Relational Database Systems I

Christoph Lofi
Simon Barthel
Institut für Informationssysteme
Technische Universität Braunschweig
www.ifis.cs.tu-bs.de
Common Mistakes in Modelling

• 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.”
• **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, isbn, summary, price)`
    • **Less good:** `Book(name, author, year, 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, summary, price)`
      - `Book(name, author, year, isbn, summary, price)`
  - **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
• **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 relationships (n>2)
– **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
• **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

– Better:

– Or maybe:
5 Relational Model

• Basic Set Theory
• Relational Model
• From Theory to Practice
• Integrity Constraints
• Conversion from ER
• **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\}\n
• 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 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?
• 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 $\approx$ database query
  – Extensional set $\approx$ 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’ \( \neq \) ‘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 $A \subseteq B$
• 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 $(a, b, c)$
• 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: $\cup$
  - 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: $\cap$
  - creates a new set containing all elements that are contained in both sets
  - $\{a, b\} \cap \{b, c\} = \{b\}$
5.1 Set Operations

- **Difference:** \( \setminus \)
  - 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
  – \( \{a, b\} \times \{b, c\} = \{\langle a, b \rangle, \langle a, c \rangle, \langle b, b \rangle, \langle b, c \rangle\} \)
  – Cleverness = \{ genius, dumb \}
  – Character = \{ hero, villain \}
  – Cleverness \times Character = \{\langle genius, hero \rangle, \langle dumb, hero \rangle, \langle genius, villain \rangle, \langle dumb, villain \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
    - $\ldots$
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
  – \( R = \{ (a,1), (c,1), (d,4), (e,5), (e,6) \} \)
    • can also be written as \( aR1, cR1, dR4, \ldots \)
  – imagine **Likes \subseteq Person \times 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 realtions
5.1 Relations

• Example
  – Accessory = \{spikes, butterfly helmet\}
  – Material = \{silk, armor plates\}
  – Color = \{pink, black\}

\[
\text{Color} \times \text{Material} \times \text{Accessory} = \\
\{\langle \text{pink}, \text{silk}, \text{butterfly helmet} \rangle, \\
\langle \text{pink}, \text{silk}, \text{spikes} \rangle, \\
\langle \text{pink}, \text{armor plates}, \text{butterfly helmet} \rangle, \\
\langle \text{pink}, \text{armor plates}, \text{spikes} \rangle, \\
\langle \text{black}, \text{silk}, \text{butterfly helmet} \rangle, \\
\langle \text{black}, \text{silk}, \text{spikes} \rangle, \\
\langle \text{black}, \text{armor plates}, \text{butterfly helmet} \rangle, \\
\langle \text{black}, \text{armor plates}, \text{spikes} \rangle\}\]

5.1 Relations

- Relation `FamousHeroCostumes` contains combinations of `Color`, `Material`, and `Accessory`.

\[ \text{FamousHeroCostumes} = \{ \langle \text{pink, silk, butterfly helmet} \rangle, \langle \text{black, armor plates, spikes} \rangle \} \]
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\}
  – to be independent of the domain order, the tuple \langle pink, silk, butterfly helmet \rangle can also be represented as the following function \( t \)
    • \( t(\text{Color}) = \text{pink} \)
    • \( t(\text{Material}) = \text{silk} \)
    • \( t(\text{Accessory}) = \text{butterfly helmet} \)
  – Usually, one writes \( t[\text{color}] \) instead of \( t(\text{color}) \)
  – This can be used to change the order of domains for tuples
    • \( t[\text{Material, Accessory, Color}] = \langle \text{silk, butterfly helmet, pink} \rangle \)
• Basic Set Theory
• **Relational Model**
• From Theory to Practice
• Integrity Constraints
• Conversion from ER
• 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 $\text{dom}$ function
  - $\text{dom}(A_1) = D_1$, $\text{dom}(A_2) = D_2$, ...
  - 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**

<table>
<thead>
<tr>
<th>PERSON</th>
<th>first_name</th>
<th>last_name</th>
<th>sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clark Joseph</td>
<td>Kent</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Louise</td>
<td>Lane</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>Lex</td>
<td>Luthor</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Charles</td>
<td>Xavier</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Erik</td>
<td>Magnus</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Jeanne</td>
<td>Gray</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>Ororo</td>
<td>Munroe</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>Tony Edward</td>
<td>Stark</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Matt</td>
<td>Murdock</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>Raven</td>
<td>Wagner</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>Robert Bruce</td>
<td>Banner</td>
<td>m</td>
<td></td>
</tr>
</tbody>
</table>
• 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
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
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?
• **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?
• 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

• **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

• 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
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
**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
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?
• 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
• 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
• 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. The popularity rating of any sidekick should always be less than the respective super hero’s.

• 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
    \[ R \subseteq D_1 \times \ldots \times D_n \] 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
    
    • e.g., \( \text{Hero}(id, \text{first name}, \text{last name}) \)
      \( \text{Aliases}(\text{alias}, \text{heroid} \rightarrow \text{Hero}) \)
  
  – 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:

Hero(id, first name, last name)
Aliases(alias, heroid→Hero)

<table>
<thead>
<tr>
<th>Person</th>
<th>id</th>
<th>firstname</th>
<th>lastname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Charles</td>
<td>Xaviar</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>James</td>
<td>Howlett</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Jean</td>
<td>Grey</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Max</td>
<td>Eisenhardt</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
<th>alias</th>
<th>heroid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>White Queen</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Logan</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Wolverine</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Weapon X</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Eric Magnus</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Magneto</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>Professor X</td>
<td>1</td>
</tr>
</tbody>
</table>

Invalid!
• Convention:
  – If a composite key is referenced, we write this as, e.g.,
    \( R_1(a, b, c) \), \( R_2(d, e, f, (d, f) \rightarrow R_1) \)
  – This is not a standard notation, but rather close to what you find in SQL
There is 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 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, ...).

<table>
<thead>
<tr>
<th>Person</th>
<th>first_name</th>
<th>last_name</th>
<th>telephone_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clark Joseph</td>
<td>Kent</td>
<td></td>
<td>5555678</td>
</tr>
<tr>
<td>Louise</td>
<td>Lane</td>
<td>{3914533, 3556576, 5463456}</td>
<td></td>
</tr>
<tr>
<td>Lex</td>
<td>Luthor</td>
<td></td>
<td>4543689</td>
</tr>
<tr>
<td>Charles</td>
<td>Xavier</td>
<td></td>
<td>7658736</td>
</tr>
<tr>
<td>Erik</td>
<td>Magnus</td>
<td>{1252345, 8766781}</td>
<td></td>
</tr>
</tbody>
</table>

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?

<table>
<thead>
<tr>
<th>Person</th>
<th>first_name</th>
<th>last_name</th>
<th>telephone_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clark</td>
<td>Joseph</td>
<td>Kent</td>
<td>555-5678</td>
</tr>
<tr>
<td>Louise</td>
<td></td>
<td>Lane</td>
<td>391-4533</td>
</tr>
<tr>
<td>Louise</td>
<td></td>
<td>Lane</td>
<td>355-6576</td>
</tr>
<tr>
<td>Louise</td>
<td></td>
<td>Lane</td>
<td>546-3456</td>
</tr>
<tr>
<td>Lex</td>
<td></td>
<td>Luthor</td>
<td>454-3689</td>
</tr>
<tr>
<td>Charles</td>
<td></td>
<td>Xavier</td>
<td>765-8736</td>
</tr>
<tr>
<td>Erik</td>
<td></td>
<td>Magnus</td>
<td>125-2345</td>
</tr>
<tr>
<td>Erik</td>
<td></td>
<td>Magnus</td>
<td>876-6781</td>
</tr>
</tbody>
</table>
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.
• The ER diagram is **semantically richer** than the relational model
  – However, it's not a real subset
• Many constraints are very hard/impossible to express
  – disjoint/overlapping generalization
  – non-trivial cardinalities (1, *)
  – ...
• Therefore, it usually is a really good idea to create an ER diagram before coding a logical schema
• 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:

\[
\text{Hero}(id, \text{name}, \text{alias})
\]
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

\[
\begin{align*}
\text{Hero} & (id, \text{name}, \text{alias}) \\
\text{Villain} & (id, \text{name}, \text{alias}) \\
\text{hero_fights_villain} & (\text{hero} \rightarrow \text{Hero}, \text{villain} \rightarrow \text{Villain})
\end{align*}
\]
Converting an 1:m relationship type to a relation schema:

- Entity Type at 1-side can only participate once at the relationship type
  
  => Push relationship type to the 1-side
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

To check if it really 1:1 we need advanced constraints
  – e.g. by using triggers (to be introduced in lecture 14)
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)

Scientist\((id, name, \#lab\_coats)\)

Invention\((id, code\_name)\)

```
scientist_works_on_invention(
    scientist \rightarrow Scientist,
    invention \rightarrow Invention,
    hours
)
```
What about n-ary relationship types? (n>2)

– Just apply the exact same approaches:

```sql
sc_wrk_on_inv_in_lab(
    scientist → Scientist,
    invention → Invention,
    lab → Lab
    hours)
```
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

Evil_Plan(code_name, chance_of_success)

todo_item(priority_order, evil_plan → Evil_Plan, done)
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

Secret_Hideout(id, addr_city, addr_street, addr_number)
hideout_name (hideout → Secret_Hideout, name)
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

Gadget\((\text{id}, \text{name})\)

Weapon\((\text{gadget} \to \text{Gadget}, \text{range})\)

Util_gadget\((\text{gadget} \to \text{Gadget}, \text{weight})\)
5.5 Conversion from ER

- Example:

![ER Diagram]

- Person
  - firstname
  - lastname
  - telephone no

- Hero
  - weakness
  - alias
  - (1, *)

- Power
  - reach
  - name
  - description
  - solution
  - has
    - (1, 1)

- Side Effect
  - (0, *)

- uses
  - (0, *)
5.5 Conversion from ER

- Entity types:

\[
\text{Person}(\text{firstname}: \text{string}, \text{lastname}: \text{string}, \text{telephone_no}: \text{string})
\]

\[
\text{Hero}(\text{firstname}: \text{string}, \text{lastname}: \text{string}, \text{alias}: \text{string}, \text{weakness}: \text{string}, (\text{firstname}, \text{lastname}) \rightarrow \text{Person})
\]
### 5.5 Conversion from ER

**Relationship types: N:M**

<table>
<thead>
<tr>
<th>Use</th>
<th>Name</th>
<th>Type</th>
<th>Reach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td><strong>name</strong>: string, <strong>type</strong>: string, <strong>reach</strong>: numeric</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Uses**
- **firstname**: string
- **lastname**: string
- **name** → Power
- **situation**: string
- **(firstname, lastname)** → Hero

#### Diagram:
- **Hero**
  - **weakness**
  - **alias**
- **Power**
  - **type**
  - **reach**
  - **name**

**Uses**:
- **Power**
  - **(1, *)**
- **uses**
  - **(0, *)**
- **(firstname, lastname) → Hero**
5.5 Conversion from ER

- Relationship types: 1:N

**Power**
- **name**: string,
- **type**: string,
- **reach**: numeric

**SideEffect**
- **description**: string,
- **power**: string → Power,
- **solution**: string
6.0 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
6.0 Preview – Relational Algebra

• Elementary operations:
  – **set algebra operations**
    • Set Union \( \cup \)
    • Set Intersection \( \cap \)
    • Set Difference \( \setminus \)
    • Cartesian Product \( \times \)
  – **new relational algebra operations**
    • Selection \( \sigma \)
    • Projection \( \pi \)
    • Renaming \( \rho \)

• Additional derived operations (for convenience)
  – all sorts of joins \( \bowtie, \bowtie, \bowtie, \ldots \)
  – division \( \div \)
  – \( \ldots \)
6.0 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
• 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
• Relational Algebra
  – Basic relational algebra operations
  – Additional derived operations
• Query Optimization
• Advanced relational algebra
  – Outer Joins
  – Aggregation