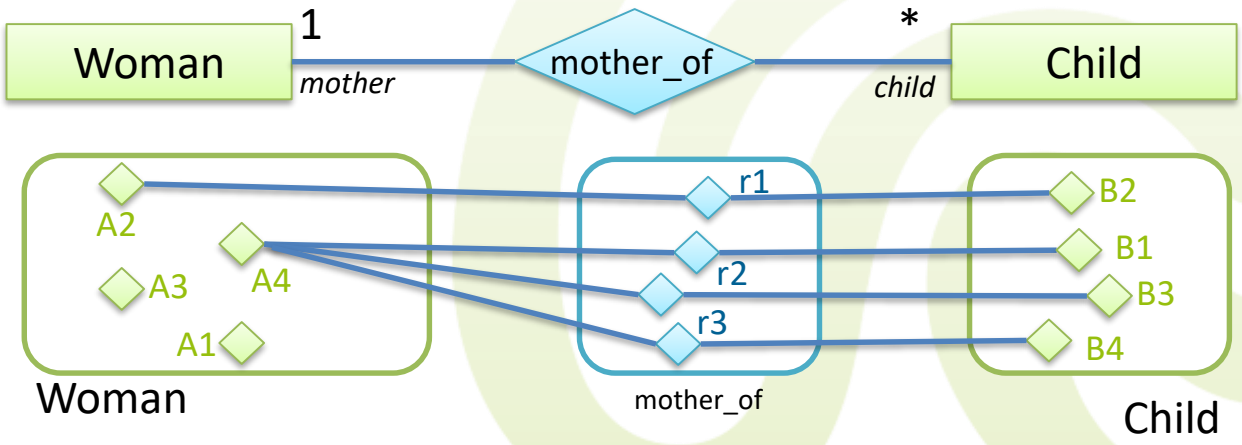


Common Mistakes in Modelling

Summary

– **UML Style** (not our lecture's default):

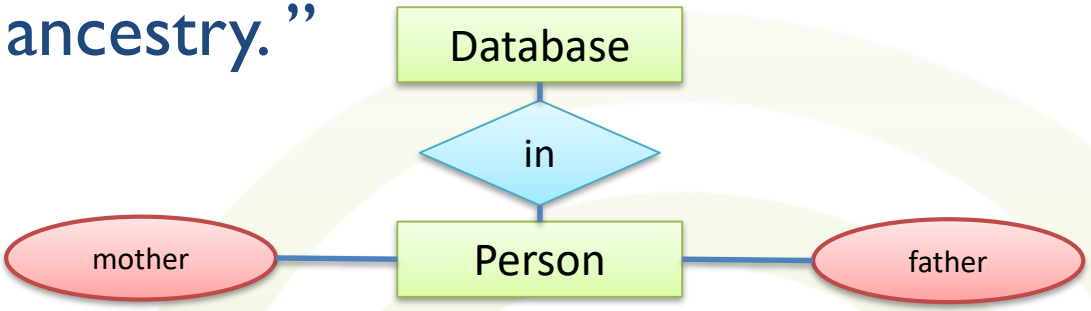
- Read cardinalities in diagram below as
  - "each one entity of *woman* is in relation (via *mother_of*) with any number of entities of *child*"
- Usually only distinguishes between 0, 0..1, 1, 0..* and *
  - Fine-granular cardinalities and higher arity relations not easily possible



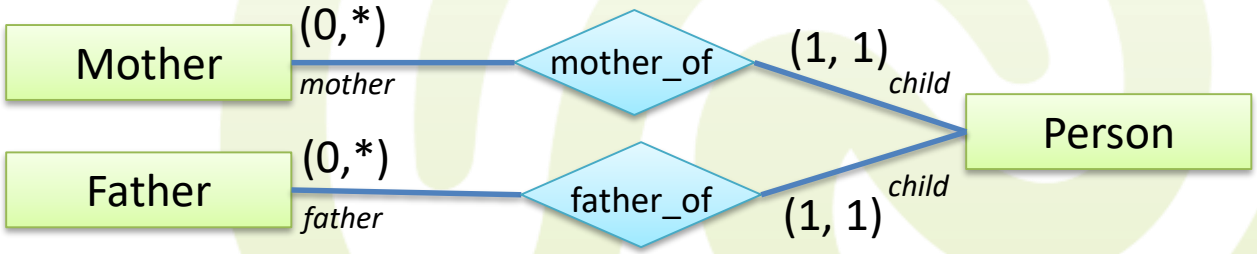
Common Mistakes in Modelling

Summary

- Mistake: **Model not suitable for the task**
  - Task: "Build a person database. Each person should have a mother and a father. The database should be used to explore ancestry."
  - Very Bad:



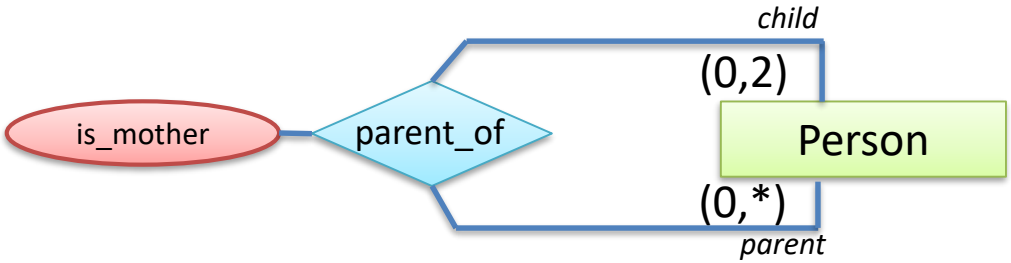
  - Still quite Bad:



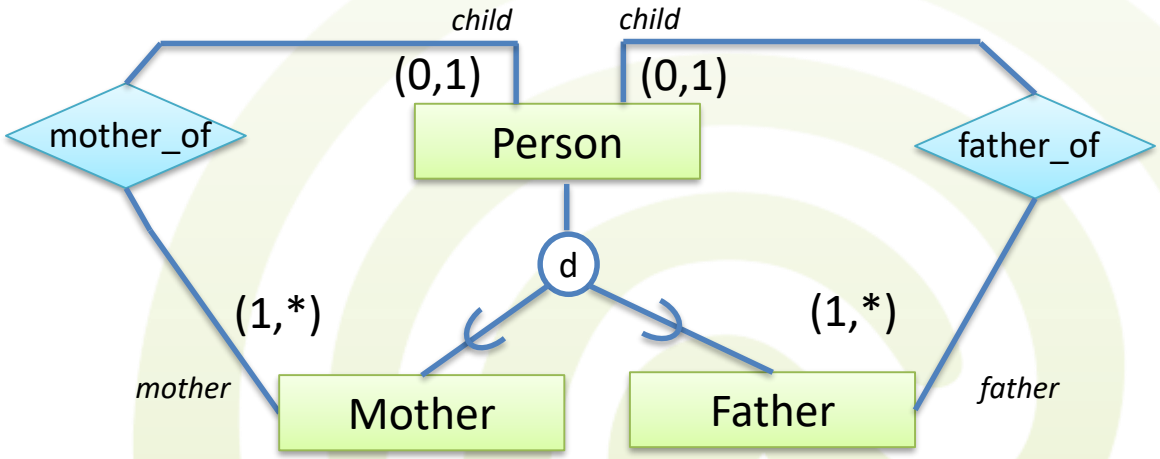
Common Mistakes in Modelling

Summary

– Better:

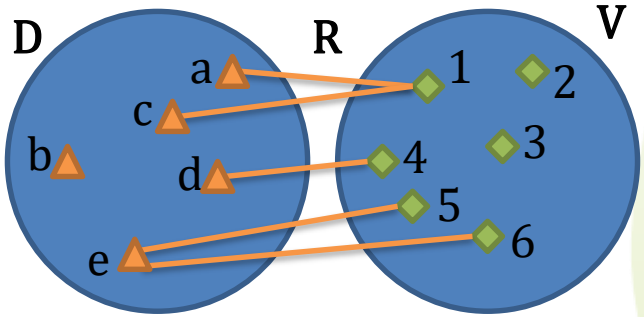


– Or maybe:



5 Relational Model

- **Basic Set Theory**
- Relational Model
- From Theory to Practice
- Integrity Constraints
- Conversion from ER



5.1 Basic Set Theory

- **Set theory** is the foundation of mathematics
  - you probably all know these things from your math courses, but repeating never hurts
  - the **relational model** is based on set theory; understanding the basic math will help a lot

5.1 Sets

- A **set** is a mathematical **primitive,** and thus has no formal definition
- A set is a **collection** of objects (called *members* or *elements* of the set)
  - objects (or entities) have to be understood in a very broad sense, can be anything from physical objects, people, abstract concepts, other sets, …
- Objects **belong** (or do **not belong**) to a set (alternatively, *are* or *are not* in the set)
- A set **consists** of all its elements

5.1 Sets

- Sets can be specified **extensionally**
  - list all its elements
  - e.g. A = {ifis, 42, Balke, Hurz!}
- Sets can be specified **intensionally**
  - provide a criterion deciding whether an object belongs to the set or not (**membership criterion**)
  - examples:
    - $A = \{x \mid x > 4 \text{ and } x \in \mathbb{Z}\}$
    - $B = \{x \in \mathbb{N} \mid x < 7\}$
    - C = {all facts about databases you should know}
- Sets can be either **finite** or **infinite**
  - set of all super villains is finite
  - set of all numbers is infinite

5.1 Sets

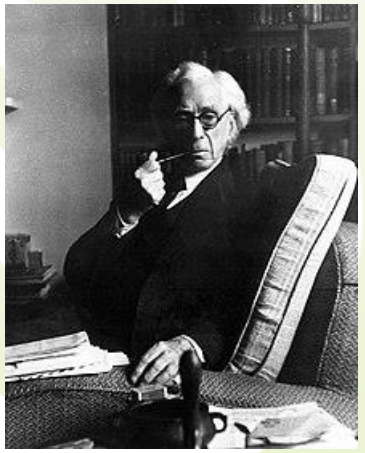
- Sets are different, iff they have different members
  - $\{a, b, c\} = \{b, c, a\}$
  - duplicates are not supported in standard set theory
    - $\{a, a, b, c\} = \{a, b, c\}$
- Sets can be empty (written as {} or ∅)
- Notations for set membership
  - $a \in \{a, b, c\}$
  - $e \notin \{a, b, c\}$

5.1 Sets

- Defining a set by its **intension**
  - intension must be **well-defined** and **unambiguous**
  - there is always is a clear **membership criterion** to determine whether an object belongs to the set (or not)
  - Example for an invalid definition (Russell's paradox):

> In a small town, there is just one male barber. He shaves all and only those men in town who do not shave themselves.

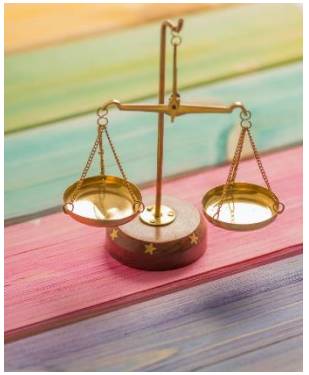
- **does the barber shave himself?**



5.1 Sets

- Still, the set's **extension** might be unknown (however, there is one)
- Example
  - *All students in this room who are older than 22.*
  - well-defined, but not known to me …
  - but (at least in principle) we can find out!
- Why should we care? Because:
  - Intensional set ≈ database query
  - Extensional set ≈ result of a query, table

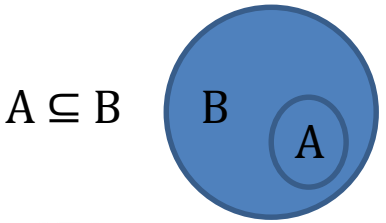
5.1 Sets



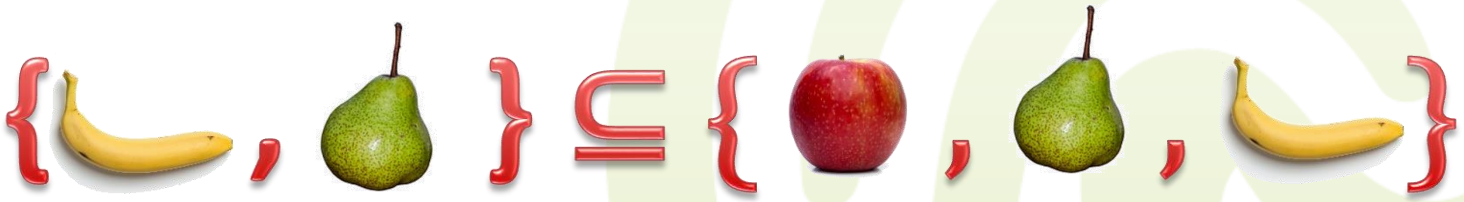
- For every set and object, there is an accompanying definition of **equality** (or equivalence)
  - is $x = y$?
- However, you could have two different descriptions of the same element
  - example: the set of all 26 *standard* letters
    - 'ö' is not contained in this set
    - 'm' = 'M' and both reflect a single element of the set
      - 'm' and 'M' are different descriptions of the **same** object
  - example: the set of all 59 letters and umlauts in German
    - 'ö' is element of the set
    - 'm' ≠ 'M' and are both elements of the set (two different objects)

5.1 Sets

- Sets have a cardinality (i.e., number of elements)
  - denoted by $|A|$
  - $|\{a, b, c\}| = 3$



- Set A is a **subset** of set B, denoted by $A \subseteq B$, iff every member of A is also a member of B
- B is a **superset** of A, denoted by $B \supseteq A$, iff $A \subseteq B$



5.1 Tuples

- A **tuple** (or vector) is a sequence of objects
  - length 1: Singleton
  - length 2: Pair
  - length 3: Triple
  - length *n*: *n*-tuple

- In contrast to sets…
  - tuples can contain an object more than once
  - the objects appear in a certain **order**
  - the length of the tuple is **finite**
- Written as ⟨a, b, c⟩ or (a, b, c)

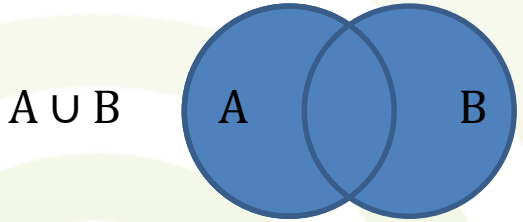
5.1 Tuples

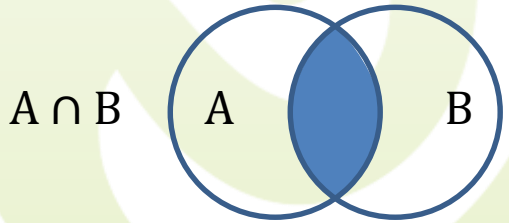
- Hence
  - $\langle a, b, c\rangle \neq \langle c, b, a\rangle$, whereas $\{a, b, c\} = \{c, b, a\}$
  - $\langle a_1, a_2\rangle = \langle b_1, b_2\rangle$ iff $a_1 = b_1$ and $a_2 = b_2$

- ***n*-tuples** ($n > 1$) can also be defined as a cascade of ordered pairs:
  - $\langle a, b, c, d\rangle = \langle a, \langle b, \langle c, d\rangle\rangle\rangle$

5.1 Set Operations

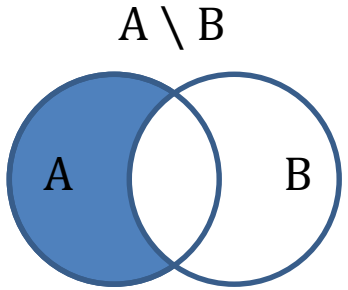
- Four binary **set operations**
  - union, intersection, difference and cartesian product
- **Union**: ∪
  - creates a new set containing all elements that are contained in (at least) one of two sets
  - $\{a, b\} \cup \{b, c\} = \{a, b, c\}$
- **Intersection**: ∩
  - creates a new set containing all elements that are contained in both sets
  - $\{a, b\} \cap \{b, c\} = \{b\}$





5.1 Set Operations

- **Difference**: \
  - creates a set containing all elements of the first set without those also being in the second set
  - $\{a, b\} \setminus \{b, c\} = \{a\}$



5.1 Set Operations

- **Cartesian Product**: $\times$
  - the cartesian product is an operation between two sets, creating a new set of pairs such that:
    $$A \times B = \{\langle a, b\rangle \mid a \in A \text{ and } b \in B\}$$
  - named after René Descartes
- Example
  - $\{a, b\} \times \{b, c\} = \{\langle a, b\rangle, \langle a, c\rangle, \langle b, b\rangle, \langle b, c\rangle\}$
  - $\text{Color} = \{ \text{red}, \text{white} \}$
  - $\text{Building} = \{ \text{house}, \text{palace} \}$
  - $\text{Color} \times \text{Building} = \{\langle \text{red}, \text{house}\rangle, \langle \text{white}, \text{house}\rangle, \langle \text{red}, \text{palace}\rangle, \langle \text{white}, \text{palace}\rangle\}$

- The cartesian product can easily be extended to higher dimensionalities: $A \times B \times C$ is a set of triples

5.1 Relations

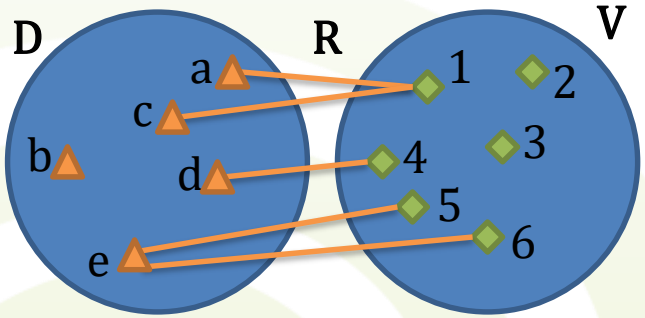
- A **relation** R over some sets $D_1, \ldots, D_n$ is a **subset** of their **cartesian** product
  - $R \subseteq D_1 \times \ldots \times D_n$
  - the elements of a relation are **tuples**
  - the D_i are called **domains**
  - each D_i corresponds to an **attribute** of a tuple
    - *n*=1: Unary relation or **property**
    - *n*=2: Binary relation
    - *n*=3: Ternary relation
    - …

5.1 Relations

- Some important properties
  - relations are sets in the mathematical sense, thus **no duplicate tuples** are allowed
  - the **set of tuples** is **unordered**
  - the **list of domains** is **ordered**
  - relations can be modified by…
    - **inserting** new tuples,
    - **deleting** existing tuples, and
    - **updating** (that is, modifying) existing tuples.

5.1 Relations

- A special case: Binary relations
  - $R \subseteq D_1 \times D_2$
    - D_1 is called **domain,** D_2 is called **co-domain** (range, target)
  - relates objects of two different sets to each other
  - R is just a set of ordered pairs
  - $R = \{\langle a,1\rangle, \langle c,1\rangle, \langle d,4\rangle, \langle e,5\rangle, \langle e,6\rangle\}$
    - can also be written as a**R**1, c**R**1, d**R**4, …
  - imagine **Likes ⊆ Person × Beverage**
    - Tilo Likes Coffee, Christian Likes Tea, …
  - For example, binary relations can naively be used to implement n:m relationship types in a logical data model
  - Functions are a special case of binary relations



5.1 Relations

- Example
  - Accessory = {thorns, spines}
  - Shape = {star, bowl}
  - Color = {red, white}

Color × Shape × Accessory =
{⟨red, star, thorns⟩,
⟨red, star, spines⟩,
⟨red, bowl, thorns⟩,
⟨red, bowl, spines⟩,
⟨white, star, thorns⟩,
⟨white, star, spines⟩,
⟨white, bowl, thorns⟩,
⟨white, bowl, spines⟩}

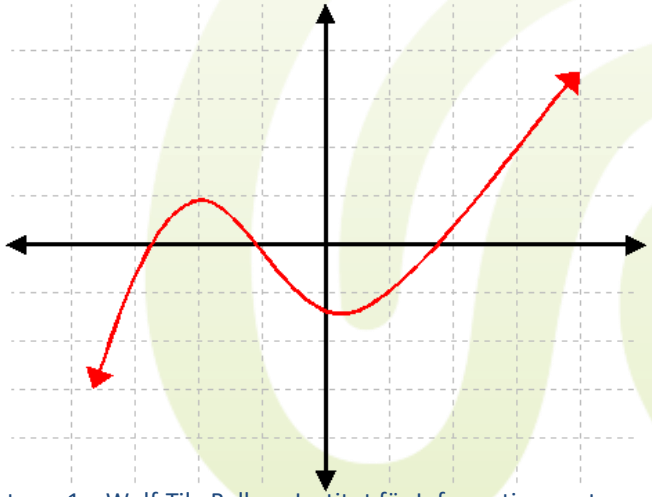
5.1 Relations

- Relation **FamousFlowerLooks** defined as:
  - FamousFlowerLooks ⊆ Color × Shape × Accessory

FamousFlowerLooks =
{⟨red, bowl, thorns⟩,
⟨white, star, spines⟩}

5.1 Functions

- **Functions** are special case of binary relations
  - **partial function:**
    each element of the domain is related to
    at most one element in the co-domain
  - **total function:**
    each element in the domain is related to
    exactly one element in the co-domain

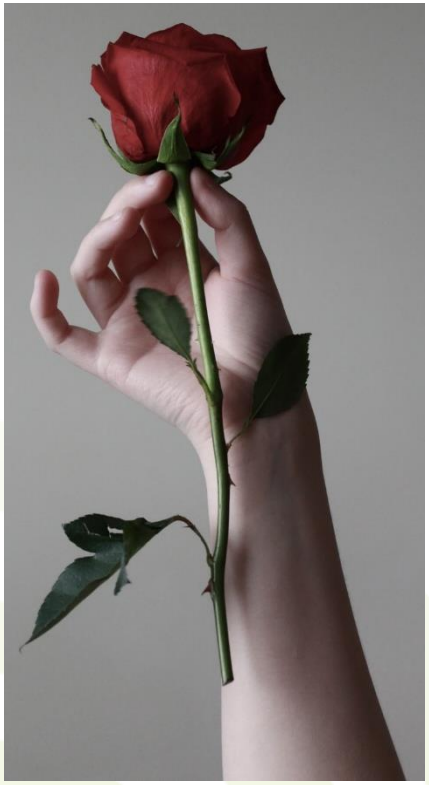


5.1 Functions

- Functions can be used to **abstract** from the exact order of domains in a relation
  - alternative definition of relations:
    a relation is a set of functions
  - every tuple in the relation is considered as a function of the type $\{A_1, ..., A_n\} \rightarrow D_1 \cup ... \cup D_n$
    - that means, every tuple maps each attribute to some value

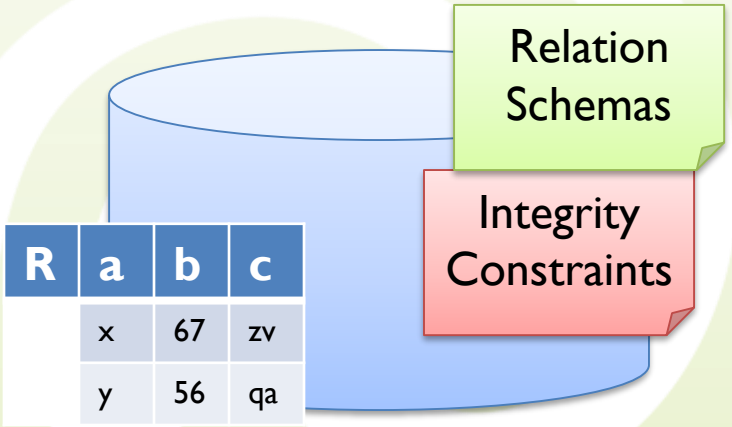
5.1 Functions

- Example
  - Color = {red, white}
  - Shape = {star, bowl}
  - Accessory = {thorns, spines}
  - to be independent of the domain order, the tuple ⟨red, bowl, thorns⟩ can also be represented as the following function *t*
    - *t*(Color) = red
    - *t*(Shape) = bowl
    - *t*(Accessory) = thorns
  - Usually, one writes t[color] instead of t(color)
  - This can be used to change the order of domains for tuples
    - t[Shape, Accessory, Color] = ⟨bowl, thorns, red⟩



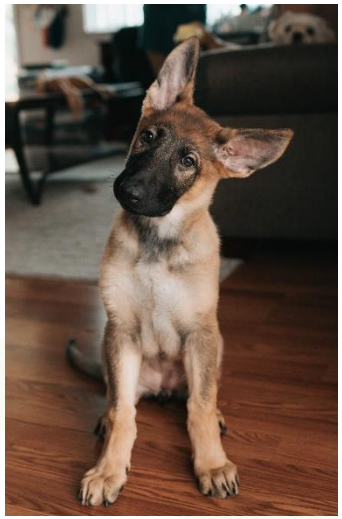
5 Relational Model

- Basic Set Theory
- **Relational Model**
- From Theory to Practice
- Integrity Constraints
- Conversion from ER



5.2 Relational Model

- Well, that's all nice to know… but: we are here to learn about **databases!**
  - where is the connection?



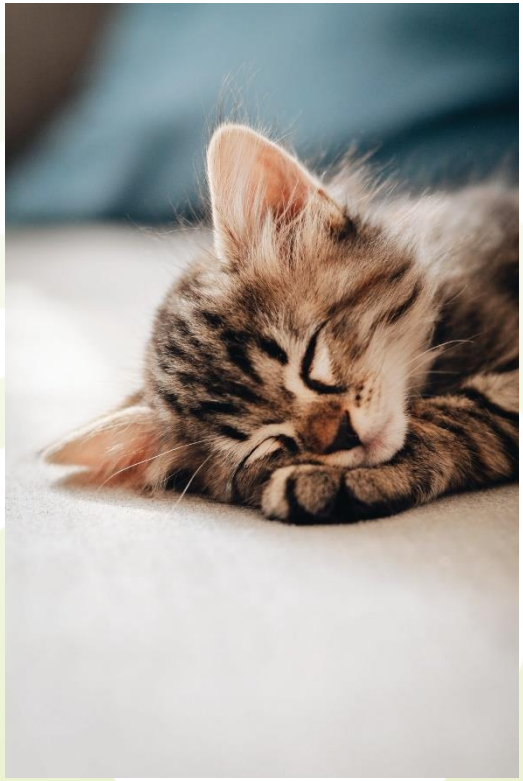
- **Here it is…**
  - a **database schema** is a description of concepts in terms of relations and attribute domains
  - a **database instance** is a set of tuples having certain attribute values

5.2 Relational Model

- OK, then…
  - designing a database schema (e.g., by ER modeling) determines entities and relationships, as well as their corresponding **sets of attributes** and associated **domains**
  - the **Cartesian product** of the respective domains is the set of all possible instances (of each entity type or relationship type)
  - a **relation** formalizes the **actually existing** subset of all possible instances

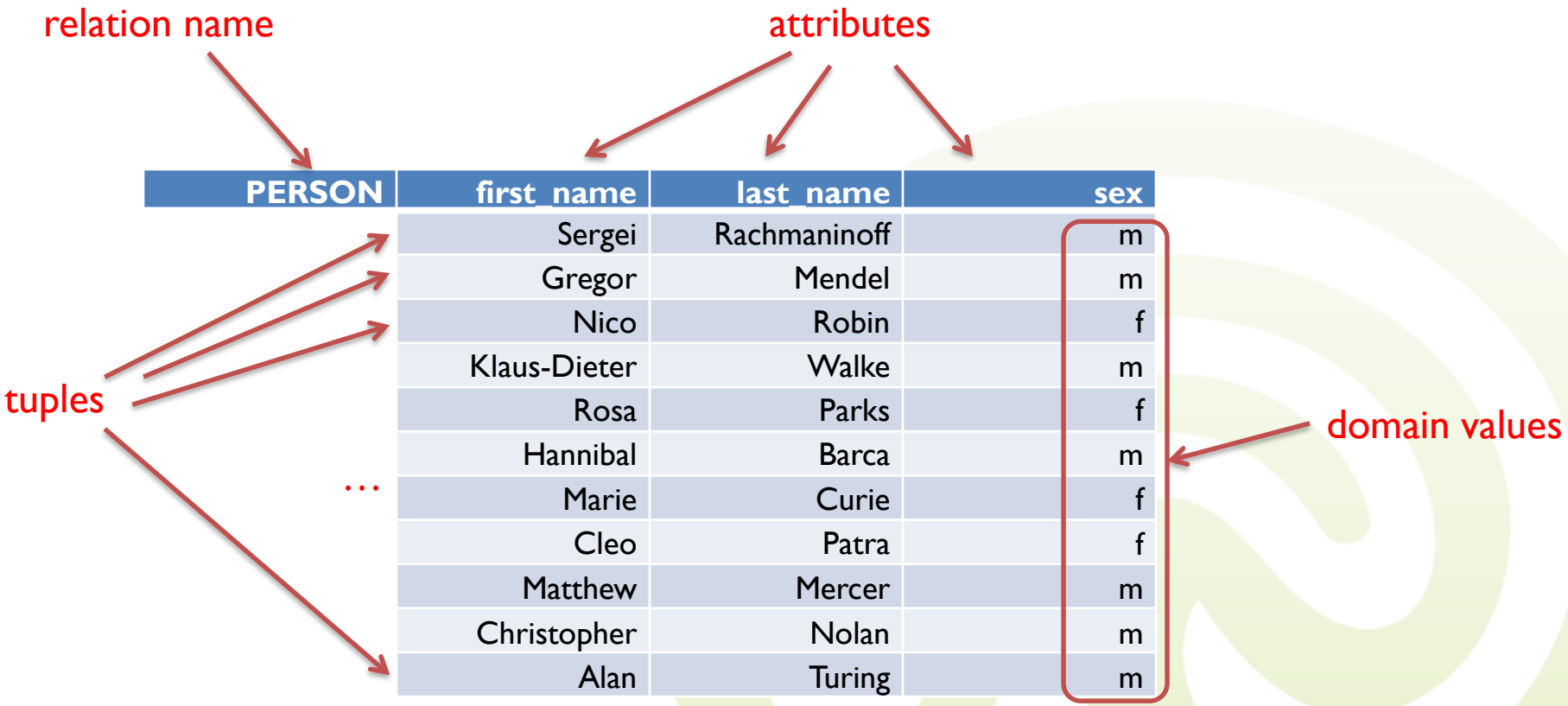
5.2 Relational Model

- Database schemas are described by **relation schemas** $R(A_1, ..., A_n)$
- Domains are assigned by the dom function
  - $dom(A_1) = D_1, dom(A_2) = D_2, ...$
  - Also written as: $R(A_1{:}D_1, ..., A_n{:}D_n)$
- The actual database instance is given by a set of matching **relations**
- Example
  - relation schema:
    $Cat(name: string, age: number)$
  - A matching relation:
    { (Blackie, 2), (Kitty, 1), (Fluffy, 4) }



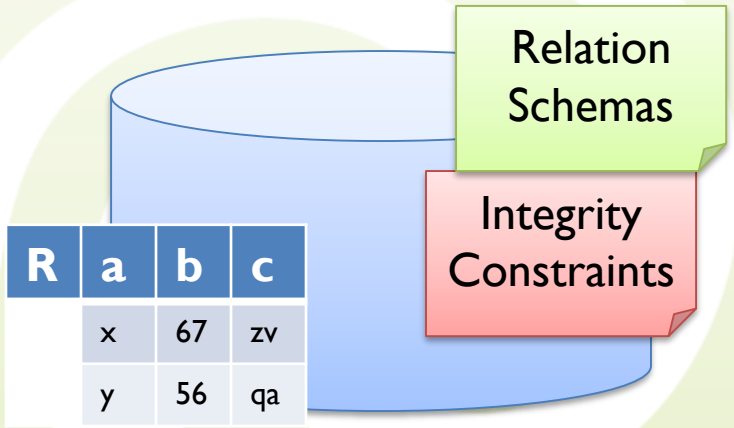
5.2 Relational Model

- Relations can be written as **tables**



5.2 Relational Model

- A **relational database schema** consists of
  - a set of relation schemas
  - a set of integrity constraints
- A **relational database instance** (or state) is
  - a set of relations adhering to the respective schemas and respecting all integrity constraints



5.2 Relational Model

- Every relational DBMS needs a language to define its relation schemas (and integrity constraints)
  - **Data Definition Language** (DDL)
  - typically, it is difficult to formalize all possible integrity constraints, since they tend to be complex and vague
- A relational DBMS also needs a language to handle and manipulate tuples
  - **Data Manipulation Language** (DML)
- Today's RDBMS use **SQL** as both DDL and DML
  - Compare to XML: Here, DDL and DML are separated

5 Relational Model

- Basic Set Theory
- Relational Model
- **From Theory to Practice**
- Integrity Constraints
- Conversion from ER

5.3 From Theory to Practice

Detour

- In the early 1970s, the **relational model** became a *hot topic* in database research
  - based on **set theory**
  - a relation is a subset of the **cartesian product** over a list of **domains**
- Early *query interfaces* for the relational model
  - **Relational Algebra**
  - **Tuple Relational Calculus (SQUARE, SEQUEL)**
  - **Domain Relational Calculus (QBE)**
- Question: **How to build a working database management system using this theory?**

5.3 From Theory to Practice

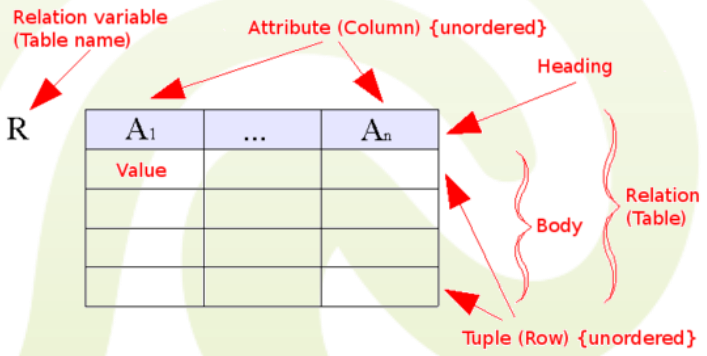
Detour

- **System R** was the first working prototype of a relational database system (starting 1973)
  - most **design decisions** taken during the development of System R substantially **influenced** the design of **subsequent systems**
- **Questions**
  - how to store and represent data?
  - how to query for data?
  - how to manipulate data?
  - how do you do all this with good performance?

5.3 From Theory to Practice

Detour

- The challenge of the **System R** project was to create a **working prototype system**
  - theory is good
  - but developers were willing to sacrifice theoretical beauty and clarity for the sake of usability and performance
- **Vocabulary change**
  - mathematical terms were too unfamiliar for most people
  - **table** = relation
  - **row** = tuple
  - **column** = attribute
  - **data type, domain** = domain



5.3 From Theory to Practice

Detour

- **Design decisions:**
  During the development of System R, two major and very controversial decisions had been made
  - allow **duplicate tuples**
  - allow **NULL values**
- Those decisions are still subject to discussions…

5.3 From Theory to Practice

Detour

- **Duplicates**
  - in a relation, there **cannot** be any **duplicate tuples**
  - also, query results cannot contain duplicates
    - the relational algebra and relational calculi all have implicit **duplicate elimination**

5.3 From Theory to Practice

Detour

- **Practical considerations**
  - you want to query for **name** and **birth year** of all **students** of TU Braunschweig
  - the result returns roughly **16000 tuples**
  - probably there are **some** duplicates
  - it's **1973,** and your computer has **16 kilobytes** of main memory and a very slow external storage device…
  - to eliminate duplicates, you need to **store** the result, **sort** it, and **scan** for adjacent duplicate lines
    - System R engineers concluded that this effort is not worth the effect
    - duplicate elimination in result sets happens only on-request

5.3 From Theory to Practice

Detour

- **Decision:** Don't eliminate duplicates in results
- What about the tables?
  - again: ensuring that no duplicates end up in the tables requires some work
  - engineers also concluded that there is actually no need in enforcing the no-duplicate policy
    - if the user wants duplicates and is willing to deal with all the arising problems – then that's fine
- **Decision:** Allow duplicates in tables
- As a result, the theory underlying relational databases shifted from **set theory** to **multi-set theory**
  - straightforward, only notation is more complicated

5.3 From Theory to Practice

Detour

- Sometimes, an attribute value is **not known** or an attribute does **not apply** for an entity
  - e.g. what value should the attribute *university_degree* take for the entity *Heinz Müller*, if Heinz Müller does not have any degree?
  - e.g. you regularly observe the weather and store temperature, wind strength, and air pressure every hour – and then your barometer breaks... what now?

5.3 From Theory to Practice

Detour

- Possible solution:
  For each domain, **define a value** indicating that data is not available, not known, not applicable, …
  - for example, use *none* for Heinz Müller's degree, use −1 for missing pressure data, ...
  - Problem:
    - you need such a special value for each domain or use case
    - you need special failure handling for queries, e.g. *compute average of all pressure values that are not −1*

5.3 From Theory to Practice

Detour

- Again, system designers chose the simplest solution (regarding implementation): **NULL values**
  - **NULL** is a special value which is usable in any domain and represents that data is just not there
    - there are many interpretations of what NULL actually means
  - Systems have some default rules how to deal with NULL values
    - aggregation functions usually ignore rows with NULL values (which is good in most, but not all cases)
    - three-valued logic
    - however, creates some strange anomalies

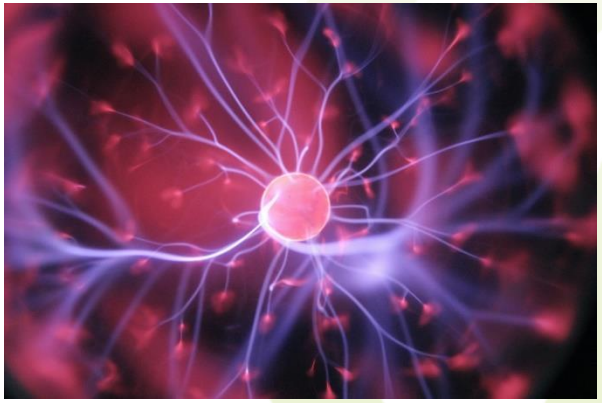
5.3 From Theory to Practice

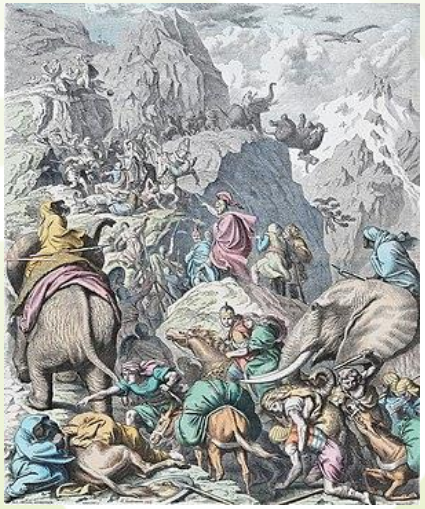
Detour

- Another tricky problem:
  How should users **query** the DB?
- Classical answer
  - Relational Algebra and Relational Calculi
  - **problem:** more and more **non-expert users**
- More *natural* query interfaces:
  - **QBE** (query by example)
  - **SEQUEL** (structured English query language)
  - **SQL:** the current standard; derived from SEQUEL

5 Relational Model

- Basic Set Theory
- Relational Model
- From Theory to Practice
- **Integrity Constraints**
- Conversion from ER





5.4 Integrity Constraints

- Integrity constraints are difficult to model in ER
  - basically annotations to the diagram, especially for **behavioral constraints**
    - **e.g.** *No allotment patch may have a larger area than the respective allotment.*
- But some **structural constraints** can directly be expressed
  - e.g., key constraints, functionalities
  - Formally, they are not part of the mathematical model, we still integrate them for practical purposes

5.4 Basic Constraints

- **Primary Key Constraint**
  - A relation is defined as a **set** of tuples
    - all tuples have to be **distinct**, i.e., no two tuples can have the same combinations of values for all attributes
    - so-called **uniqueness (unique key) constraint** or primary key constraint
  - Therefore, we can define the **key** of a relation as a designated **subset of attributes** for which no two tuples have the same values (are **unique**)
    - It's a little bit more complex than that…see lecture 10
  - Each relation will need a designated key
    - We will write this as for example **Hero**(alias, name, age, ...)

5.4 Basic Constraints

- **NOT NULL Constraint**
  - Remember, a relation is defined as $\boldsymbol{R \subseteq D_1 \times \ldots \times D_n}$ with tuples $t \in R$
  - However, in a practical application its common that not always all attribute values are known
    - Therefore, it is usually assumed that there is a special NULL value in each domain, i.e. $NULL \in D_i$
  - Sometimes, this is not desired for certain attributes
    - Introduces the NOT NULL constraint
  - **Primary Key must never be NULL**
  - e.g. Address(street: string NOT NULL,
    number: numeric NOT NULL,
    zip-code: numeric NOT NULL,
    city: string NOT NULL
    postbox : string)

5.4 Basic Constraints

- **Foreign Key Constraint**
  - Sometimes, we want to link tuples in different relations
    - This will be integral for realizing ER relationships in a database
  - A **foreign key constraint** can be defined between the key attributes of one relation and some attributes of another one

    References the key 'id' of member
    - e.g., Member(id, first name, last name)
      Nickname(nickname, memid→Member)
  - Tuples of the referring relation can only have values for the referencing attribute which are the key of an existing tuple in the referenced relation
    - This is called **referential integrity**

5.4 Basic Constraints

– Example:

Member(id, first name, last name)
Nickname(nickname, memid→Member)

Member

id	firstname	lastname
1	Gregor	Mendel
2	Hannibal	Barca
3	Matthew	Mercer
4	Alan	Turing

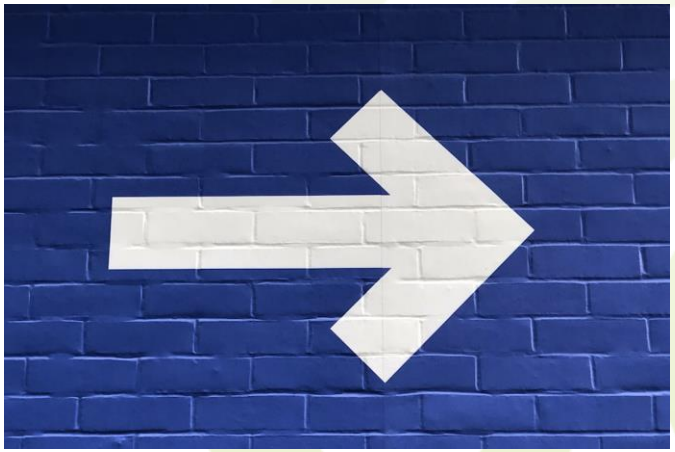
Nickname

nickname	memid
Bean Boy	1
Benedict Cucumberbatch	4
Benjamin Flower	2
Klaus	3
Matthias Merkur	3
Thunderbolt	2
Uluru Rocker	5

Invalid!

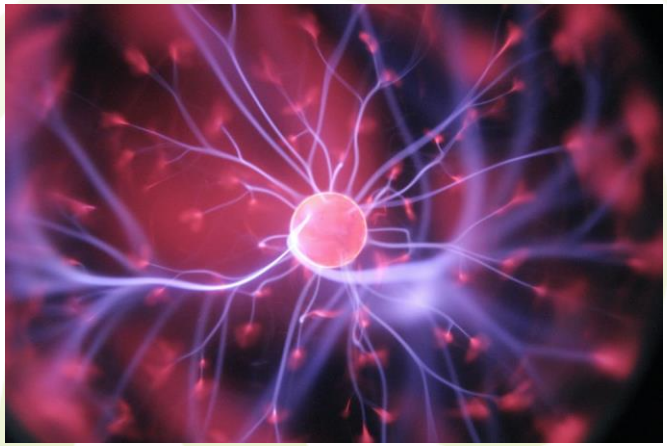
5.4 Basic Constraints

- Convention:
  - If a composite key is referenced, we write this as, e.g., $R1(\underline{a}, \underline{b}, c)$ $R2(d, \underline{e}, f, (d, f) \rightarrow R1)$
  - This is not a standard notation, but rather close to what you find in SQL



5.4 First Normal Form

- There is a another major **constraint** on the attributes' data types in the relational model
  - the value of any attribute must be **atomic,** that is, it **cannot be composed** of several other attributes
    - if this property is met, the relation is often referred to as a being in **first normal form** (1NF or minimal form)
    - in particular, **set-valued** and relation-valued attributes (tables within tables) are **prohibited**



5.4 First Normal Form

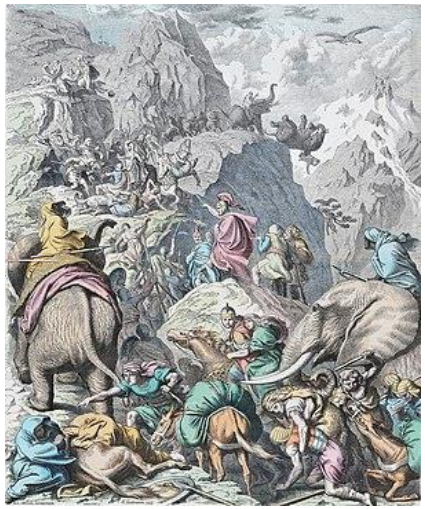
- Example of a **set-valued** column
  - *A person may own several telephones (home, office, cell, …).*

Person	first_name	last_name	telephone_no
	Gregor	Mendel	5555678
	Hannibal	Barca	{3914533, 3556576, 5463456}
	Matthew	Mercer	4543689
	Alan	Turing	7658736
	Rosa	Parks	{1252345, 8766781}

prohibited

5.4 First Normal Form

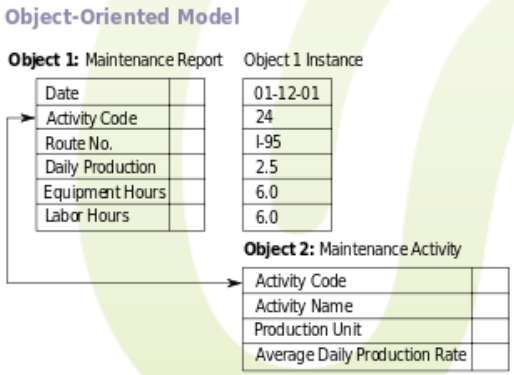
- Please note, it is possible to **model** composed attributes in ER models…
- To transform such a model into the relational model, a **normalization** step is needed
  - this is not always trivial, e.g., what happens to keys?



Person	first_name	last_name	telephone_no
	Gregor	Mendel	555-5678
	Hannibal	Barca	391-4533
	Hannibal	Barca	355-6576
	Hannibal	Barca	546-3456
	Matthew	Mercer	454-3689
	Alan	Turing	765-8736
	Rosa	Parks	125-2345
	Rosa	Parks	876-6781

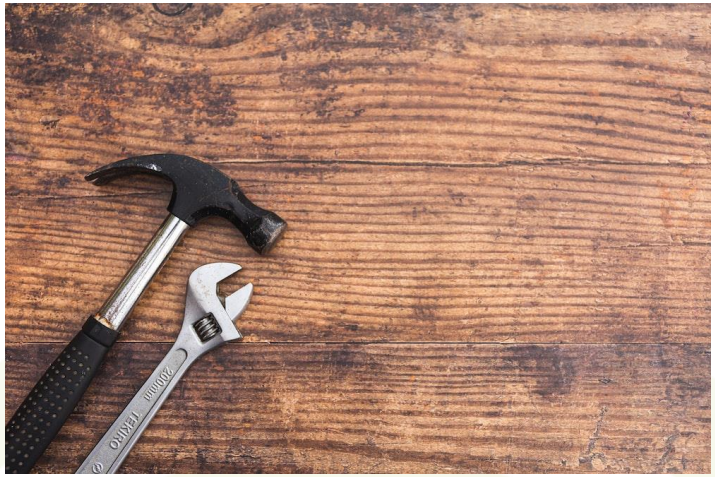
5.4 First Normal Form

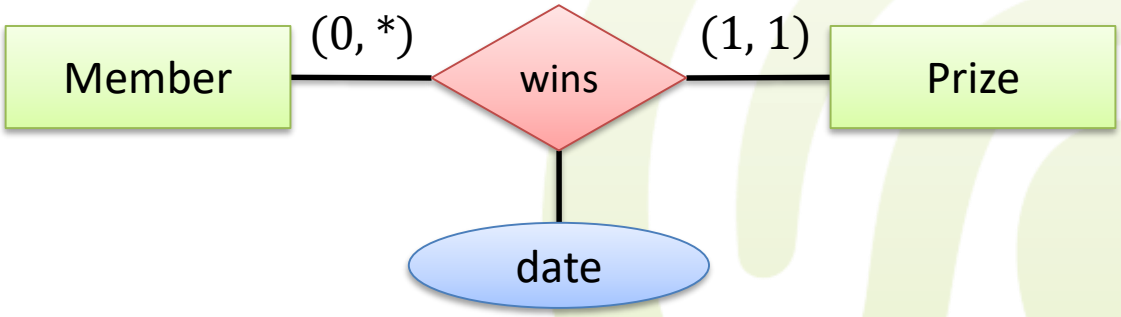
- In a purely relational database, all relations are in first normal form
  - **object-oriented** databases feature multi-valued attributes, thus closing the modeling gap
  - **object-relational extensions** integrate user-defined types (UDTs) into relational databases
    - Oracle from version 9i, IBM DB2 from version 8.1, …



5 Relational Model

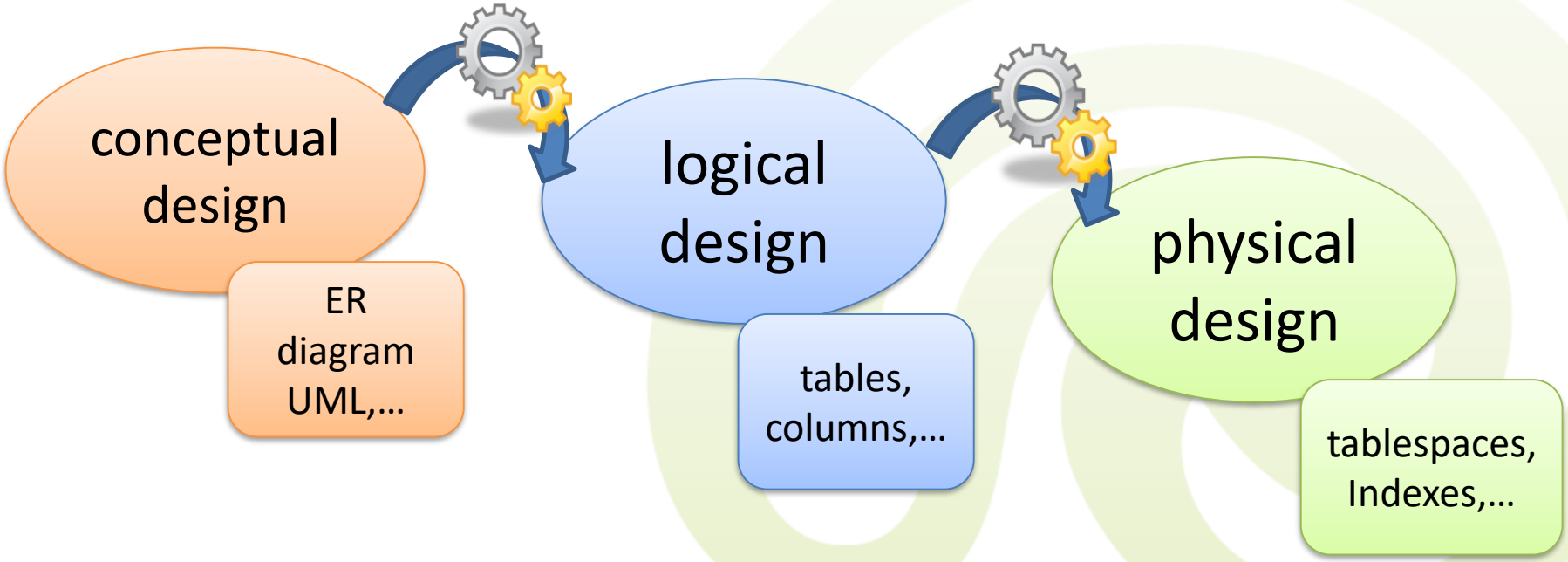
- Basic Set Theory
- Relational Model
- From Theory to Practice
- Integrity Constraints
- **Conversion from ER**





5.5 Conversion from ER

- After modeling a conceptual schema (e.g., using an **ER diagram**), the schema can be (semi-) **automatically** transformed into a **relational schema**

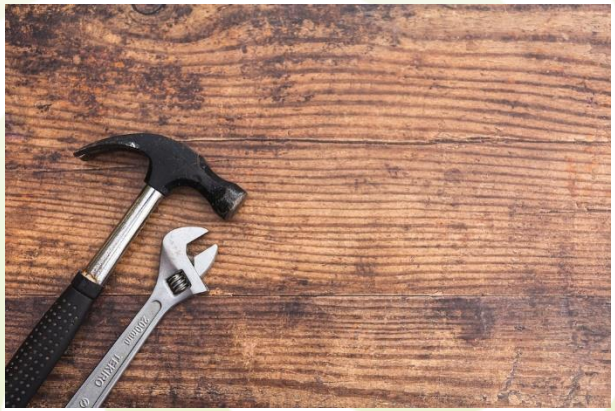


5.5 Conversion from ER

- The ER diagram is **semantically richer** than the relational model
  - However, its not a real subset
- Many constraints are very hard/impossible to express
  - disjoint/overlapping generalization
  - non-trivial cardinalities – like (11,20), (3,3) or even (1,*)
  - ...
- Therefore, it usually is a really good idea to create an ER diagram before coding a logical schema

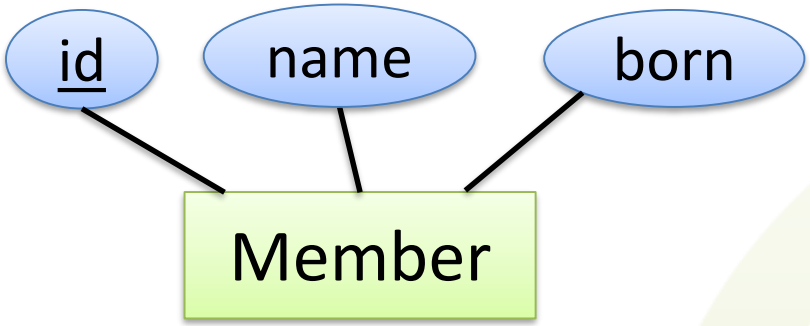
5.5 Conversion from ER

- Question: How to convert EER to relation?
  - We can automatically convert a conceptual ER model to relations
    - Some heuristics follow…
    - However, quite often the result will not be as desired
    - Therefore, still some manual optimization and steering is beneficial
    - While designing a model, it might be very beneficial to keep the result relations and the desired queries in mind…



5.5 Conversion from ER

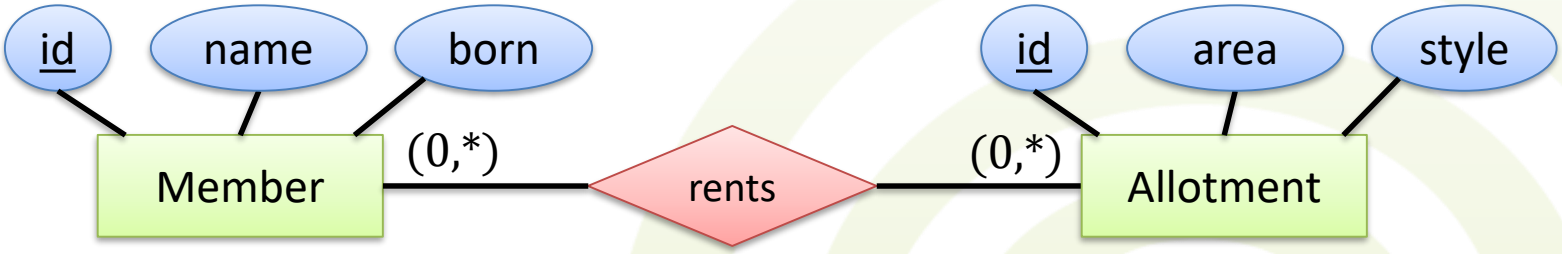
- Converting a simple Entity Type into a relation schema:



Member(id, name, born)

5.5 Conversion from ER

- Converting an n:m relationship type into a relation schema:
  - Relationship type becomes a separate relation schema
    - Links entities of the respective types by using their foreign keys



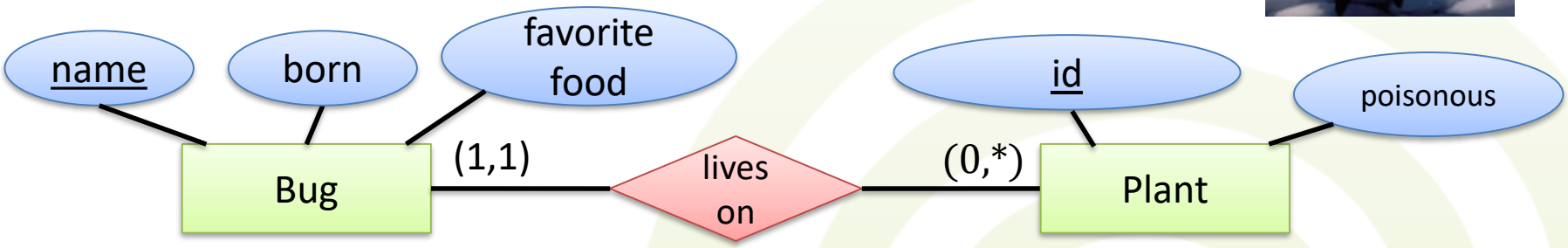
Member(id, name, born)

Allotment(id, area, style)

Member_rents_allotment(member → Member, allotment→Allotment)

5.5 Conversion from ER

- Converting an 1:m relationship type a relation schema:
  - Entity Type at 1-side can only participate once at the relationship type
    => Push relationship type to the 1-side

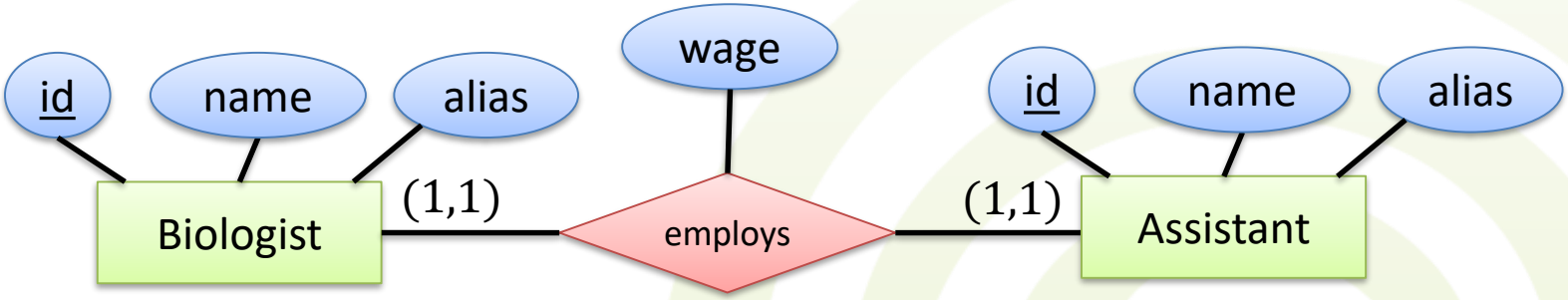


Plant(id, poisonous)

Bug(name, born, favourite_food, real_estate→Plant)

5.5 Conversion from ER

- Converting an 1:1 relationship type a relation schema:
  - A little bit tricky...
    - Cannot be expressed just by the relation schemas...
    - Just choose one side as the 1-side and implement it just like a 1:m relationship type

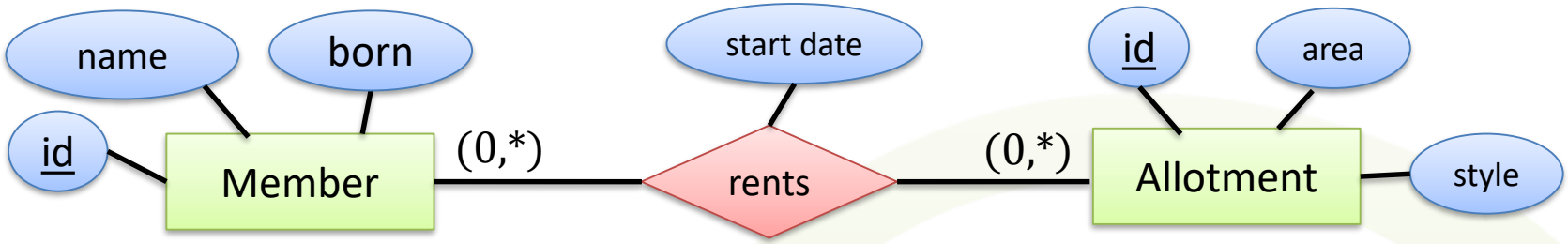


Biologist(id, name, alias) **Assistant**(id, name, alias, wage, **supervisor** → **Biologist**)

- To check if it really 1:1 we need advanced constraints
  - e.g. by using triggers (to be introduced in lecture 12)

5.5 Conversion from ER

- How to deal with attributes attached to the rel. type
  - Put them wherever you put the respective foreign key(s)



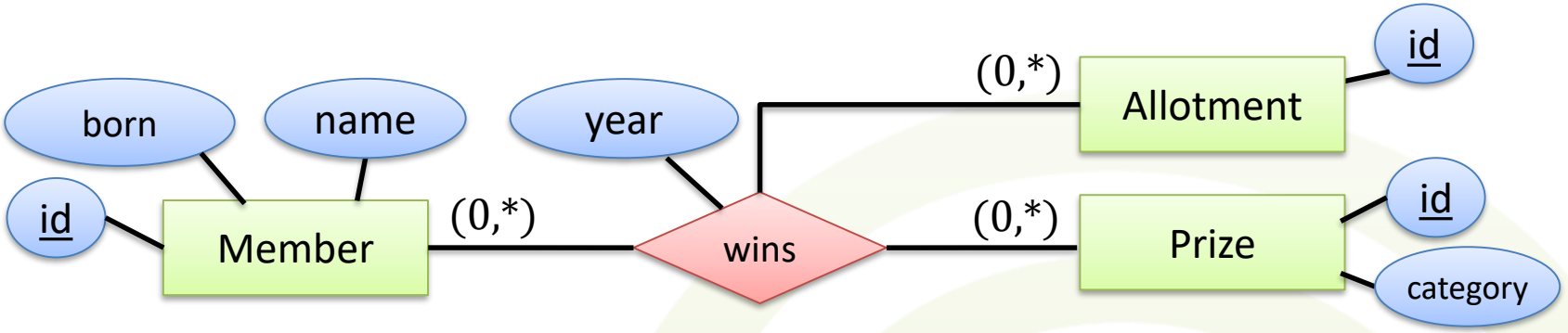
Member(id, name, born)

Allotment(id, area, style)

Member_rents_allotment(
member → Member,
allotment → Allotment,
start_date
)

5.5 Conversion from ER

- What about n-ary relationship types? (n>2)
  - Just apply the exact same approaches:

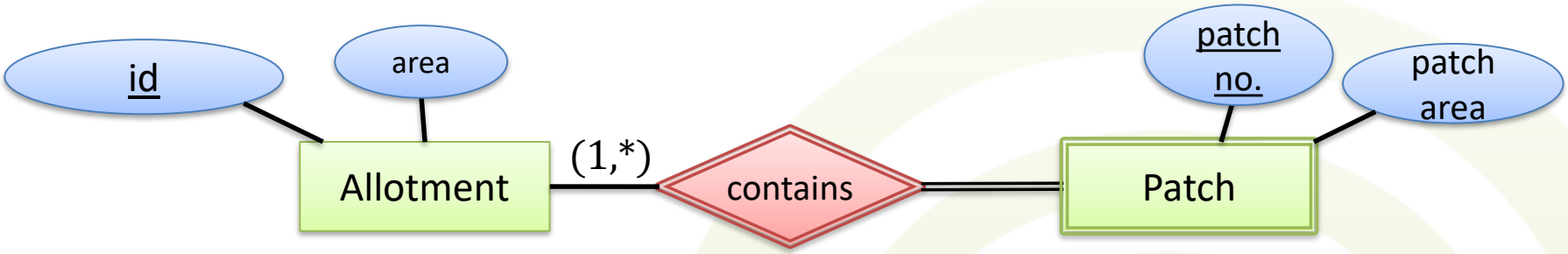


mem_wins_prz_for_allot(
member → Member,
prize → Prize,
allotment → Allotment
year)

5.5 Conversion from ER

- Converting a weak entity into a relation schema:
  - Weak entities are only unique together with the entity at the **identifying relationship**

  => Follow ident. rel. and inherit respective foreign keys

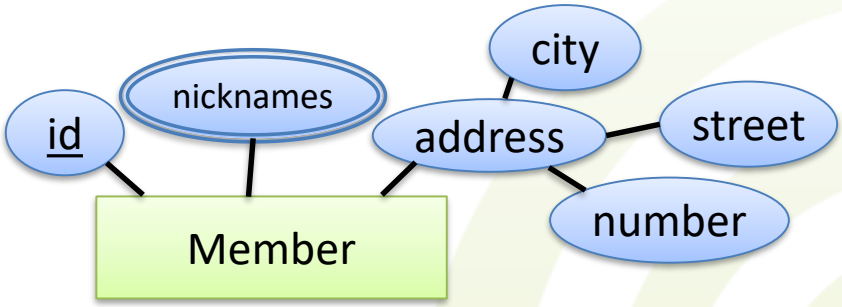


Allotment(id, area)

allotment_patch(patch_no, allot_id→ Allotment, patch_area)

5.5 Conversion from ER

- How to deal with multi-attributes and composite attributes
  - **composition:** just flatten it
  - **multi-attribute:** treat it like a weak entity

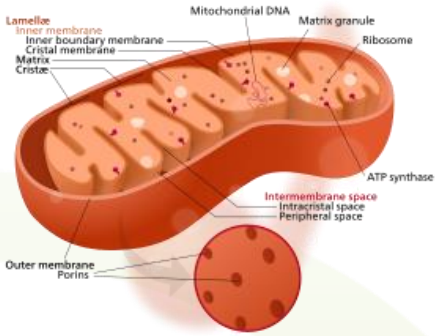


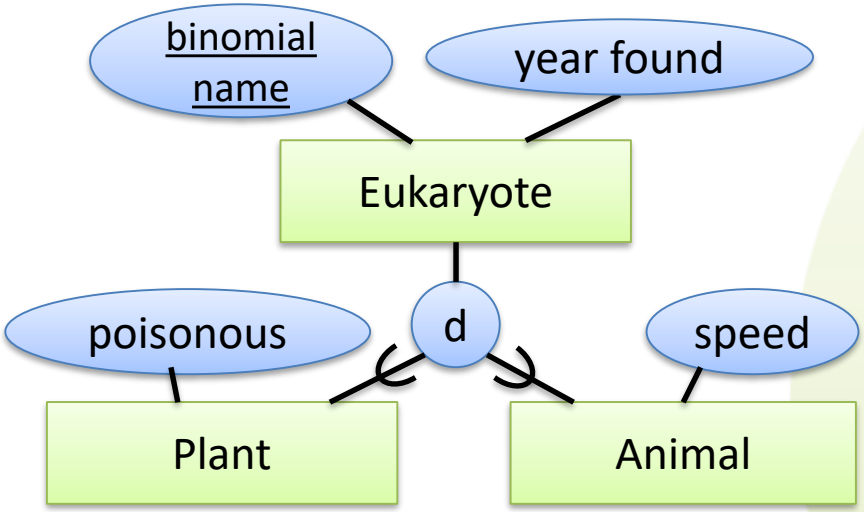
Member(<u>id</u>, addr_city, addr_street, addr_number)

member_nicks(<u>member</u> → Member, <u>nickname</u>)

5.5 Conversion from ER

- Converting types with inherited attributes/ relations into a relation schema:
  - Can be implemented in many ways (depending on inheritance type)
  - **Most generic way:** Inherit foreign keys from super type





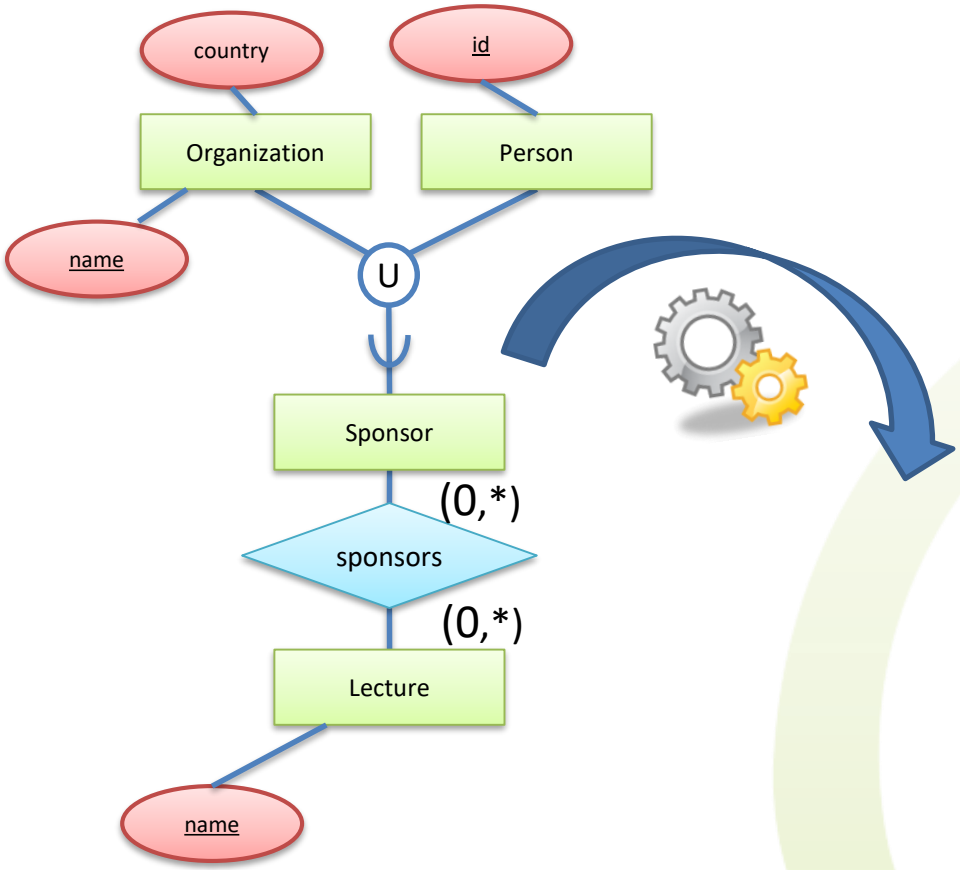
Eukaryote(binomial name, year_found)

Plant(eukaryote → Eukaryote, poisonous)

Animal(eukaryote → Eukaryote, speed)

5.5 Conversion from ER

- Union-Type:



Sponsor(newkey)

Organization
(name,
country,
newkey → **Sponsor**)

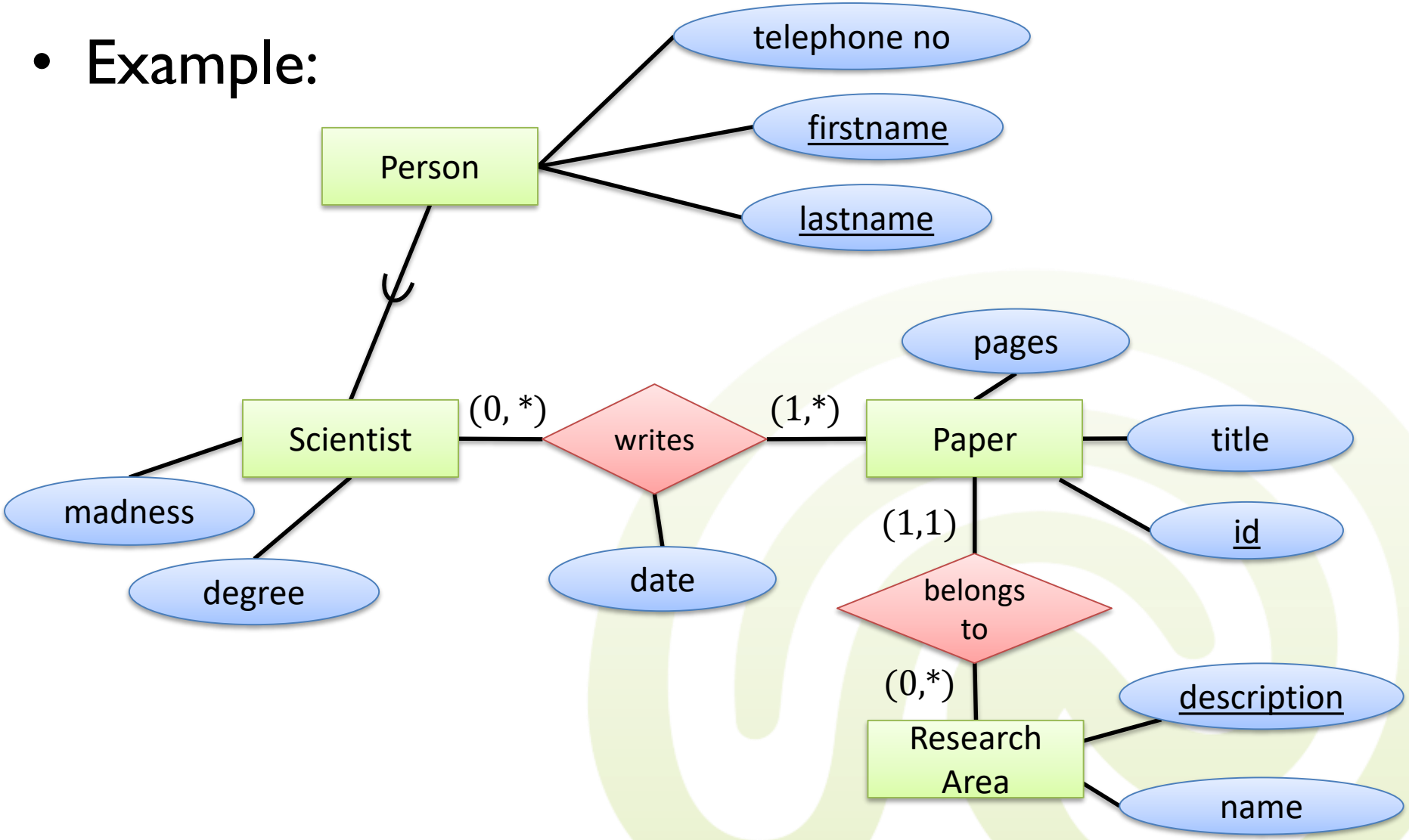
Person
(id,
newkey → **Sponsor**)

Lecture(name)

lecture_sponsored_by
((newkey) → **Sponsor**,
(name) → Lecture)

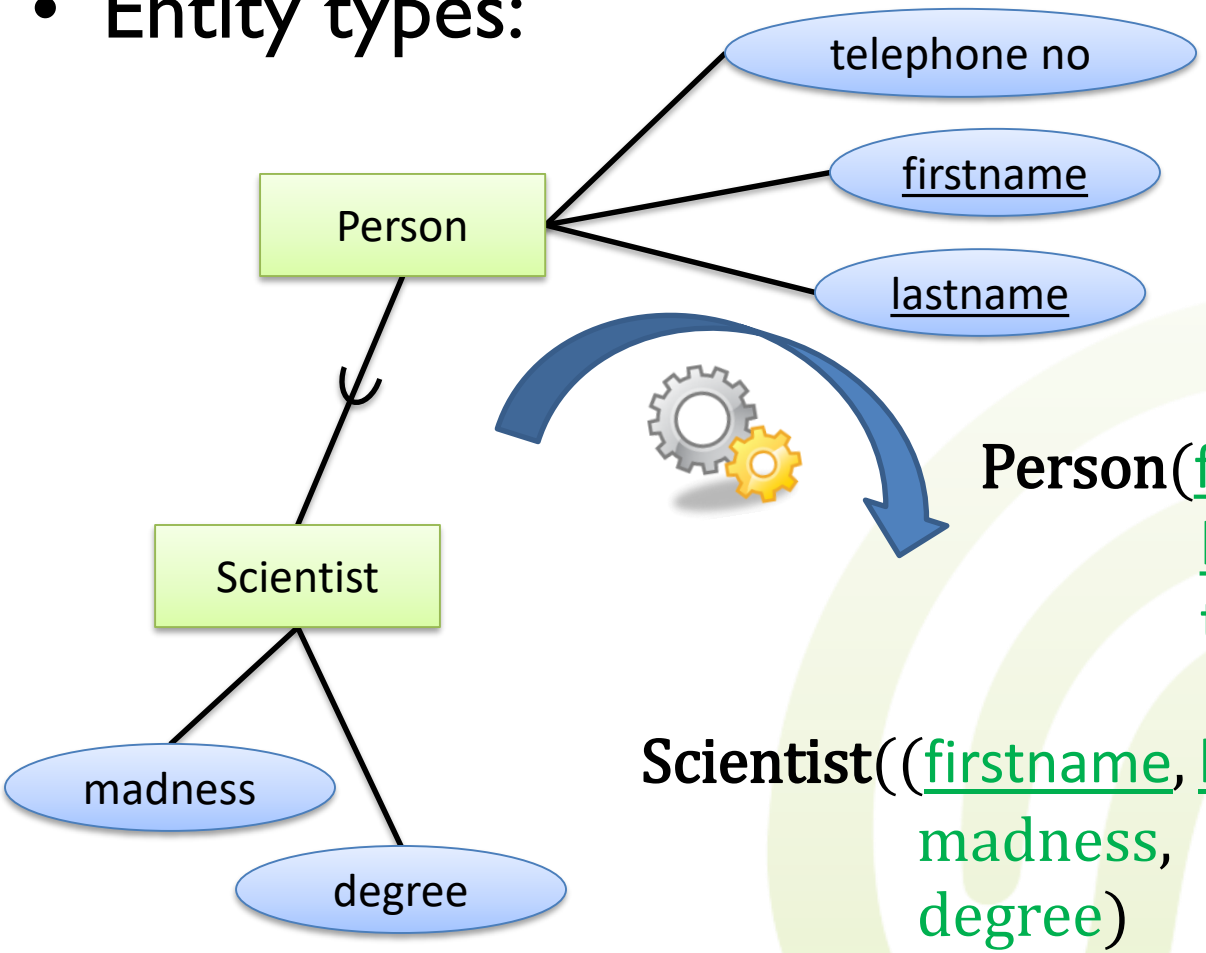
5.5 Conversion from ER

- Example:



5.5 Conversion from ER

- Entity types:

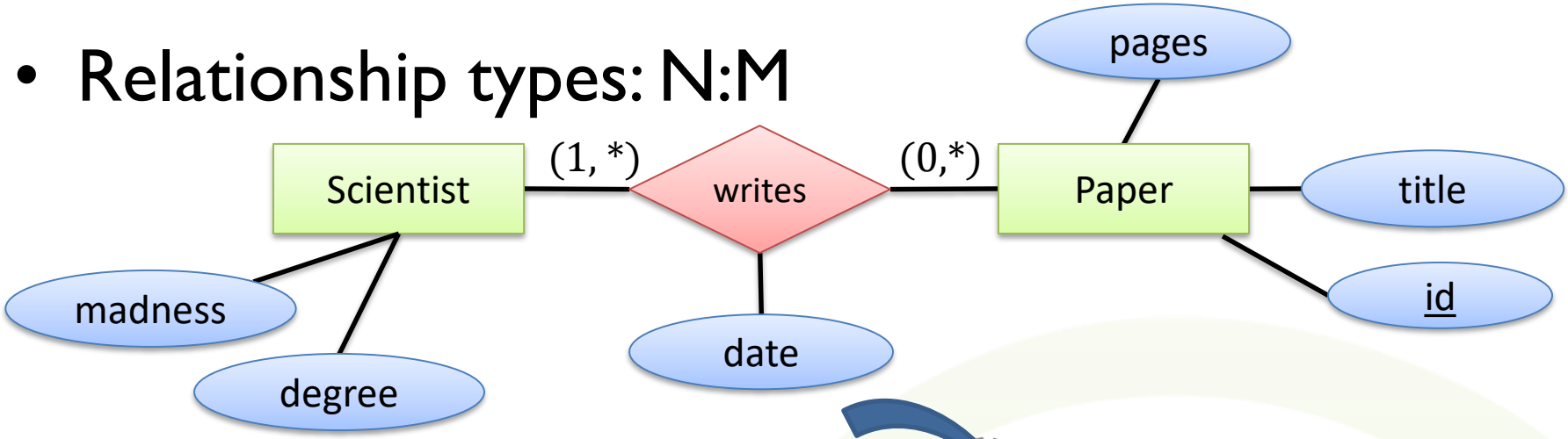


Person(firstname,
lastname,
telephone_no)

Scientist((firstname, lastname) → Person,
madness,
degree)

5.5 Conversion from ER

- Relationship types: N:M



Paper(id, pages, title, ...)

writes((id) → Paper
(fistname, lastname) → Scientist,
date)

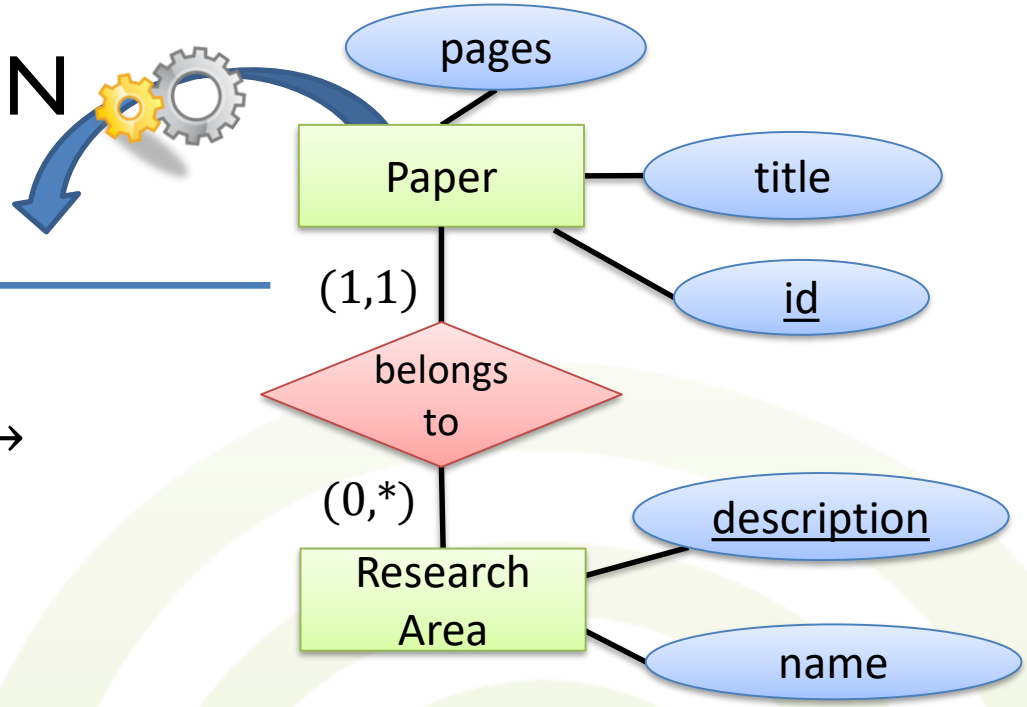
5.5 Conversion from ER

- Relationship types: 1:N

Paper(id, pages, title)

Paper(id, pages, title, field → ResearchArea)

Research Area(description, name)



6 Next Lecture

- Relational Algebra
  - Basic relational algebra operations
  - Additional derived operations
- Query Optimization
- Advanced relational algebra
  - Outer Joins
  - Aggregation