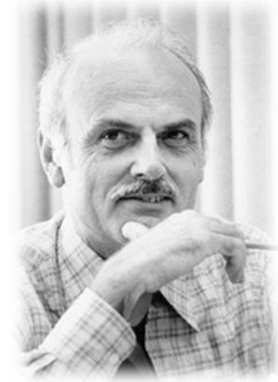
Institut für Informationssysteme
Technische Universität Braunschweig

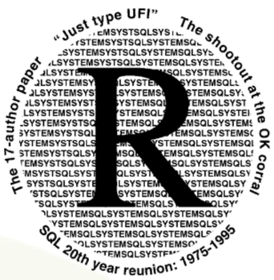
Relational Database Systems 1

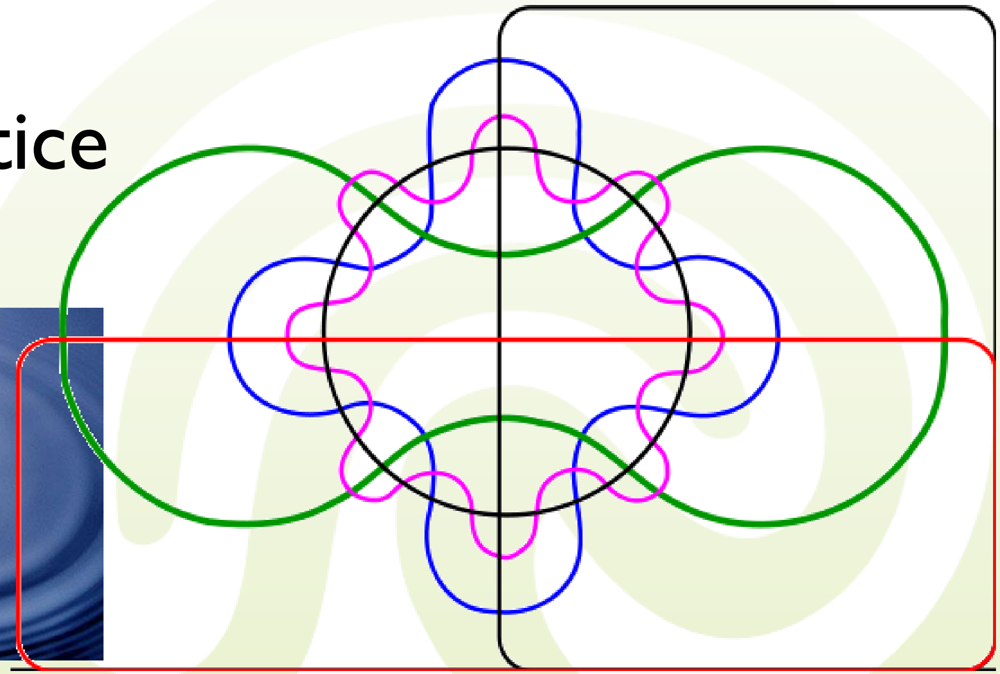
Wolf-Tilo Balke
Simon Barthel
Institut für Informationssysteme
Technische Universität Braunschweig
www.ifis.cs.tu-bs.de

Overview

- Basic set theory
- Relational data model
- Transformation from ER
- Integrity Constraints
- From Theory to Practice







5.1 Basic Set Theory

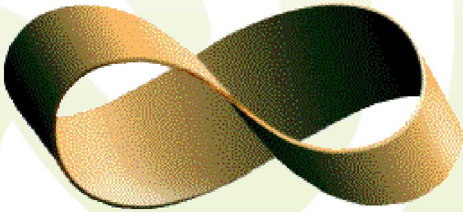
- **Set theory** is the foundation of mathematics
  - You probably all know these things from your math course, but repeating never hurts
  - The **relational model** is based on set theory; understanding the basic math will help a lot

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

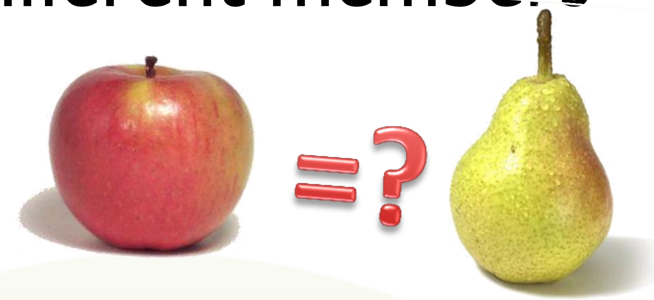
5.1 Sets

- Sets can be specified **extensionally**
  - List all its elements
  - Example: 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}

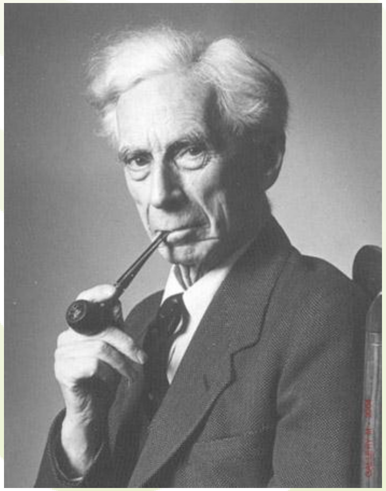


- Sets can be empty (written as {} or ∅)
- Notations for set membership:
  – a ∈ {a, b, c}
  – e ∉ {a, b, c}

5.1 Sets

- Defining a set by its **intension:**
  - Intension must be **well-defined** and **unambiguous**
  - There always is a clear **membership criterion** to determine whether an object belongs to the set (or not)
  - Not a valid 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!
- As we will see later:
  - Intension ≈ Database query
  - Extension ≈ Result of a query

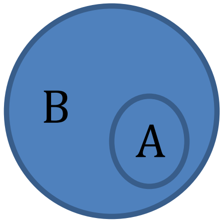
5.1 Sets

- For every set, there is an accompanying definition of **equality** (or equivalence)
  - Is $x = y$?
  - If they are equal, they are actually just one element
- 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)

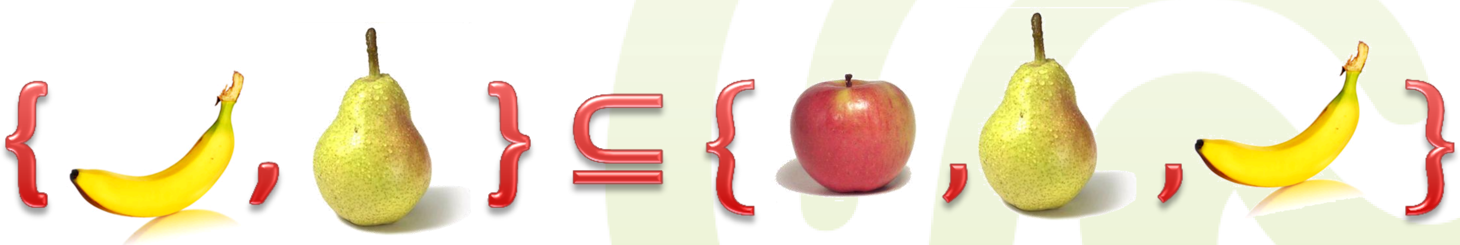
5.1 Sets

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

$A \subseteq B$



- 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 $\langle a, b, c\rangle$ or (a, b, c)

5.1 Tuples

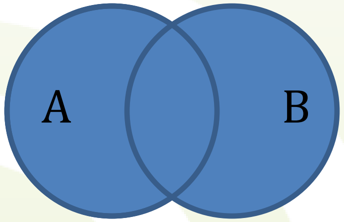
- Hence:
  - $<a, b, c> \neq <c, b, a>$, whereas $\{a, b, c\} = \{c, b, a\}$
  - $<a_1, a_2> = <b_1, b_2>$ iff $a_1 = b_1$ and $a_2 = b_2$

- ***n*-tuples** ($n > 2$) can also be defined as a cascade of ordered pairs:
  - $<a, b, c, d> = <a, <b, <c, d>>>$

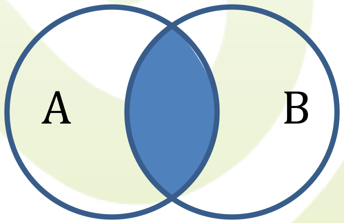
5.1 Set Operations

- 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
  - $\{a, b\} \cap \{b, c\} = \{b\}$

A ∪ B

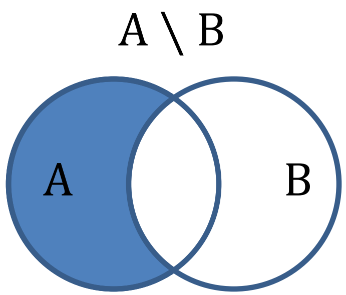


A ∩ B



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: ×
  - The Cartesian product is an operation between two sets, creating a new set of pairs such that:

$$A \times B = \{<a, b> \mid a \in A \text{ and } b \in B\}$$

  - Named after René Descartes

- Example:
  - {a, b} × {b, c} = {<a, b>, <a, c>, <b, b>, <b, c>}
  - Cleverness = { genius, dumb }
  - Character = { hero, villain }
  - Cleverness × Character = {<genius, hero>,
    <dumb, hero>,
    <genius, villain>,
    <dumb, villain>}

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

5.1 Relations

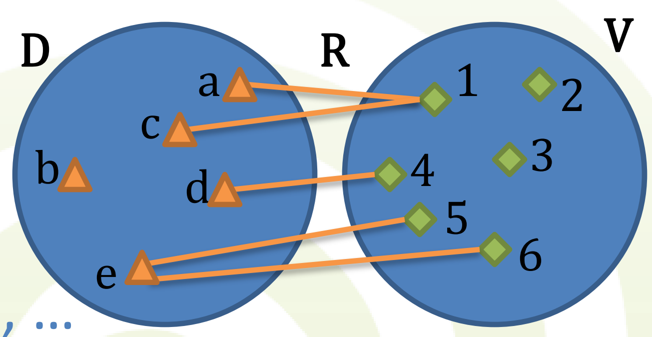
- A **relation** R over some sets $D_1, \ldots, D_n$ is a **subset** of their **Cartesian** product
  - $R \subseteq D_1 \times \ldots \times D_n$
  - The elements of a relation are **tuples**
  - The D_i are called **domains**
  - Each D_i corresponds to an **attribute** of a tuple
    - n=1: Unary 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 **list 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 = {<a,1>, <c,1>, <d,4>, <e,5>, <e,6>}
    - Can also be written as a**R**1, c**R**1, d**R**4, …
  - Imagine **Likes ⊆ Person × Beverage**
    - Joachim Likes Coffee, Tilo Likes Tea, …



5.1 Relations

- Example:
  - accessory = {spikes, butterfly helmet}
  - material = {silk, armor plates}
  - color = {pink, black}

color × material × accessory =
{<pink, silk, butterfly helmet>,
<pink, silk, spikes>,
<pink, armor plates, butterfly helmet>,
<pink, armor plates, spikes>,
<black, silk, butterfly helmet>,
<black, silk, spikes>,
<black, armor plates, butterfly helmet>,
<black, armor plates, spikes>}

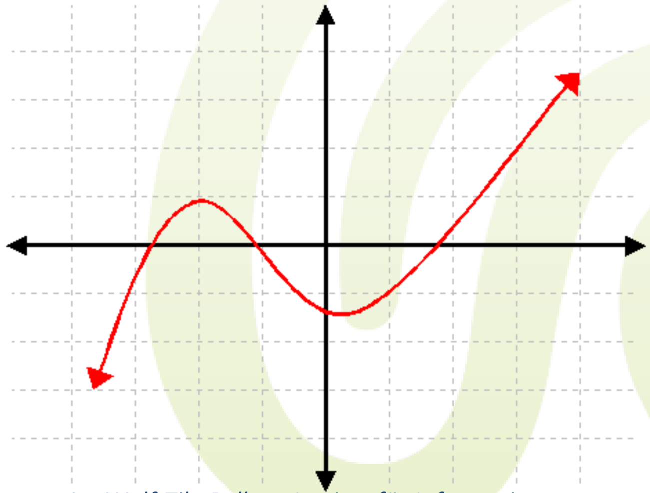
5.1 Relations

- Relation **FamousHeroCostumes** **⊆ color × material × accessory**

FamousHeroCostumes =
{<pink, silk, butterfly helmet>,
<black, armor plates, spikes>}

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 $\{A_1, \ldots, A_n\} \rightarrow D_1 \cup \ldots \cup D_n$
    - That means, every tuple maps each attribute to some value

5.1 Functions

- Example:
  - color = {pink, black}
  - material = {silk, armor plates}
  - accessory = {spikes, butterfly helmet}
  - The tuple <pink, silk, butterfly helmet>
    can also be represented as the following function *t*:
    - *t*(color) = pink
    - *t*(material) = silk
    - *t*(accessory) = butterfly helmet
  - Usually, one writes t[color] instead of t(color)

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 attributes and domains
  - A **database instance** is a set of objects 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**, denoted by $R(A_1:D_1, \ldots, A_n:D_n)$
- The actual database instance is given by a set of matching **relations**
- Example
  - Relation schema:
    CATS(name : varchar(10), age : integer)
  - A matching relation:
    { (Blackie, 10), (Pussy, 5), (Fluffy, 12) }

5.2 Relational Model

- Relations can be written as **tables:**

relation name

attributes

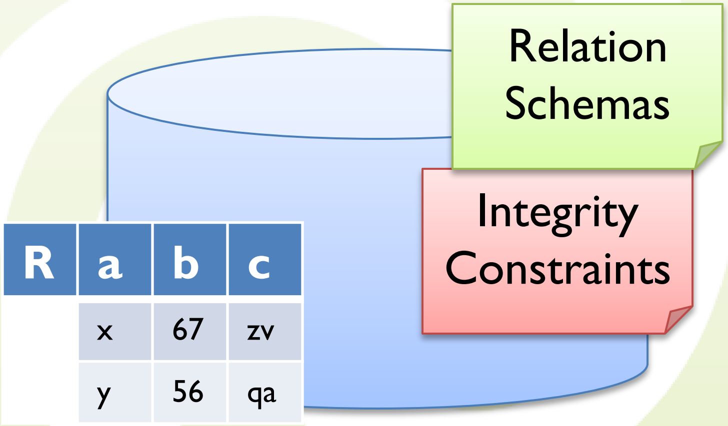
tuples

domain values

PERSON	firstName	lastName	sex
	Clark Joseph	Kent	m
	Louise	Lane	f
	Lex	Luthor	m
	Charles	Xavier	m
	Erik	Magnus	m
	Jeanne	Gray	f
…	Ororo	Munroe	f
	Tony Edward	Stark	m
	Matt	Murdock	m
	Raven	Wagner	f
	Robert Bruce	Banner	m

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

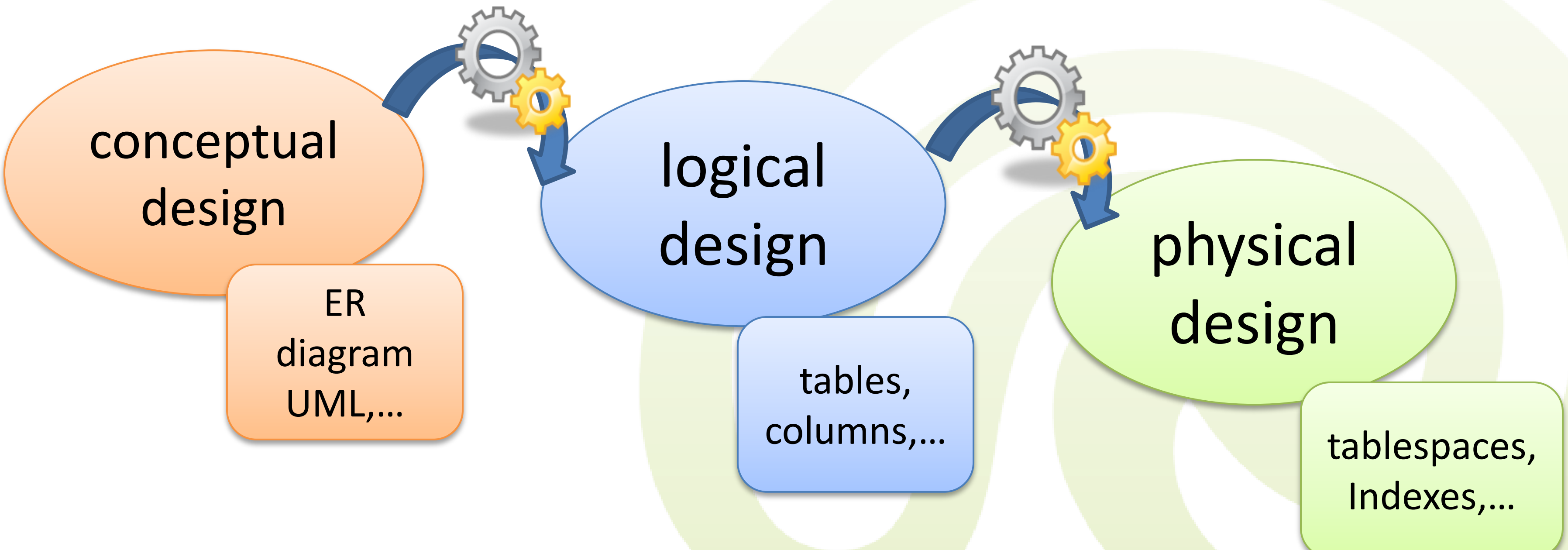


5.2 Relational Model

- 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 tuples
  - **Data manipulation language** (DML)
- Today's RDBMS use **SQL** as both DDL and DML

5.3 Conversion from ER Detour

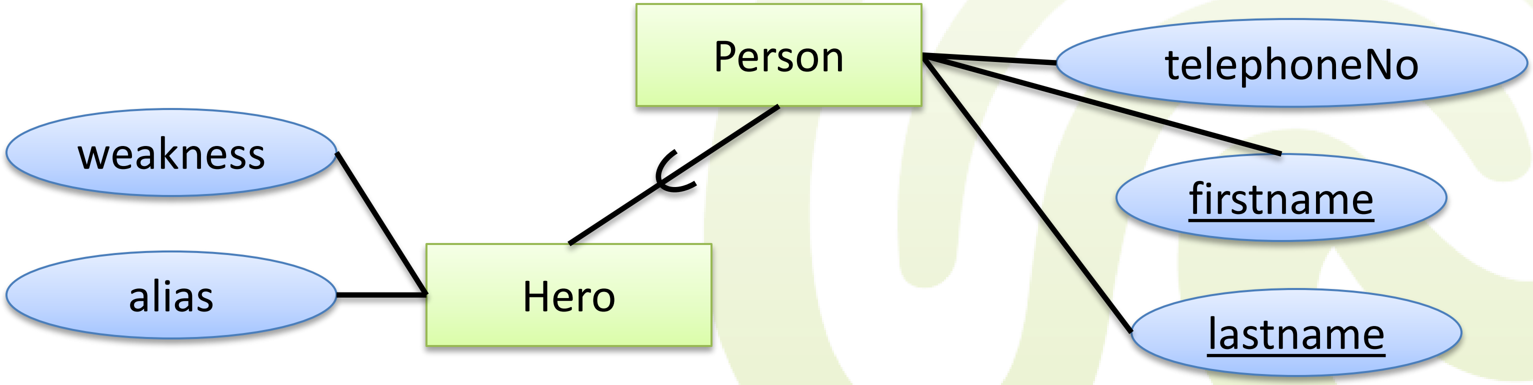
- After modeling a conceptual schema (e.g., using an **ER diagram**), the schema can be **automatically** transformed into a **relational schema**
- Remember:



5.3 Conversion from ER

Detour

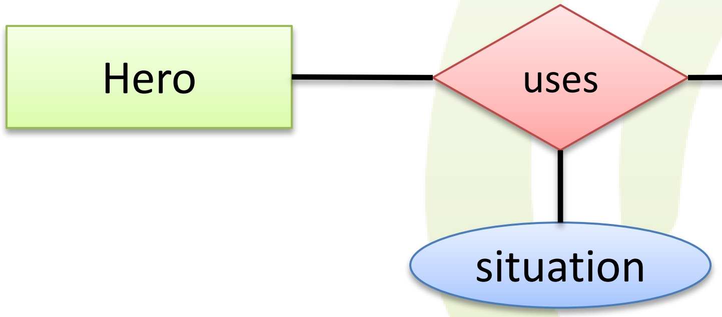
- Each **entity type** E with attributes $\mathrm{A}_1, \ldots, \mathrm{A}_n$ from domains $\mathrm{D}_1, \ldots, \mathrm{D}_n$ is converted into an ***n*-ary relation schema** $\mathrm{E}(\mathrm{A}_1{:}\mathrm{D}_1, \ldots, \mathrm{A}_n{:}\mathrm{D}_n)$
- If there is a relationship type E is_a F involved (**specialization**), the inheritance relationship can be expressed by copying all key attributes from F



5.3 Conversion from ER

Detour

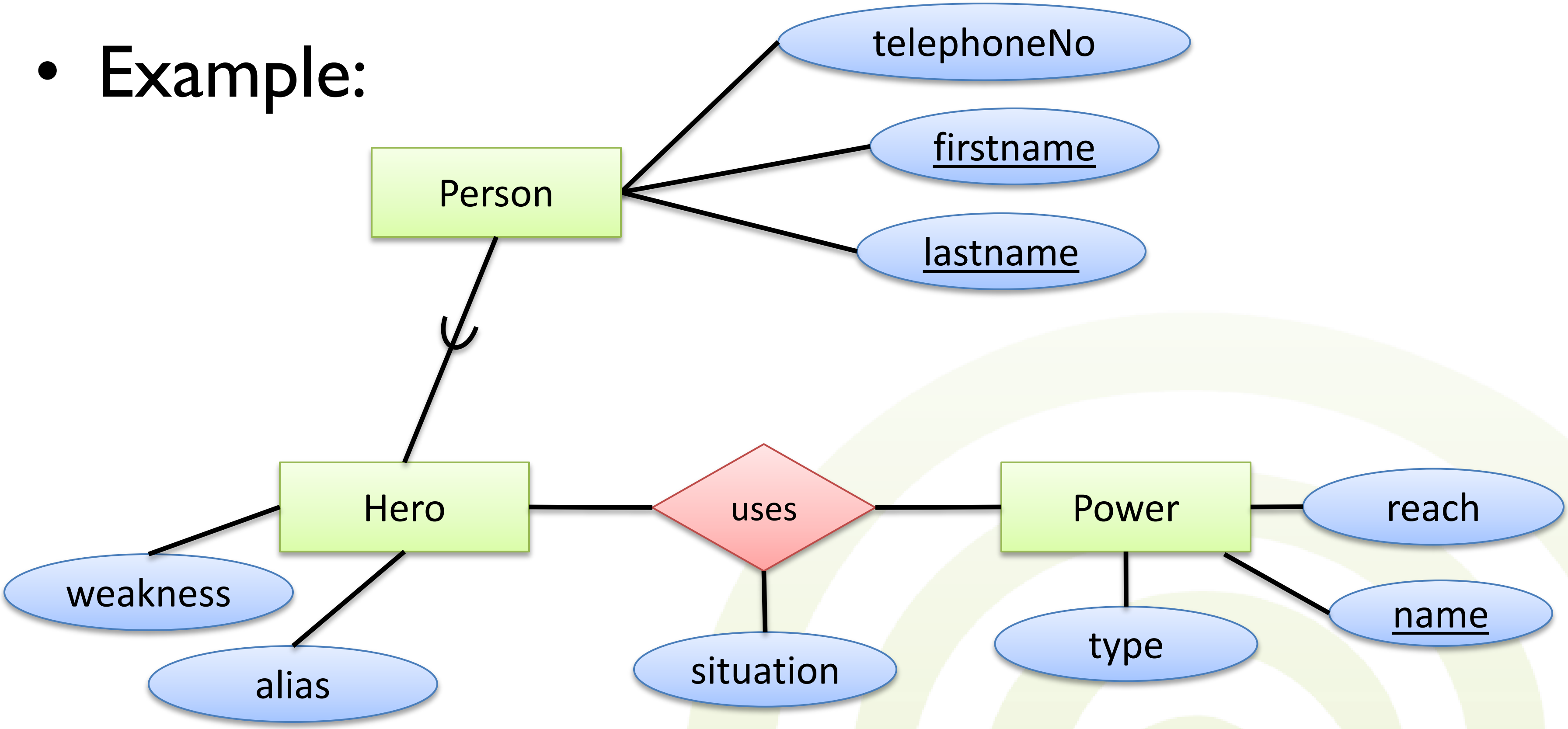
- A **relationship type** R between entity types $E_1, \ldots, E_n$ is converted to a relation schema whose attributes are all the key attributes of E_i
  - If keys share the same name, they have to be renamed
- If the relationship type has **own attributes** they are also copied to the relation schema



5.3 Conversion from ER

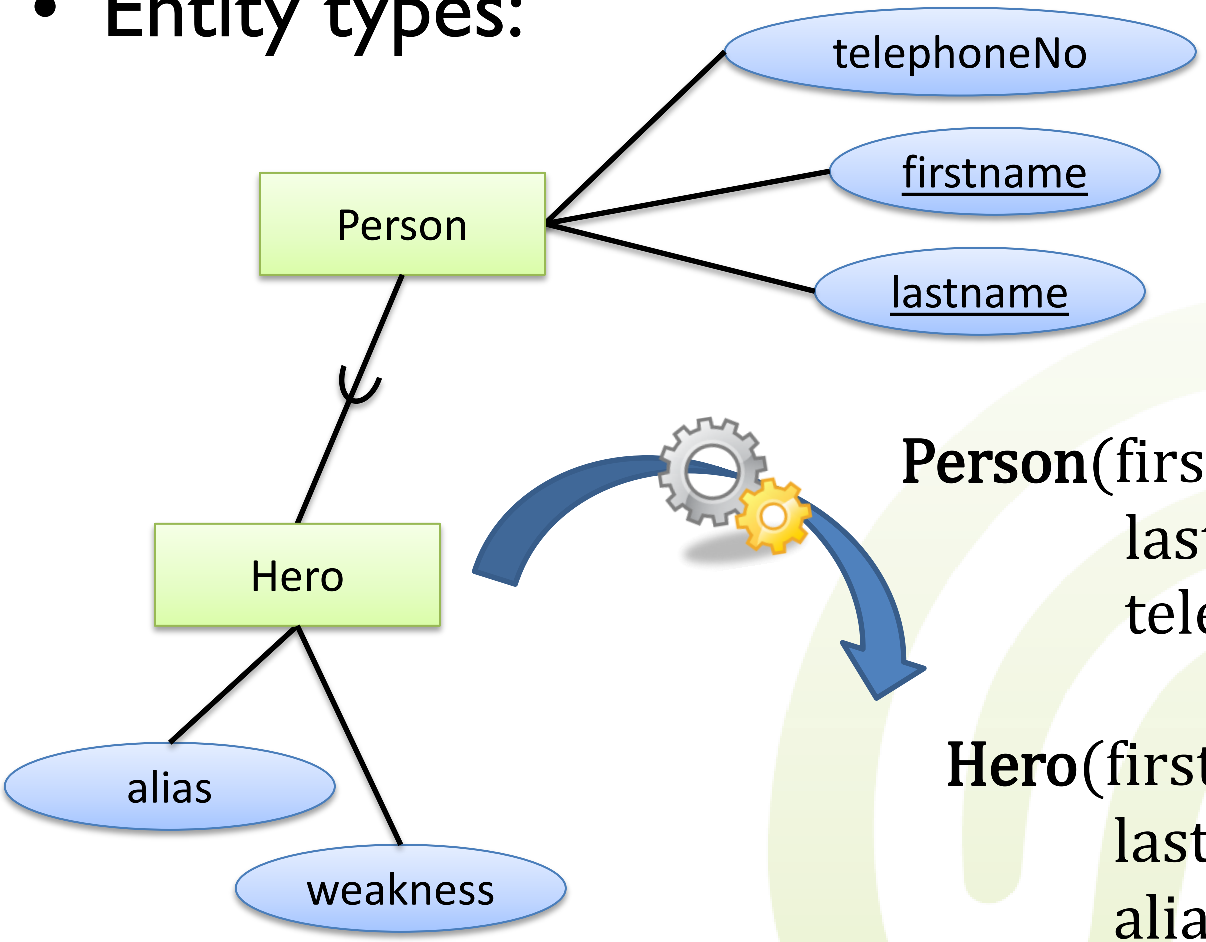
Detour

- Example:



5.3 Conversion from ER Detour

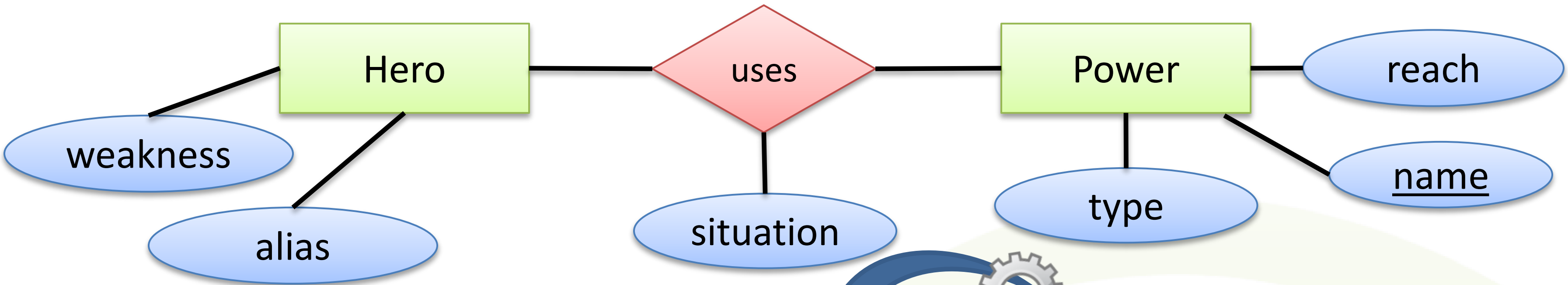
- Entity types:



Person(firstname :Varchar(20),
lastname : Varchar(30),
telephoneNo : Integer)

Hero(firstname : Varchar(20),
lastname : Varchar(30),
alias : Varchar(20),
weakness : Varchar(10))

- Relationship types:



Hero(firstname : Varchar(20),
lastname : Varchar(30),
alias : Varchar(20),
weakness : Varchar(10))

Power(name :Varchar(20),
type : Varchar(20),
reach : Integer)

Uses(firstname :Varchar(20),
lastname : Varchar(30),
name : Varchar(20),
situation : Varchar(30))

5.3 Conversion from ER

Detour

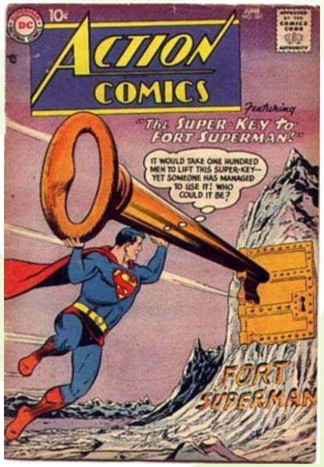
- Note: The ER diagram is **semantically richer** than the relational model
- Examples:
  - No key constraints and functionalities yet
  - Integrity constraints like disjoint/overlapping generalization cannot be expressed
  - …
- Therefore, it usually is a really good idea to create an ER diagram before coding a logical schema

5.4 Integrity Constraints

- Integrity constraints are difficult to model in ER
  - Basically annotations to the diagram, especially for **behavioral constraints**
    - **Example:** The popularity rating of any assistant should always be less than the respective professor's
- But some **structural constraints** can directly be expressed
  - Key constraints
  - Functionalities

5.4 Inherent Constraints

- 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**
- Any subset of a relation type's attributes is called a **superkey,** if any two tuples can never share the same values with respect to this subset
  - The set of all attributes is always a superkey, but there may be smaller sets
  - Superkeys may have redundant attributes



5.4 Inherent Constraints

- A minimal superkey is called a **key**
  - "Minimal" means that no attribute can be removed without losing the superkey property
  - Of course, a relation can have several keys
  - The key property is determined from the **semantics** of the attributes, not from the current data instances
  - Example:
    - Relation **address**(street, number, zip code, city)
    - Keys:
      {street, number, zip code} and {street, number, city}

5.4 Inherent Constraints

- The set of keys of some relation R is called **candidate key set** or cand(R)
- For each relation, a single candidate key has to be chosen to identify tuples in the relation, the so-called **primary key**
  - Analogously to ER diagrams, the chosen primary key is often underlined in the relation schema
    - Example: **address**(<u>street</u>, <u>number</u>, zip-code, <u>city</u>)
  - Though any candidate key can be chosen, it is usually better to choose a primary key with a **small** number of attributes

5.4 Inherent Constraints

- Assume that an ER diagram is converted into relation schemas, what are the **candidate keys** of the relation schemas?
- If the relation schema…
  - … has been derived from some **entity type** $\mathbf{E}$ with key attributes $K_E := \{A_1,\ldots, A_n\}$, then $K_E \in$ cand(R)
  - … has been derived from an **N:M relationship type** between E and F, then $K_E \cup K_F \in$ cand(R)

5.4 Inherent Constraints

- If the relation schema…
  - … has been derived from an **1:1 relationship type** between E and F, then cand(E) ∪ cand(F) = cand(R)
  - … has been derived from an **N:1 relationship type** between $E_1,\ldots,E_n$ and F, then $K_{E_1} \cup \ldots \cup K_{E_n} \in$ cand(R)
    - In this case, it might also be a good to add F's key attributes

5.4 Inherent Constraints

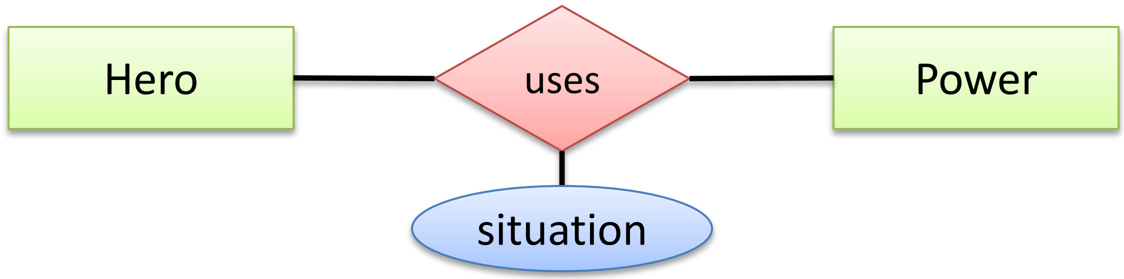
- Another constraint from ER diagrams is whether a value has to be provided for some attribute
  - NULL values, allowed by default
  - Again, this is a semantic property
- A second inherent constraint for each relation is that primary keys must **never** be NULL
  - So-called **entity integrity constraint**
  - Example: address(<u>street</u> NOT NULL,
    <u>number</u> NOT NULL,
    zip-code,
    <u>city</u> NOT NULL)

5.4 Inherent Constraints

- A third integrity constraint applies to all relation schemas that have been derived from **relationship types**
  - Relationship types borrow their (primary) keys from the entities involved, so-called **foreign keys**
  - Relationships only exist, if the respective entities exist
  - So-called **referential integrity constraint**

5.4 Inherent Constraints

- Example:



Uses(<u>firstname</u> :Varchar(20), ← Foreign key from Hero
<u>lastname</u> : Varchar(30),
<u>name</u> : Varchar(20), ← Foreign key from Power
situation : Varchar(30))

– If a tuple ('Clark', 'Kent', 'X-Ray vision', 'Bomb threat') exists in **Uses**, there has to be a tuple ('Clark', 'Kent',…) in Hero and a tuple ('X-Ray vision',…) in Power

5.4 Inherent Constraints

- All three **structural constraints** have to be checked by the database
  - Unique key constraint
  - Entity integrity constraint
  - Referential integrity constraint
  - This is especially necessary when inserting, deleting, or updating tuples in relations

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	firstName	lastName	telephoneNo
	Clark Joseph	Kent	5555678
	Louise	Lane	{3914533, 3556576, 5463456}
	Lex	Luthor	4543689
	Charles	Xavier	7658736
	Erik	Magnus	{1252345, 8766781}

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?

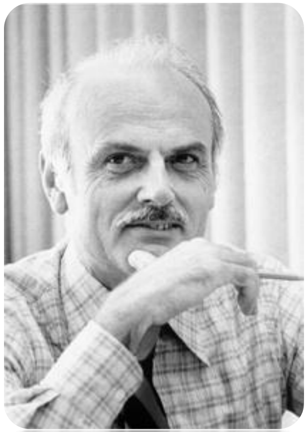
Person	firstName	lastName	telephoneNo
	Clark Joseph	Kent	555-5678
	Louise	Lane	391-4533
	Louise	Lane	355-6576
	Louise	Lane	546-3456
	Lex	Luthor	454-3689
	Charles	Xavier	765-8736
	Erik	Magnus	125-2345
	Erik	Magnus	876-6781

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, …

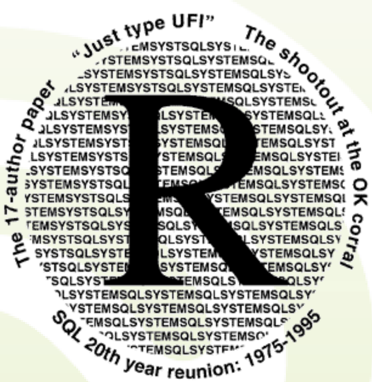
5.5 From Theory to Practice

Detour



- In the early 1970s, the **relational model** became a "hot topic" 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?**

- **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?



5.5 From Theory to Practice

Detour

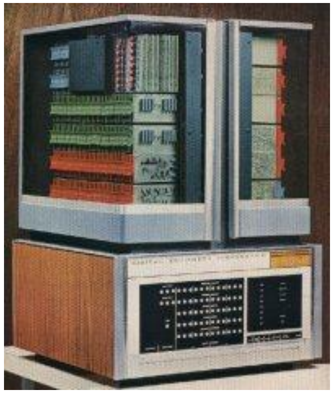
- The challenge of the **System R** project was to create a **working prototype system**
  - Theory is good
  - But developers were willing to sacrifice theoretical beauty and clarity for the sake of usability and performance
- **Vocabulary change**
  - Mathematical terms were too unfamiliar for most people
  - **Table** = relation
  - **Row** = tuple
  - **Column** = attribute
  - **Data type, domain** = domain

- **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…

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**

Detour



- **Practical considerations**
  - You want to query for **name** and **birth year** of all **students** of TU Braunschweig
  - The result returns roughly **13,000 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

5.5 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

5.5 From Theory to Practice

Detour

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

- 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"

5.5 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 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

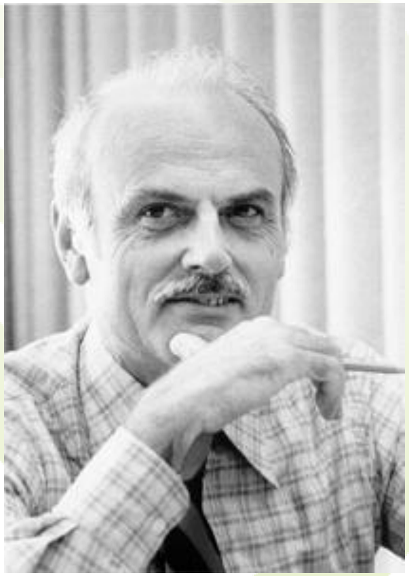
5.5 From Theory to Practice

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

Preview – Relational Algebra

- How do you work with relations?
- **Relational algebra!**
  - Proposed by Edgar F. Codd: "A Relational Model for Large Shared Data Banks," Communications of the ACM, 1970
- The theoretical foundation of all relational databases
  - Describes how to manipulate relations and retrieve interesting parts of available relations
  - Relational algebra is mandatory for advanced tasks like query optimization

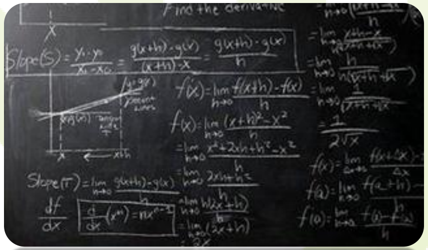


Preview – Relational Algebra

- Elementary operations:
  - **Set algebra operations**
    - Set Union ∪
    - Set Intersection ∩
    - Set Difference \
    - Cartesian Product ×
  - New **relational algebra operations**
    - Selection σ
    - Projection π
    - Renaming ρ
- Additional derived operations (for convenience)
  - All sorts of joins ⋈,⋉,⋊, …
  - Division ÷
  - …

Preview – Relational Calculi

- Beside the **relational algebra,** there are two other major **query** paradigms within the relational model
  - **Tuple relational calculus** (TRC)
  - **Domain relational calculus** (DRC)
- All three provide the **theoretical** foundation of the relational database model
- They are mandatory for certain DB features:
  - Relational algebra → Query **optimization**
  - TRC → **SQL** query language
  - DRC → **Query-by-example** paradigm



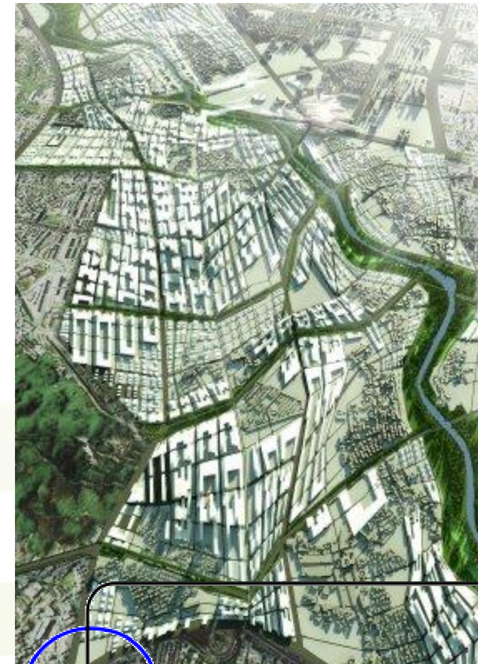
Preview – Relational Calculi

- Relational algebra has some **procedural** aspects
  - You specify an **order of operations** describing how to retrieve data
- Relational calculi (TRC, DRC) are **declarative**
  - You just **specify** how the desired **tuples look like**
  - The query contains no information about how to create the result set
  - Provides an alternative approach to querying

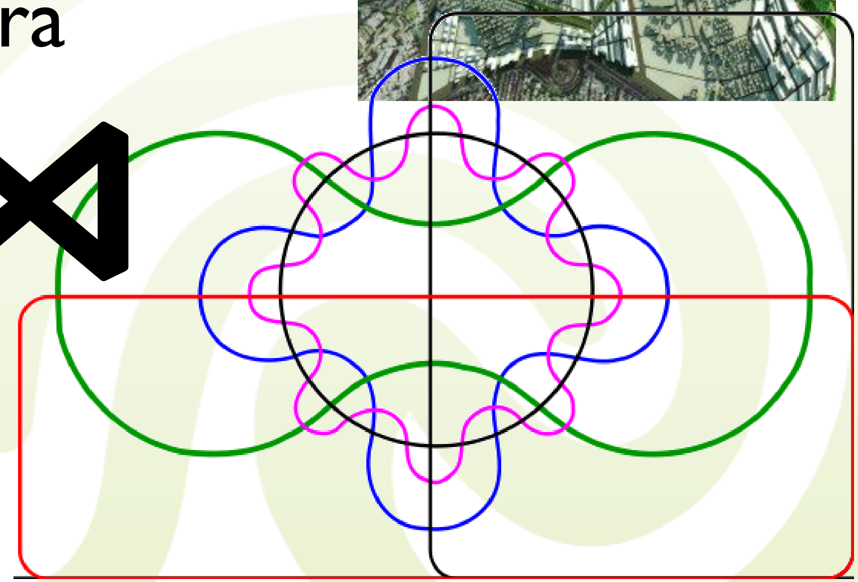
Next Lecture

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

σ



$\bowtie$



π