ifis

Institut für Informationssysteme
Technische Universität Braunschweig

Relational Database Systems I

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

Overview

- Normalization
  - Functional dependencies
  - 2NF
  - 3NF
  - BCNF
  - 4NF, 5NF, 6NF
- Denormalization

10.1 Introduction

- Up to now, we have learned ...
  - ... how the relational model works.
  - ... how it is implemented in current RDBMS.
  - ... how to create relational databases (SQL DDL).
  - ... how to define constraints (SQL DDL).
  - ... how to query relational databases.
  - ... how to insert, delete, and update data (SQL DML).
- What's missing?
  - How to create a "good" database design?
  - By the way: What is a **"good" database design?**

10.1 Introduction

Quiz

- Which table design is better?

A

heroID	teamID	heroName	teamName	joinYear
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Hulk	The Avengers	1963
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

B

heroID	heroName
1	Thor
2	Mister Fantastic
3	Iron Man
4	Hulk
5	Captain America
6	Invisible Girl

teamID	teamName
1	The Avengers
2	Fantastic Four

heroID	teamID	joinYear
1	1	1963
2	2	1961
3	1	1963
4	1	1963
5	1	1964
6	2	1961

10.1 Introduction

A

heroID	teamID	heroName	teamName	joinYear
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Hulk	The Avengers	1963
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

- What's wrong with design A?
  - **Redundancy:** The fact that certain teams have certain names is represented several times.
  - **Inferior expressiveness:** We cannot represent heroes that currently have no team.
  - **Modification anomalies:** (see next slide)

10.1 Introduction

A

heroID	teamID	heroName	teamName	joinYear
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Hulk	The Avengers	1963
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

- There are three kinds of modification anomalies:
  - **Insertion anomalies**
    - How do you add heroes that currently have no team?
    - How do you (consistently!) add new tuples?
  - **Deletion anomalies**
    - Deleting Mister Fantastic and Invisible Girl also deletes all information about the Fantastic Four
  - **Update anomalies**
    - Renaming a team requires updating several tuples (due to redundancy)

10.1 Introduction

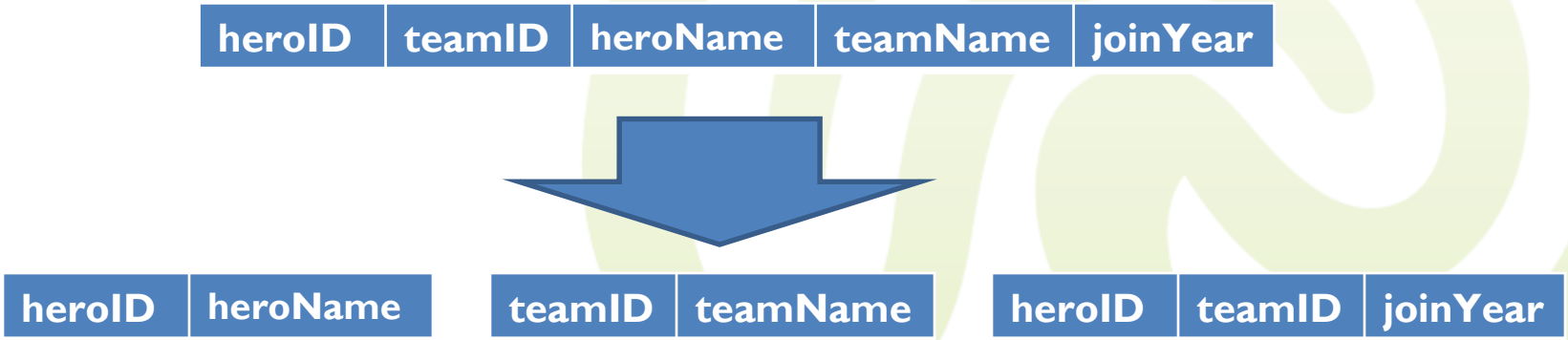
- In general, **"good" relational database designs** have the following properties:
  - Redundancy is minimized
    - That is: no information is represented several times!
    - Logically distinct information is placed in distinct relation schemes
  - Modification anomalies are prevented "by design"
    - That is: by using keys and foreign keys, not by enforcing an excessive amount of (hard to check) constraints!
  - In practice, "good" designs should also match the characteristics of the used RDBMS
    - Enable efficient query processing
- In essence, it's all about splitting up tables ...
  - Remember design B

10.2 Normalization

- These "rules of thumb" can be formalized by the concept of relational database **normalization**
- But before going into details, let's recap some definitions from the relational model:
  - Data is represented using a **relation schema** $S(R_1, ..., R_n)$
    - Each relation $R(A_1, \ldots A_n)$ contains attributes $A_1, \ldots A_n$
  - A **relational database schema** consists of
    - A set of relations
    - A set of **integrity constraints** (e.g. "heroID is unique" and "heroID determines heroName")
  - A **relational database instance** (or extension) is
    - A set of tuples adhering to the respective schemas and respecting all integrity constraints

10.2 Normalization

- For this lecture, let's assume the following:
  - $S(R_1, ..., R_n)$ is a **relation schema**
  - $R(A_1, ..., A_n)$ is a **relation** in S
  - $\mathcal{C}$ is a set of **constraints** satisfied by all extensions of S
- Our ultimate goal is to enhance the database design by **decomposing** the relations in S into a set of smaller relations, as we did in our example:



10.2 Normalization

- **Definition (decomposition):**
  - Let $\alpha_1, ..., \alpha_k \subseteq \{A_1, ..., A_n\}$ be k subsets of R's attributes
    - Note that these subsets may be overlapping
  - Then, for any α_i, a new relation R_i can be derived:

$$R_i = \pi_{\alpha_i}(R)$$

  - $\alpha_1, ..., \alpha_k$ is called a **decomposition** of R

- "Good" decompositions have to be **reversible:**
  - The decomposition $\alpha_1, ..., \alpha_k$ is called **lossless** if and only if $R = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k$, for any extension of R satisfying the constraints $\mathcal{C}$

10.2 Normalization

- Example:

$\mathcal{C}$ = {"{heroID, teamID} is unique",
"heroID determines heroName",
"teamID determines teamName",
"{heroID, teamID} determines joinYear"}

Hero

heroID	teamID	heroName	teamName	joinYear
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Hulk	The Avengers	1963
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

– Our example decomposition is lossless:

α_1 = {heroID, heroname}, α_2 = {teamID, teamName}, α_3 = {heroID, teamID, joinYear}

π_{α_1}(Hero)

heroID	heroName
1	Thor
2	Mister Fantastic
3	Iron Man
4	Hulk
5	Captain America
6	Invisible Girl

π_{α_2}(Hero)

teamID	teamName
1	The Avengers
2	Fantastic Four

π_{α_3}(Hero)

heroID	teamID	joinYear
1	1	1963
2	2	1961
3	1	1963
4	1	1963
5	1	1964
6	2	1961

10.2 Normalization

Quiz

$\mathcal{C}$ = {"{heroID, teamID} is unique",
"heroID determines heroName",
"teamID determines teamName",
"{heroID, teamID} determines joinYear"}

Hero

heroID	teamID	heroName	teamName	joinYear
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Hulk	The Avengers	1963
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

– Is the following a lossless decomposition?

α_1 = {heroID, teamID, joinYear}, α_2 = {teamID, heroName, teamName, joinYear}

π_{α_1}(Hero)

heroID	teamID	joinYear
1	1	1963
2	2	1961
3	1	1963
4	1	1963
5	1	1964
6	2	1961

π_{α_2}(Hero)

teamID	heroName	teamName	joinYear
1	Thor	The Avengers	1963
2	Mister Fantastic	Fantastic Four	1961
1	Iron Man	The Avengers	1963
1	Hulk	The Avengers	1963
1	Captain America	The Avengers	1964
2	Invisible Girl	Fantastic Four	1961

10.2 Normalization

- **Normalizing** a relation schema S means replacing relations in S by lossless decompositions
- However, this raises some new questions:
  - Under which conditions there is a (nontrivial) lossless decomposition?
    - Decompositions involving $\alpha_i = \{A_1, ..., A_n\}$ or $\alpha_i = \emptyset$ are called **trivial**
  - If there is a lossless decomposition, how to find it?
  - How to measure a relation schema's "design quality"?
    - We may abstain from further normalization if the quality is "good enough" ...

10.2 Normalization

- The normalization of S depends entirely on the set of **constraints** $\mathcal{C}$ imposed on S
- Instead of dealing with constraints of arbitrary complexity, we restrict $\mathcal{C}$ to the class of **functional dependencies (FDs)**
  - "heroName is completely determined by heroID" is an example for a functional dependency
  - Most update anomalies and problems of redundancy occurring in practice can be traced back to violations of functional dependency constraints
    - Typically, functional dependencies are all you need

10.3 Functional Dependencies

- Informally, functional dependencies can be described as follows:
  - Let *X* and *Y* be some sets of attributes
  - "If *Y* **functionally depends** on *X*, and two tuples agree on their *X* values, then they also **have to** agree on their *Y* values"
- Examples:
  - "{end time} functionally depends on {start time, duration}"
  - "{duration} functionally depends on {start time, end time}"
  - "{end time} functionally depends on {end time}"

10.3 Functional Dependencies

Formal definition:

- Let X and Y be subsets of R's attributes
  - That is, $X, Y \subseteq \{A_1, ..., A_n\}$
- There is **functional dependency (FD)** between X and Y (denoted as $X \rightarrow Y$), if and only if, ...
  - … for any two tuples t_1 and t_2 within any instance of R, the following is true:

$$\text{If } \pi_X t_1 = \pi_X t_2, \text{ then } \pi_Y t_1 = \pi_Y t_2$$

10.3 Functional Dependencies

- If $X \rightarrow Y$, then one says that ...
  - X **functionally determines** Y, and
  - Y **functionally depends** on X.
- X is called the **determinant** of the FD $X \rightarrow Y$
- Y is called the **dependent** of the FD $X \rightarrow Y$

10.3 Functional Dependencies

- Functional dependencies are semantic properties of the underlying domain and data model
- FDs are NOT a property of a particular instance (extension) of the relation schema!
  - The **designer** is responsible for **identifying** FDs
  - FDs are **manually defined** integrity constraints on *S*
  - All extensions respecting *S*'s functional dependencies are called **legal extensions** of *S*

10.3 Functional Dependencies

- In fact, functional dependencies are a generalization of **key constraints**
- To see this, we need a short recap:
  - A set of attributes *X* is a **(candidate) key** for *R* if and only if it has both of the following properties:
    - **Uniqueness:** No legal instance of *R* ever contains two distinct tuples with the same value for *X*
    - **Irreducibility:** No proper subset of *X* has the uniqueness property
  - A **superkey** is a superset of a key
    - That is, only uniqueness is required

10.3 Functional Dependencies

- In practice, if there is more than one key, we usually choose one and call it the **primary key**
  - However, for normalization purposes, only keys are important. Thus, **we ignore primary keys** today.
- The following is true:
  - X is a superkey of R

    if and only if

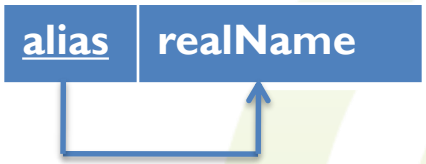
    $X \rightarrow \{A_1, ..., A_n\}$ is a functional dependency in R

10.3 Functional Dependencies

- Example:
  - A relation containing students
    - Semantics: matriculationNo is **unique**
    - {matriculationNo} → {firstName, lastName, dateOfBirth}

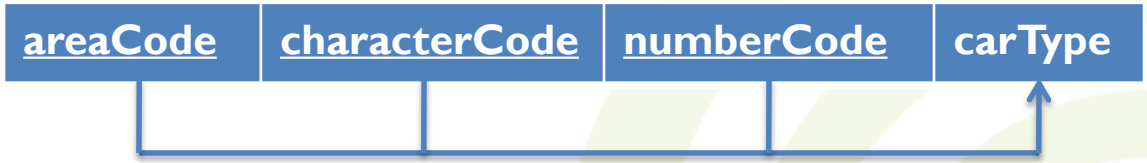
10.3 Functional Dependencies

- Example:
  - A relation containing real names and aliases of heroes, where each hero has only one unique alias
    - {alias} → {realName}



10.3 Functional Dependencies

- Example:
  - A relation containing license plates and the type of the respective car
    - {areaCode, characterCode, numberCode} → {carType}



10.3 Functional Dependencies

Quiz

- What FDs can be derived from the following description of an **address book?**

name	street	city	state	zip

  - For any given zip code, there is just one city and state.
  - For any given street, city, and state, there is just one zip code.
- FDs and candidate keys?

10.3 Functional Dependencies

- One possible solution:
  - {zip} → {city, state}
  - {street, city, state} → {zip}
- Typically, not all actual FDs are modeled explicitly:
  - {zip} → {city}
  - {street} → {street}
  - {state} → ∅
  - ...

name	street	city	state	zip

10.3 Functional Dependencies

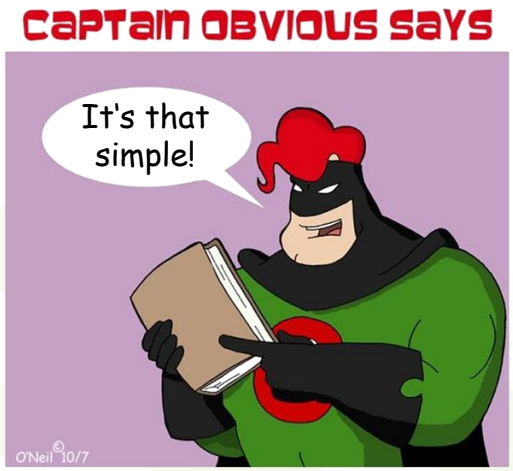
- Obviously, some FDs are **implied** by others
  - {zip} → {city, state} implies {zip} → {city}
- Moreover, some FDs are **trivial**
  - {street} → {street}
  - {state} → ∅
  - **Definition:** The FD $X \rightarrow Y$ is called trivial iff $X \supseteq Y$
- What do we need?
  - A **compact representation** for sets of FD constraints
    - No redundant FDs
  - An **algorithm** to compute the set of all implied FDs

10.3 Functional Dependencies

- **Definition:**
  For any set *F* of FDs, the **closure** of *F* (denoted F^+) is the set of all FDs that are logically **implied** by *F*
  - *F* **implies** the FD $X \rightarrow Y$, if and only if any extension of *R* satisfying any FD in *F*, also satisfies the FD $X \rightarrow Y$
- Fortunately, the closure of *F* can easily be computed using a small set of **inference rules**

10.3 Functional Dependencies

- For any attribute sets X, Y, Z, the following is true:
  - **Reflexivity:**
    If $X \supseteq Y$, then $X \rightarrow Y$
  - **Augmentation:**
    If $X \rightarrow Y$, then $X \cup Z \rightarrow Y \cup Z$
  - **Transitivity:**
    If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$



- These rules are called **Armstrong's axioms**
  - One can show that they are **complete** and **sound**
    - **Completeness:** Every implied FD can be derived
    - **Soundness:** No non-implied FD can be derived

10.3 Functional Dependencies

- To **simplify the practical task** of computing F^+ from F, several **additional rules** can be derived from Armstrong's axioms:
  - **Decomposition:**
    If $X \rightarrow Y \cup Z$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - **Union:**
    If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow Y \cup Z$
  - **Composition:**
    If $X \rightarrow Y$ and $Z \rightarrow W$, then $X \cup Z \rightarrow Y \cup W$

10.3 Functional Dependencies

- Example:
  - Relational schema *R*(A, B, C, D, E, F)
  - FDs: {A} → {B, C} {B} → {E} {C, D} → {E, F}
  - Then we can make the following derivation:
    1. **{A} → {B, C}** (given)
    2. {A} → {C} (by decomposition)
    3. {A, D} → {C, D} (by augmentation)
    4. {A, D} → {E, F} (by transitivity with given {C, D} → {E, F})
    5. **{A, D} → {F}** (by decomposition)

10.3 Functional Dependencies

- In principle, we can compute the closure F^+ of a given set F of FDs by means of the following algorithm:
  - Repeatedly apply the six inference rules until they stop producing new FDs.
- In practice, this algorithm is hardly very efficient
  - However, there usually is little need to compute the closure per se
  - Instead, it often suffices to compute a certain subset of the closure: namely, that subset consisting of all FDs with a certain left side

10.3 Functional Dependencies

- **Definition:**
  Given a set of attributes X and a set of FDs F,
  the **closure** of X under F, written as $(X, F)^+$,
  consists of all attributes that functionally depend on X
  - That is, $(X, F)^+ := \{A_i \mid X \rightarrow A_i \text{ is implied by } F\}$
- The following algorithm computes $(X, F)^+$:

```
unused := F
closure := X
do {
  for(Y → Z ∈ unused) { if(Y ⊆ closure) {
      unused := unused \ {Y → Z}
      closure := closure ∪ Z
  }}
}while(unused and closure did not change)
return closure
```

10.3 Functional Dependencies

- **Quiz:**
  - $F = \{ \quad \{A\} \rightarrow \{B, C\}, \quad \{E\} \rightarrow \{C, F\}, \quad \{B\} \rightarrow \{E\}, \quad \{C, D\} \rightarrow \{E, F\} \quad \}$
  - **What is the closure of {A, B} under *F*?**

10.3 Functional Dependencies

- Now, we can do the following:
  - Given a set F of FDs, we can easily tell whether a specific FD $X \rightarrow Y$ is **contained in F^+**
    - Just check whether $Y \subseteq (X, F)^+$
  - In particular, we can find out whether a set of attributes X is a **superkey** of R
    - Just check whether $(X, F)^+ = \{A_1, ..., A_n\}$
- What's still missing?
  - Given a set of FDs F, how to find a set of FDs G, such that **$F^+ = G^+$**, and G is **as small as possible?**
  - Given sets of FDs F and G, does **$F^+ = G^+$** hold?

10.3 Functional Dependencies

- **Definition:**
  Two sets of FDs F and G are **equivalent** iff $F^+ = G^+$

- How can we find out whether two given sets of FDs F and G are equivalent?
  - **Theorem:**
    $F^+ = G^+$ iff for any FD $X \rightarrow Y \in F \cup G$, it is $(X, F)^+ = (X, G)^+$
  - Proof:
    - Let $F' = \{X \rightarrow (X, F)^+ \mid X \rightarrow Y \in F \cup G\}$
    - Analogously, derive G' from G
    - Obviously, then $F'^+ = F^+$ and $G'^+ = G^+$
    - Moreover, every left side of an FD in F' occurs as a left side of an FD in G' (and reverse)
    - If F' and G' are different, then also F^+ and G^+ must be different

10.3 Functional Dependencies

- **Example:**
  - $F = \{ \quad \{A, B\} \rightarrow \{C\}, \quad \{C\} \rightarrow \{B\} \quad \}$
  - $G = \{ \quad \{A\} \rightarrow \{C\}, \quad \{A, C\} \rightarrow \{B\} \quad \}$
  - Are *F* and *G* equivalent?

  - We must check $(X, F)^+ = (X, G)^+$ for the following *X*:
    - {A, B}, {C}, {A}, and {A, C}
  - $(\{A, B\}, F)^+ = \{A, B, C\}$ $\quad$ $(\{A, B\}, G)^+ = \{A, B, C\}$
  - $(\{C\}, F)^+ = \{B, C\}$ $\quad$ $(\{C\}, G)^+ = \{C\}$

  - Therefore, *F* and *G* are not equivalent!

10.3 Functional Dependencies

- **Remember:**
  To have a **small representation** of *F*, we want to find a *G*, such that:
  - *F* and *G* are equivalent
  - *G* is "as small as possible" (we will call this property **minimality**)

- **Definition:**
  A set of FDs *F* is **minimal** iff the following is true:
  - Every FD $X \rightarrow Y$ in *F* is **in canonical form**
    - That is, *Y* consists of exactly one attribute
  - Every FD $X \rightarrow Y$ in *F* is **left-irreducible**
    - That is, no attribute can be removed from *X* without changing F^+
  - Every FD $X \rightarrow Y$ in *F* is **non-redundant**
    - That is, $X \rightarrow Y$ cannot be removed from *F* without changing F^+

10.3 Functional Dependencies

- The following algorithm "minimizes" *F*, that is, it transforms *F* into a minimal equivalent of *F*:
  1. Split up all right sides to get FDs in canonical form
  2. Remove all redundant attributes from the left sides (by checking which attribute removals change F^+)
  3. Remove all redundant FDs from *F* (by checking which FD removals change F^+)

10.3 Functional Dependencies

- **Example:**
  - Given F = { $\{A\} \rightarrow \{B, C\}$, $\{B\} \rightarrow \{C\}$, $\{A\} \rightarrow \{B\}$, $\{A, B\} \rightarrow \{C\}$, $\{A, C\} \rightarrow \{D\}$ }

1. Split up the right sides:
   $\{A\} \rightarrow \{B\}$, $\{A\} \rightarrow \{C\}$, $\{B\} \rightarrow \{C\}$, $\{A, B\} \rightarrow \{C\}$, $\{A, C\} \rightarrow \{D\}$
2. Remove C from $\{A, C\} \rightarrow \{D\}$
   - $\{A\} \rightarrow \{C\}$ implies $\{A\} \rightarrow \{A, C\}$ (augmentation)
   - $\{A\} \rightarrow \{A, C\}$ and $\{A, C\} \rightarrow \{D\}$ imply $\{A\} \rightarrow \{D\}$ (transitivity)

10.3 Functional Dependencies

– Now we have:
  $\{A\} \rightarrow \{B\}$, $\{A\} \rightarrow \{C\}$, $\{B\} \rightarrow \{C\}$,
  $\{A, B\} \rightarrow \{C\}$, $\{A\} \rightarrow \{D\}$

3. Remove $\{A, B\} \rightarrow \{C\}$
   - $\{A\} \rightarrow \{C\}$ implies $\{A, B\} \rightarrow \{C\}$
4. Remove $\{A\} \rightarrow \{C\}$
   - $\{A\} \rightarrow \{B\}$ and $\{B\} \rightarrow \{C\}$ imply $\{A\} \rightarrow \{C\}$ (transitivity)

– Finally, we end up with a **minimal equivalent** of *F*:
  $\{A\} \rightarrow \{B\}$, $\{B\} \rightarrow \{C\}$, $\{A\} \rightarrow \{D\}$

10.3 Functional Dependencies

- **Functional dependencies** are the perfect tool for performing **lossless decompositions**
  - **Heath's Theorem:**

    Let $X \rightarrow Y$ be an FD constraint of the relation schema $R(A_1, ..., A_n)$. Then, the following decomposition of R is **lossless:**

    $$\alpha_1 = X \cup Y \quad \text{and} \quad \alpha_2 = \{A_1, ..., A_n\} \setminus Y.$$

  - **Example:**

FDs:
{heroID} → {heroName}
{teamID} → {teamName}
{heroID, teamID} → {joinYear}

heroID	teamID	heroName	teamName	joinYear

Decompose with respect to
{heroID} → {heroName}

heroID	heroName

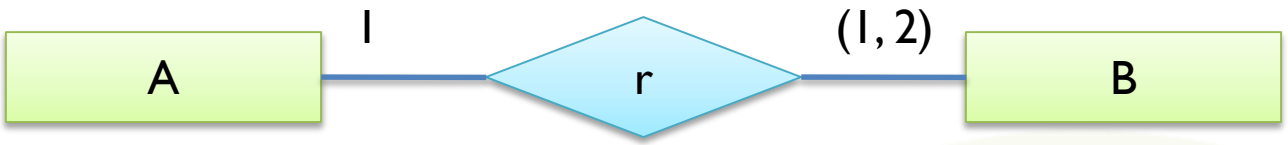
heroID	teamID	teamName	joinYear

10.3 Functional Dependencies Detour

- How to come up with functional dependencies?
  - There are several ways:
    - Based on "domain knowledge"
    - Based on an explicit data model
    - Based on existing data

1. Based on "domain knowledge"
   - "Obvious" FDs are easy to find
   - What about more complicated FDs?
   - No guarantee that you found all (important) FDs!

2. Based on an explicit model



- Automated generation of FDs possible
- But: Are all actual FDs present in the model?
  - What about FDs between attributes of the same entity?

10.3 Functional Dependencies Detour

3. Based on existing data
   - In practice, often there is already some data available (that is, tuples)
   - We can use the data to derive FD constraints
   - Obviously:
     - All FDs that hold in general for some relation schema, also hold for any given extension
     - Therefore, the set of all FDs that hold in some extension, is a superset of all "true" FDs of the relation schema
   - What we can do:
     - Find all FDs that hold in a given extension
     - Find a minimal representation of this FD set
     - Ask a domain expert, what FDs are generally true

10.3 Functional Dependencies

Quiz

A	B	C	D	E
1	1	1	1	1
1	2	2	2	1
2	1	3	3	1
2	1	4	3	1
3	2	5	1	1

- Which of the following FDs are satisfied in this particular extension?
  a) $\{C\} \rightarrow \{A, B\}$
  b) $\{A, D\} \rightarrow \{C\}$
  c) $\emptyset \rightarrow \{E\}$

10.3 Functional Dependencies

Detour

– **Find all FDs that are satisfied in this extension!**
  - We will check any FD $X \rightarrow Y$ in **canonical form,** i.e.,
    X is a **subset** of {A, B, C, D, E} and
    Y is an **element** of {A, B, C, D, E}

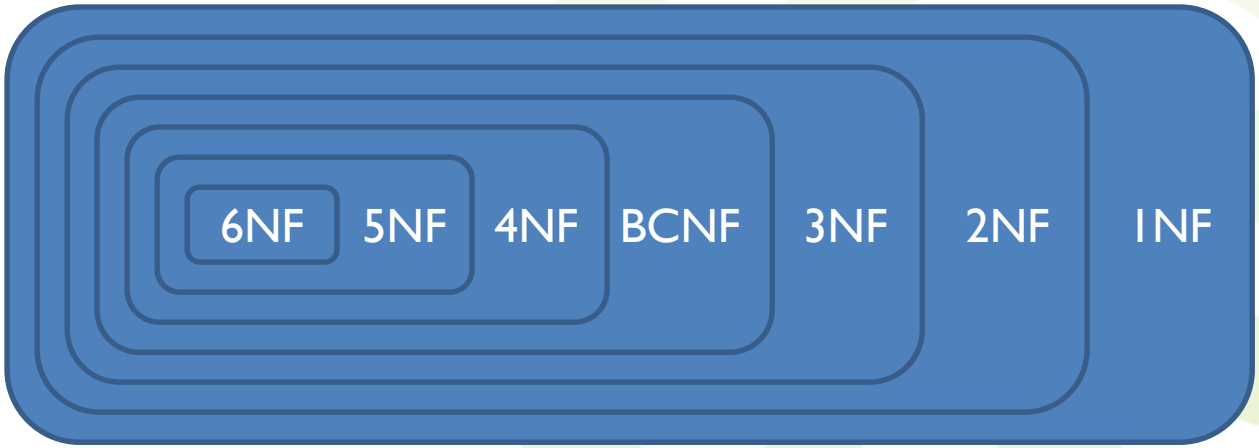
A	B	C	D	E
1	1	1	1	1
1	2	2	2	1
2	1	3	3	1
2	1	4	3	1
3	2	5	1	1

10.4 Normal Forms

- Back to **normalization:**
  - Remember:
    Normalization = Finding lossless decompositions
  - But only decompose, if the relation schema is of "bad quality"
- How to measure the **quality** of a relation schema?
  - Clearly: The quality depends on the constraints
  - In our case:
    Quality depends on the **FDs** of the relation schema
  - Schemas can be classified into different "quality levels," which are called **normal forms**

10.4 Normal Forms

- Part of a schema design process is to choose a desired normal form and convert the schema into that form
- There are **seven normal forms**
  - The higher the number, ...
    - ... the stricter the requirements,
    - ... the less anomalies and redundancy, and
    - ... the better the "design quality."



10.4 1NF

- **First normal form (1NF)**
  - Already known from previous lectures
    - Has nothing to do with functional dependencies!
  - Restricts relations to being "flat"
    - Only atomic attribute values are allowed
  - Multi-valued attributes must be normalized, e.g., by
    A) Introducing a **new relation** for the multi-valued attribute
    B) **Replicating** the tuple for each multi-value
    C) introducing an **own attribute** for each multi-value (if there is a small maximum number of values)
  - Solution A is usually considered the best

10.4 INF

- A: Introducing a **new relation**
  - Uses old key and multi-attribute as composite key

heroID	heroName	powers
1	Storm	weather control, flight
2	Wolverine	extreme cellular regeneration
3	Phoenix	omnipotence, indestructibility, limitless energy manipulation

heroID	heroName
1	Storm
2	Wolverine
3	Phoenix

heroID	power
1	weather control
1	flight
2	extreme cellular regeneration
3	omnipotence
3	indestructibility
3	limitless energy manipulation

10.4 INF

- B: **Replicating** the tuple for each multi-value
  - Uses old key and multi-attribute as composite key

heroID	heroName	powers
1	Storm	weather control, flight
2	Wolverine	extreme cellular regeneration
3	Phoenix	omnipotence, indestructibility, limitless energy manipulation

heroID	heroName	powers
1	Storm	weather control
1	Storm	flight
2	Wolverine	extreme cellular regeneration
3	Phoenix	omnipotence
3	Phoenix	indestructibility
3	Phoenix	limitless energy manipulation

10.4 INF

- C: Introducing an **own attribute** for each multi-value

heroID	heroName	powers
1	Storm	weather control, flight
2	Wolverine	extreme cellular regeneration
3	Phoenix	omnipotence, indestructibility, limitless energy manipulation

heroID	heroName	power1	power2	power3
1	Storm	weather control	flight	NULL
2	Wolverine	cellular regeneration	NULL	NULL
3	Phoenix	omnipotence	indestructibility	limitless energy manipulation

10.4 2NF

- **The second normal form (2NF)**
  - The 2NF aims to avoid attributes that are functionally dependent on (proper) subsets of keys
  - **Remember:**
    - A set of attributes X is a **(candidate) key** if and only if $X \rightarrow \{A_1, ..., A_n\}$ is a valid FD
    - An attribute A_i is a **key attribute** if and only if it is contained in some key; otherwise, it is a **non-key attribute**
  - **Definition (2NF):**
    A relation schema is in **2NF** (wrt. a set of FDs) iff ...
    - It is in 1NF and
    - **no non-key attribute is functionally dependent on a proper subset of some key.**

10.4 2NF

- Functional dependence on key parts is only a problem in relation schemas with composite keys
  - A key is called **composite key** if it consists of more than one attribute
- **Corollary:**
  Every 1NF-relation without constant attributes and without **composite keys** also is in 2NF.
  - 2NF is violated, if there is a **composite key** and some **non-key attribute** depends only on a **proper subset** of this composite key

10.4 2NF

- **Normalization** into 2NF is archived by **decomposition** according to the "non-2NF" FDs
  - If $X \rightarrow Y$ is a valid FD and X is a proper subset of some key, then decompose into $\alpha_1 = X \cup Y$ and $\alpha_2 = \{A_1, ..., A_n\} \setminus Y$
  - According to Heath's Theorem, this decomposition is **lossless**

FDs:
{heroID} → {heroName}
{teamID} → {teamName}
{heroID, teamID} → {joinYear}

Decompose with respect to
{heroID} → {heroName}

10.4 2NF

- **Repeat this decomposition step** for every created relation schema that is still not in 2NF

FDs:
{teamID} → {teamName}
{heroID, teamID} → {joinYear}

Decompose with respect to
{teamID} → {teamName}

10.4 3NF

- **The third normal form (3NF)**
  - The 3NF relies on the concept of **transitive FDs**
  - **Definition:**
    Given a set of FDs F, an FD $X \rightarrow Z \in F^+$ is **transitive** in F, if and only if there is an attribute set Y such that:
    - $X \rightarrow Y \in F^+$,
    - $Y \rightarrow X \notin F^+$, and
    - $Y \rightarrow Z \in F^+$.
  - **Example:**
    - {heroID} → {heroName}
    - {heroID} → {homeCityID}
    - {heroID} → {homeCityName}
    - {homeCityID} → {homeCityName}

heroID	heroName	homeCityID	homeCityName
11	Professor X	563	New York
12	Wolverine	782	Alberta
13	Cyclops	112	Anchorage
14	Phoenix	563	New York

10.4 3NF

- **Definition:**
  A relation schema is in 3NF if and only if:
  - It is 2NF and
  - no key transitively determines a non-key attribute.

heroID	heroName	homeCityID	homeCityName
11	Professor X	563	New York
12	Wolverine	782	Alberta
13	Cyclops	112	Anchorage
14	Phoenix	563	New York

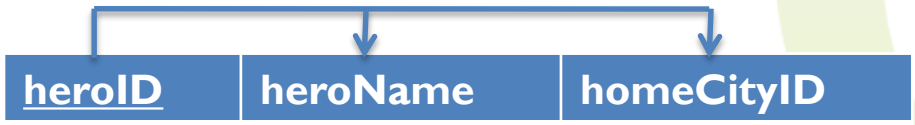
$$\{heroID\} \rightarrow \{heroName\}$$
$$\{heroID\} \rightarrow \{homeCityID\}$$
$$\{homeCityID\} \rightarrow \{homeCityName\}$$

10.4 3NF

- Assume that the "non-3NF" transitive FD $X \rightarrow Z$ has been created by FDs $X \rightarrow Y$ and $Y \rightarrow Z$
- Then, **normalization** into 3NF is archived by **decomposition** according to $Y \rightarrow Z$
  - Again, this decomposition is **lossless**

FDs:
{heroID} → {heroName}
{heroID} → {homeCityID}
{homeCityID} → {homeCityName}

Decompose with respect to
{homeCityID} → {homeCityName}



10.4 BCNF

- **Boyce-Codd normal form (BCNF)**
  - Was actually proposed by Ian Heath (he called it 3NF) three years before Boyce and Codd did
  - **Definition:**
    A relation schema *R* is in **BCNF** if and only if,
    in any **non-trivial** FD $X \rightarrow Y$, the set *X* is a **superkey**

- All BCNF schemas are also in 3NF,
  and most 3NF schemas are also in BCNF
  - There are some rare exceptions

10.4 BCNF

– BCNF is very **similar** to **3NF:**

- **BCNF:**
  In any non-trivial FD $X \rightarrow Y$, the set X is a superkey.
- **3NF (alternative definition):**
  In any non-trivial FD $X \rightarrow Y$, the set X is a superkey, or the set Y is a subset of some key.

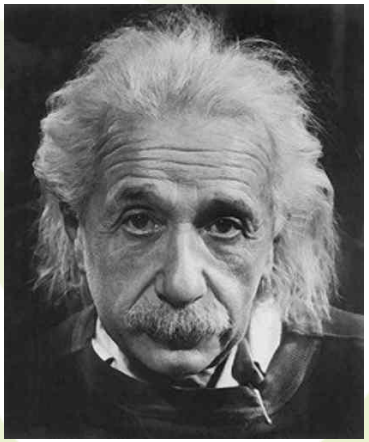
– A 3NF schema is **not in BCNF,** if it has two or more **overlapping composite keys.**

- That is: There are different keys X and Y such that $|X|, |Y| \geq 2$ and $X \cap Y \neq \emptyset$.

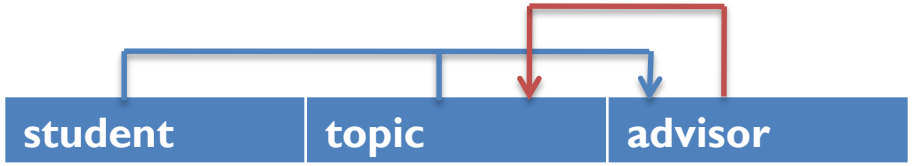
10.4 BCNF

- **Example:**
  - **Students,** a **topic**, and an **advisor**
  - Let's assume that the following dependencies hold:
    - {student, topic} → {advisor}
    - {advisor} → {topic}
  - That is: For each topic, a student has a specific advisor. Each advisor is responsible for a single specific topic.

student	topic	advisor
100	Math	Gauss
100	Physics	Einstein
101	Math	Leibniz
102	Math	Gauss



10.4 BCNF



– Consequently, there are the following **keys:**
  - {student, topic}
  - {student, advisor}

– The schema **is in 3NF,** because it is in 1NF and there are **no non-key attributes**
– However, it **is not in BCNF**
  - It is {advisor} → {topic} but {advisor} is not a superkey

10.4 BCNF

- Moreover, there are **modification anomalies:**

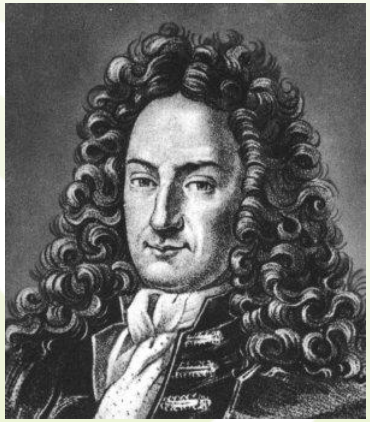
Student	Topic	Advisor
100	Math	Gauss
100	Physics	Einstein
101	Math	Leibniz
102	Math	Gauss

If you delete this row, all information about Leibniz doing math is lost

- Normalization by **decomposition** prevents these anomalies:

Student	Advisor
100	Gauss
100	Einstein
101	Leibniz
102	Gauss

Advisor	Topic
Gauss	Math
Einstein	Physics
Leibniz	Math



10.4 Higher Normal Forms

Detour

- BCNF is the "ultimate" normal form when using only functional dependencies as constraints
  - "Every attribute depends on a key, a whole key, and nothing but a key, so help me Codd."
- However, there are higher normal forms (4NF to 6NF) that rely on generalizations of FDs
  - 4NF: Multivalued dependencies
  - 5NF/6NF: Join dependencies

10.4 4NF

Detour

- The **4NF** is about **multivalued dependencies (MVDs)**
- **Example:**

course	teacher	textbook
Physics	Prof. Green	Basic Mechanics
Physics	Prof. Green	Principles of Optics
Physics	Prof. Brown	Basic Mechanics
Physics	Prof. Brown	Principles of Optics
Math	Prof. Green	Basic Mechanics
Math	Prof. Green	Vector Analysis
Math	Prof. Green	Trigonometry

Dependencies:

- "For any course, there is a fixed set of teachers." (written as **{course} ↠ {teacher}**)
- "For any course, there is a fixed set of textbooks, which is independent of the teacher" (written as **{course} ↠ {textbook}**)

- In fact, every FD can be expressed as a MVD:
  - If $X \rightarrow Y$ then also $X \twoheadrightarrow Y$
  - But both expressions are not equivalent!

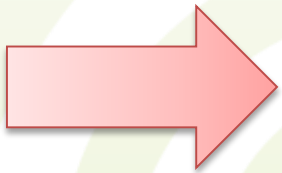
10.4 4NF

Detour

- **Definition:**
  A relation schema is in 4NF if and only if, for any non-trivial multivalued dependency $X \twoheadrightarrow Y$, also the functional dependency $Z \rightarrow Y$ holds, for some key Z

- **Decomposition into 4NF schemas:**

course	teacher	textbook
Physics	Prof. Green	Basic Mechanics
Physics	Prof. Green	Principles of Optics
Physics	Prof. Brown	Basic Mechanics
Physics	Prof. Brown	Principles of Optics
Math	Prof. Green	Basic Mechanics
Math	Prof. Green	Vector Analysis
Math	Prof. Green	Trigonometry



course	teacher
Physics	Prof. Green
Physics	Prof. Brown
Math	Prof. Green

course	textbook
Physics	Basic Mechanics
Physics	Principles of Optics
Math	Basic Mechanics
Math	Vector Analysis
Math	Trigonometry

10.4 5NF

Detour

- The **5NF** deals with **join dependencies (JDs)**
  - Directly related to **lossless decompositions**
  - **Definition:**
    Let $\alpha_1, ..., \alpha_k \subseteq \{A_1, ..., A_n\}$ be k subsets of R's attributes (possibly overlapping). We say that R satisfies the join dependency $*\{\alpha_1, ..., \alpha_k\}$ if and only if $\alpha_1, ..., \alpha_k$ is a lossless decomposition of R.
  - **Definition:**
    A relation schema is in 5NF if and only if, for every non-trivial join dependency $*\{\alpha_1, ..., \alpha_k\}$, **each α_i is a superkey.**

10.4 6NF

Detour

- The **6NF** also is about **join dependencies**
  - **Definition:**
    A relation schema is in 6NF if and only if it satisfies **no non-trivial JDs** at all.
    - In other words: You cannot decompose it anymore.
- Decomposition into 6NF means that every resulting relation schema contains a key and one(!) additional non-key attribute
  - This means **a lot of tables!**
- By definition, **6NF** is the final word on normalization by lossless decomposition
  - All kinds of dependencies can be expressed by **key and foreign key constraints**

10.5 Denormalization

- Normalization in **real world databases:**
  - Guided by normal form theory
  - But: Normalization is not everything!
  - Trade-off: Redundancy/anomalies vs. speed
    - General design: Avoid redundancy wherever possible, because redundancies often lead to inconsistent states
    - An exception: Materialized views (≈ precomputed joins) – expensive to maintain, but can boost read efficiency

10.5 Denormalization

- Usually, a schema in a **higher normal form** is **better** than one in a **lower normal form**
  - However, sometimes it is a good idea to artificially create lower-form schemas to, e.g., increase read performance
    - This is called **denormalization**
    - Denormalization usually **increases query speed** and **decreases update efficiency** due to the introduction of redundancy

10.5 Denormalization

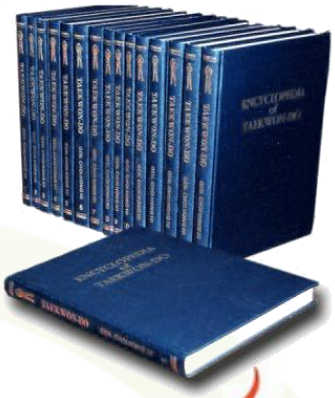
- Rules of thumb:
  - A **good data model** almost always directly leads to relational schemas in high normal forms
    - Carefully design your models!
    - Think of dependencies and other constraints!
    - Have normal forms in mind during modeling!
  - Denormalize only when faced with a performance problem that cannot be resolved by:
    - money
    - hardware scalability
    - current SQL technology
    - network optimization
    - parallelization
    - other performance techniques

10.5 Denormalization

- Sometimes, you even can perform denormalization at the **physical level** of the database
  - Let your RDBMS know what attributes are often are accessed together, even if they are located in different tables
  - State-of-the-art RDBMS can exploit this information to physically cluster data or precompute some joins, even without changing your table designs!

Next Week

- Advanced **database concepts** for application programming
  - **Views**
  - **Indexes**
  - **Transactions**



- **Accessing databases** from applications
  - Embedded SQL
  - SQLJ