ifis
Institut für Informationssysteme
Technische Universität Braunschweig

Relational Database Systems I

Wolf-Tilo Balke

Niklas Kiehne, Enrique Pinto Dominguez

Institut für Informationssysteme

Technische Universität Braunschweig

http://www.ifis.cs.tu-bs.de

10.0 Introduction

- Up to now, we have learned ...
  - …how to model schemas from a conceptual point of view
  - ... 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?**

10.0 Introduction

- Which table design is better?

A

member_id	club_id	member_name	club_name	join_year
1	1	Florian Flaschenbaum	Garden Groupies	1992
2	2	Hermann Heidelbeer	Fern Fans	1980
3	1	Denis Douglasie	Garden Groupies	1993
4	1	Niklas Nigella	Garden Groupies	1994
5	1	Regine Ringelblum	Garden Groupies	1965
6	2	Tilo Tanne	Fern Fans	1971

B

member_id	member_name
1	Florian Flaschenbaum
2	Hermann Heidelbeer
3	Denis Douglasie
4	Niklas Nigella
5	Regine Ringelblum
6	Tilo Tanne

club_id	club_name
1	Garden Groupies
2	Fern Fans

member_id	club_id	join_year
1	1	1992
2	2	1980
3	1	1993
4	1	1994
5	1	1965
6	2	1971

10.0 Introduction

A

member_id	club_id	member_name	club_name	join_year
1	1	Florian Flaschenbaum	Garden Groupies	1992
2	2	Hermann Heidelbeer	Fern Fans	1980
3	1	Denis Douglasie	Garden Groupies	1993
4	1	Niklas Nigella	Garden Groupies	1994
5	1	Regine Ringelblum	Garden Groupies	1965
6	2	Tilo Tanne	Fern Fans	1971

- What's wrong with design A?
  - **redundancy:** the club names are stored several times
  - **inferior expressiveness:** we cannot nicely represent clubs that currently have no members.
  - **modification anomalies:** (see next slide)

10.0 Introduction

A

member_id	club_id	member_name	club_name	join_year
1	1	Florian Flaschenbaum	Garden Groupies	1992
2	2	Hermann Heidelbeer	Fern Fans	1980
3	1	Denis Douglasie	Garden Groupies	1993
4	1	Niklas Nigella	Garden Groupies	1994
5	1	Regine Ringelblum	Garden Groupies	1965
6	2	Tilo Tanne	Fern Fans	1971

- There are three kinds of modification anomalies
  - **insertion anomalies**
    - how do you add clubs that currently have no member?
    - how do you (consistently!) add new tuples?
  - **deletion anomalies**
    - deleting *Hermann Heidelbeer* and *Tilo Tanne* also deletes all information about the *Fern Fans*
  - **update anomalies**
    - renaming a club requires updating several tuples (due to redundancy)

10.0 Introduction

- In general, ***good* relational database designs** have the following properties
  - redundancy is minimized
    - i.e. no information is represented several times!
    - logically distinct information is placed in distinct relation schemes
  - modification anomalies are prevented *by design*
    - i.e. 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
    - ….this, however, might in some cases mean that redundancy is beneficial
      - It's quite tricky to find the proper balance between different optimization goals
- In essence, it's all about splitting up tables ...
  - remember design B

10 Normalization

- **Normalization**
- Functional dependencies
- Normal forms
  - 1NF, 2NF, 3NF, BCNF
  - Higher normal forms
- Denormalization

member_id	member_name
1	Florian Flaschenbaum
2	Hermann Heidelbeer
3	Denis Douglasie
4	Niklas Nigella
5	Regine Ringelblum
6	Tilo Tanne

member_id	club_id	join_year
1	1	1992
2	2	1980
3	1	1993
4	1	1994
5	1	1965
6	2	1971

club_id	club_name
1	Garden Groupies
2	Fern Fans

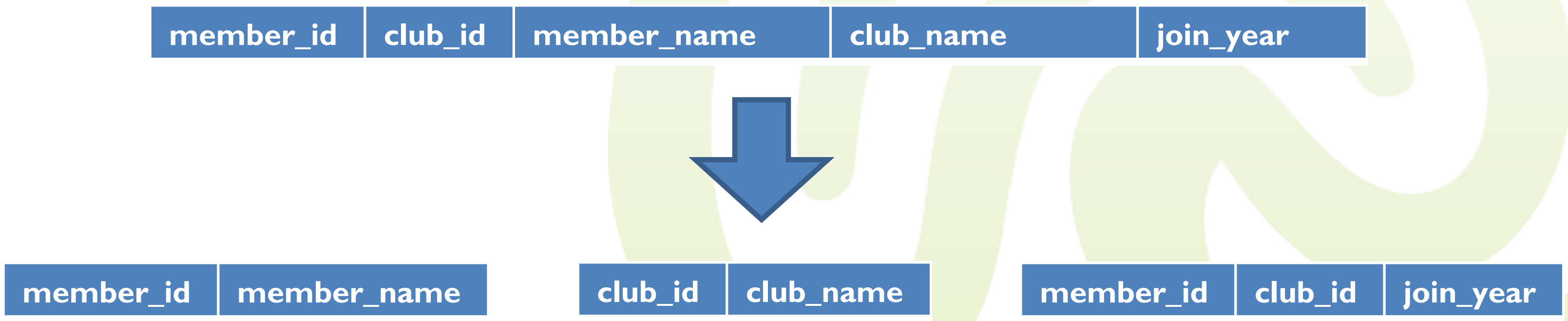
10.1 Normalization

- The *rules of thumb* for *good database design* 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. *member_id is unique* and *member_id determines member_name*)
  - a **relational database instance** (or extension) is
    - a set of tuples adhering to the respective schemas and respecting all integrity constraints

10.1 Normalization

- For this lecture, let's assume the following
  - $S(R_1, \ldots, R_n)$ is a **relation schema**
  - $R(A_1, \ldots, A_n)$ 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:



10.1 Normalization

- **Definition (decomposition)**
  - let $\alpha_1, \ldots, \alpha_k \subseteq \{A_1, \ldots, A_n\}$ 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:

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

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

- *Good* decompositions have to be **reversible**
  - the decomposition $\alpha_1, \ldots, \alpha_k$ 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 $\mathcal{C}$

10.1 Normalization

- Example

$\mathcal{C}$ = {"{member_id, club_id} is unique",
"member_id determines member_name",
"club_id determines club_name",
"{member_id, club_id} determines join_year"}

Club_Members

member_id	club_id	member_name	club_name	join_year
1	1	Florian Flaschenbaum	Garden Groupies	1992
2	2	Hermann Heidelbeer	Fern Fans	1980
3	1	Denis Douglasie	Garden Groupies	1993
4	1	Niklas Nigella	Garden Groupies	1994
5	1	Regine Ringelblum	Garden Groupies	1965
6	2	Tilo Tanne	Fern Fans	1971

– our example decomposition is lossless

α_1 = {member_id, member_name}, α_2 = {club_id, club_name}, α_3 = {member_id, club_id, join_year}

π_{α_1}(Club_Members)

member_id	member_name
1	Florian Flaschenbaum
2	Hermann Heidelbeer
3	Denis Douglasie
4	Niklas Nigella
5	Regine Ringelblum
6	Tilo Tanne

π_{α_2}(Club_Members)

club_id	club_name
1	Garden Groupies
2	Fern Fans

π_{α_3}(Club_Members)

member_id	club_id	join_year
1	1	1992
2	2	1980
3	1	1993
4	1	1994
5	1	1965
6	2	1971

10.1 Normalization

- **Normalizing** a relation schema *S* means replacing relations in *S* by lossless decompositions
- However, this raises some new questions
  - under which conditions is there a (nontrivial) lossless decomposition?
    - decompositions involving $\alpha_i = \{A_1, \ldots, A_n\}$ 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*...

10.1 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)**
  - *member_name is completely determined by member_id* is an example for a functional dependency
  - most update anomalies and problems with redundancy occurring in practice can be traced back to violations of functional dependency constraints
    - typically, functional dependencies are all you need

10 Normalization

- Normalization
- **Functional dependencies**
- Normal forms
  - 1NF, 2NF, 3NF, BCNF
  - Higher normal forms
- Denormalization

10.2 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}*

10.2 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:

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

10.2 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 $X \rightarrow Y$

10.2 Functional Dependencies

- Functional dependencies are semantic properties of the underlying domain and data model
  - They depend on real world knowledge
- 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*

10.2 Functional Dependencies

- In fact, functional dependencies are a generalization of **key constraints**
- To show 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
    - i.e. only uniqueness is required

10.2 Functional Dependencies

- In practice, if there is more than one candidate key, we usually choose one and call it the **primary key**
  - however, for normalization purposes, only candidate keys are important – thus, **we ignore primary keys** today

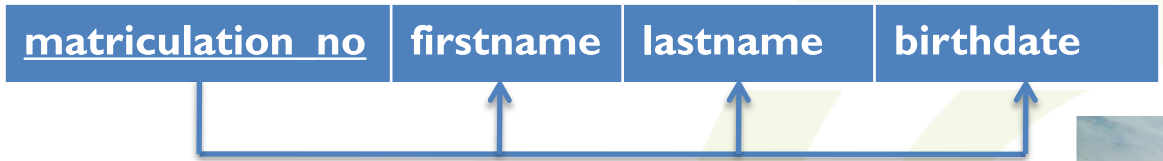
- The following is true
  - X is a superkey of R

    if and only if

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

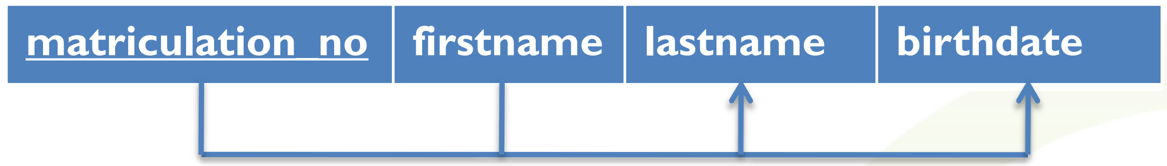
10.2 Functional Dependencies

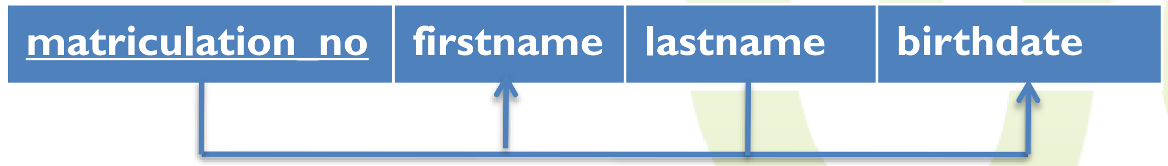
- Example
  - a relation containing students
    - semantics: matriculation_no is **unique**
    - {matriculation_no} → {firstname, lastname, birthdate}



10.2 Functional Dependencies

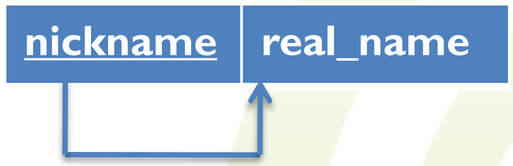
- Obviously, there can also be non-minimal super keys with correct functional dependencies





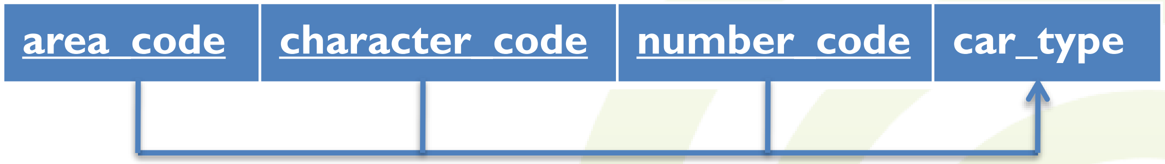
10.2 Functional Dependencies

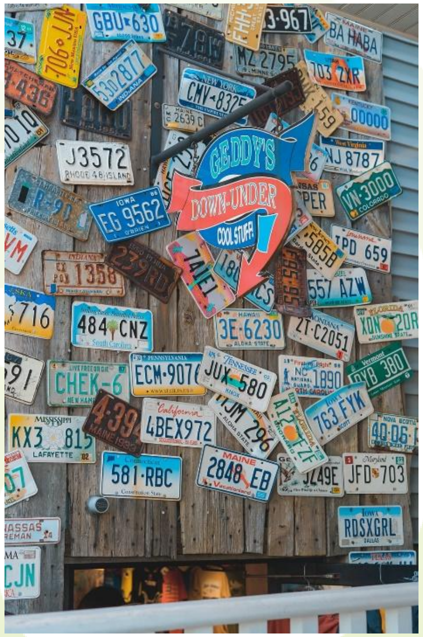
- Example
  - a relation containing real names and nicknames of members, where each member has only one unique nickname
    - {nickname} → {real_name}



10.2 Functional Dependencies

- Example
  - a relation containing license plates and the type of the respective car
    - {area_code, character_code, number_code} → {car_type}





10.2 Functional Dependencies

- Quick Summary on keys:
  - **Candidate Key** (or simply key)
    - A **irreducible** set of attributes which **uniquely** identifies a tuple
      - i.e.: all non-key attributes are functional dependent on the key, and no attribute can be removed without loosing the key properties
  - **Superkey** is a superset of a candidate key
    - i.e. only uniqueness is required
      - Superkey also identifies a tuple, but is not irreducible
  - **Primary Key**
    - A primary key is one single key chosen from the set of candidate keys by the database designer
      - This choice impacts the way the DBMS manages relations and queries

10.2 Functional Dependencies

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

street	city	state	zip

  - for any given zip code, there is exactly one city and state
    - …which, to be exact, is not true in reality
  - for any given street, city, and state, there is exactly one zip code.

- What are the FDs and candidate keys?

10.2 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} → ∅
  - …

street	city	state	zip

10.2 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

10.2 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*
  - *Abstract Definition: F* **implies** $X \rightarrow Y$, if and only if any extension of *R* satisfying any FD in *F*, also satisfies the $X \rightarrow Y$
- Fortunately, the closure of *F* can easily be computed using a small set of **inference rules**

10.2 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

10.2 Functional Dependencies

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

10.2 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)

10.2 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 full closure
  - instead, it often suffices to compute a certain subset of the closure: the subset consisting of all FDs with given left side
    - This will later serve for finding proper keys or normalizing relations

10.2 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*
  - i.e. $(X, F)^+ := \{A_i \mid X \rightarrow A_i \text{ is implied by } F\}$
- The following algorithm computes $(X, F)^+$:

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

10.2 Functional Dependencies

- Example
  - $F = \{ \{A\} \to \{B, C\}, \{B\} \to \{E\}, \{E\} \to \{C, F\}, \{C, D\} \to \{E, F\} \}$
  - *What is the closure of {A, B} under F?*

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

10.2 Functional Dependencies

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

    Intermediate Closure: {A, B}

    Add C, because $\{A\} \rightarrow \{B, C\}$ — {A, B, C}

    Add E, because $\{B\} \rightarrow \{E\}$ — {A, B, C, E}

    Add F, because $\{E\} \rightarrow \{C, F\}$

    $(\{A, B\}, F)^{+} = \{A, B, C, E, F\}$

10.2 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 $(X, F)^+ = \{A_1, \ldots, A_n\}$
- What's still missing?
  - given a set of FDs *F*, how to find a set of FDs *G*, such that $\boldsymbol{F^+ = G^+}$, and *G* is **as small as possible?**
  - given sets of FDs *F* and *G*, does $\boldsymbol{F^+ = G^+}$ hold?

10.2 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

10.2 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}

  - $(\{A, B\}, F)^+ = \{A, B, C\}$ $(\{A, B\}, G)^+ = \{A, B, C\}$
  - $(\{C\}, F)^+ = \{B, C\}$ $(\{C\}, G)^+ = \{C\}$

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

10.2 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**
    - i.e. *Y* consists of exactly one attribute
  - every FD $X \rightarrow Y$ in *F* is **left-irreducible**
    - i.e. no attribute can be removed from *X* without changing F^+
  - every FD $X \rightarrow Y$ in *F* is **non-redundant**
    - i.e. $X \rightarrow Y$ cannot be removed from *F* without changing F^+

10.2 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^+).

10.2 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)

10.2 Functional Dependencies

- Now we have:
  {A} → {B}, {A} → {C}, {B} → {C},
  {A, B} → {C}, {A} → {D}

3. Remove {A, B} → {C}:
   - {A} → {C} implies {A, B} → {C}
4. Remove {A} → {C}:
   - {A} → {B} and {B} → {C} imply {A} → {C} (transitivity)

- Finally, we end up with a **minimal equivalent** of *F*:
  {A} → {B}, {B} → {C}, {A} → {D}

10.2 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 $R(A_1, \ldots, A_n)$. Then, the following decomposition of R is **lossless:**

    $$\alpha_1 = X \cup Y \quad \text{and} \quad \alpha_2 = \{A_1, \ldots, A_n\} \setminus Y.$$
  - **Example:**

FDs:
{member_id} → {member_name}
{club_id} → {club_name}
{member_id, club_id} → {join_year}

member_id	club_id	member_name	club_name	join_year

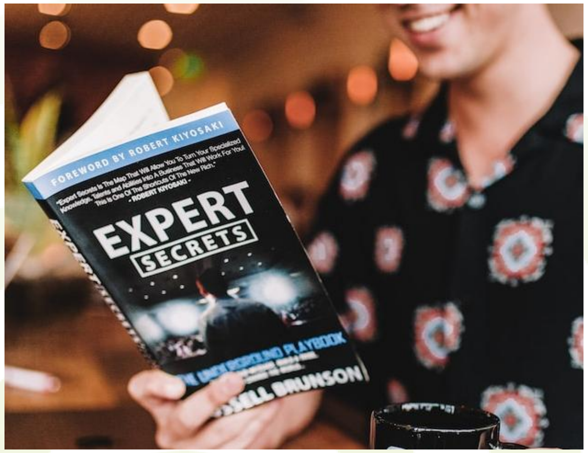
Decompose with respect to
{member_id} → {member_name}

member_id	member_name

member_id	club_id	club_name	join_year

10.2 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

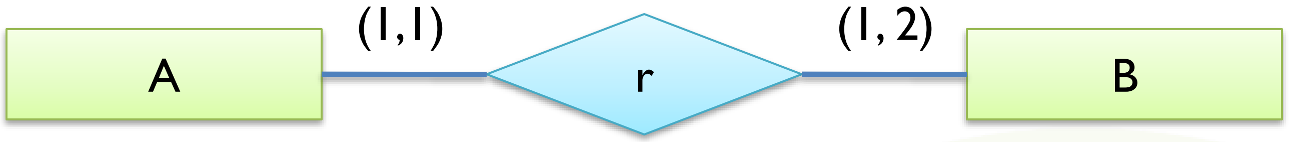


1. Based on *domain knowledge*
   - *obvious* FDs are easy to find
   - what about more complicated FDs?
   - no guarantee that you found all (important) FDs!

10.2 Functional Dependencies

Detour

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?

10.2 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

10.2 Functional Dependencies

Detour

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} → {A, B}
  b) {A, D} → {C}
  c) Ø → {E}

10.2 Functional Dependencies

Detour

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) $\{C\} \rightarrow \{A, B\}$: Yes

  b) $\{A, D\} \rightarrow \{C\}$: No

  c) $\emptyset \rightarrow \{E\}$: Yes

10 Normalization

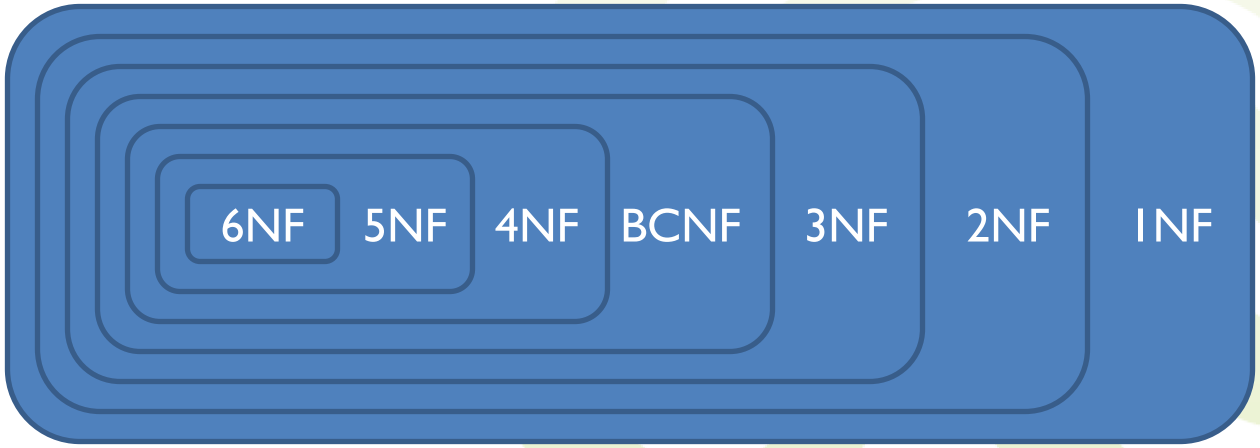
- Normalization
- Functional dependencies
- **Normal forms**
  - 1NF, 2NF, 3NF, BCNF
  - Higher normal forms
- Denormalization

10.3 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?
  - claim: 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**

10.3 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*.
      - (well, from a theoretical point of view; in the real world, there might be good reasons why a lower normal formal is better….)



10.3 1NF

attribute

- **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
       - most common solution

    b) **replicating** the tuple for each multi-value
       - as e.g., often done for song list metadata (e.g., mp3 tags)

    c) introducing an **own attribute** for each multi-value
    (if there is a small maximum number of values)
       - as sometimes done in Big Data Database (e.g., Bigtable)

10.3 1NF

- a) Introducing a **new relation**
  - uses old key and multi-attribute as composite key

member_id	member_name	hobbies
1	Florian Flaschenbaum	tuba player, dike hiking
2	Hermann Heidelbeer	guinea pigs
3	Denis Douglasie	archery, history, cooking

member_id	member_name
1	Florian Flaschenbaum
2	Hermann Heidelbeer
3	Denis Douglasie

member_id	hobby
1	tuba player
1	dike hiking
2	guinea pigs
3	archery
3	history
3	cooking

10.3 1NF

- b) **Replicating** the tuple for each multi-value
  - uses old key and multi-attribute as composite key

member_id	member_name	hobbies
1	Florian Flaschenbaum	tuba player, dike hiking
2	Hermann Heidelbeer	guinea pigs
3	Denis Douglasie	archery, history, cooking

member_id	member_name	hobby
1	Florian Flaschenbaum	tuba player
1	Florian Flaschenbaum	dike hiking
2	Hermann Heidelbeer	guinea pigs
3	Denis Douglasie	archery
3	Denis Douglasie	history
3	Denis Douglasie	cooking

10.3 1NF

- c) Introducing an **own attribute** for each multi-value

member_id	member_name	hobbies
1	Florian Flaschenbaum	tuba player, dike hiking
2	Hermann Heidelbeer	guinea pigs
3	Denis Douglasie	archery, history, cooking

$\Downarrow$

member_id	member_name	hobby1	hobby2	hobby3
1	Florian Flaschenbaum	tuba player	dike hiking	NULL
2	Hermann Heidelbeer	guinea pigs	NULL	NULL
3	Denis Douglasie	archery	history	cooking

10.3 2NF

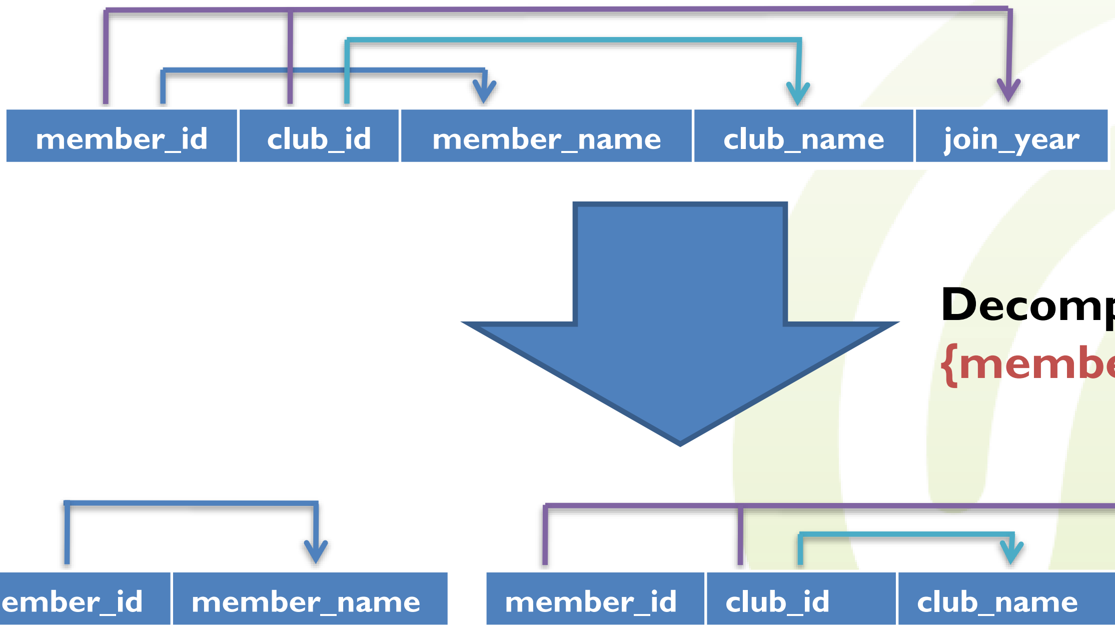
- **The second normal form (2NF)**
  - the 2NF aims to avoid attributes that are functionally dependent on proper subsets of keys
  - **remember**
    - a set of attributes X is a **(candidate) key** if and only if $X \rightarrow \{A_1, \ldots, A_n\}$ is a valid FD
    - an attribute A_i is a **key attribute** 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 any candidate key.**

10.3 2NF

- Functional dependence on key parts is only a problem in relation schemas with composite keys
  - a (candidate) key is called **composite key** if it consists of more than one attribute
- **Corollary:**
  Every 1NF-relation without constant attributes and without **composite keys** 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

10.3 2NF

- **Normalization** into 2NF is achieved 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 $\alpha_2 = \{A_1, \ldots, A_n\} \setminus Y$
  - according to Heath's Theorem, this decomposition is **lossless**

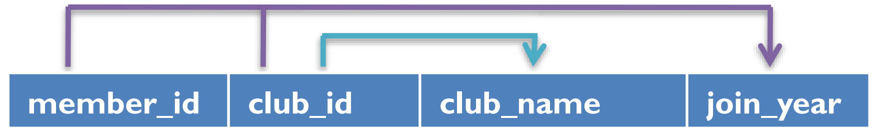


FDs:
{member_id} → {member_nan
{club_id} → {club_name}
{member_id, club_id} → {join_yea

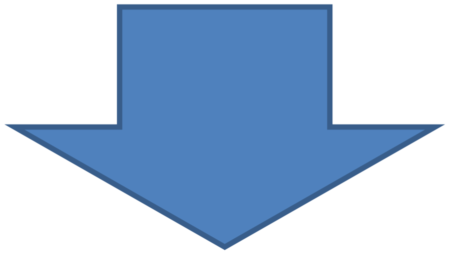
Decompose with respect to
{member_id} → {member_name}

10.3 2NF

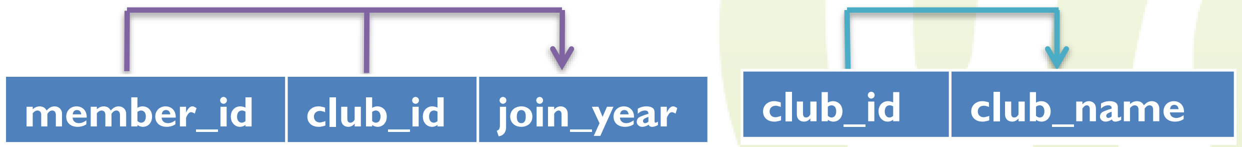
- **Repeat this decomposition step** for every created relation schema that is still not in 2NF



FDs:
{club_id} → {club_name}
{member_id, club_id} → {join_year}



Decompose with respect to
{club_id} → {club_name}



10.3 2NF

- Practical Implication of 2NF:
  - Normalized tables tend to focus on a single topic
    - Other topics are usually pulled in own tables
    - Some topic mixes remain

10.3 3NF

- **The third normal form (3NF)**
  - **Most relevant and practical normal form!**
  - A relation schema is in 3NF if and only if:
    - it is in 2NF and
    - all non-key attributes are determined ONLY by candidate keys.

member_id	member_name	home_city_id	home_city_name
11	Agathe Apfel	753	Aachen
12	Cedric Citrus	112	Copenhagen
13	Paul Pflaume	983	Potsdam
14	Aurelia Ahorn	753	Aachen

{member_id} → {member_name}
{member_id} → {home_city_id}
{home_city_id} → {home_city_name}

Not in 3NF

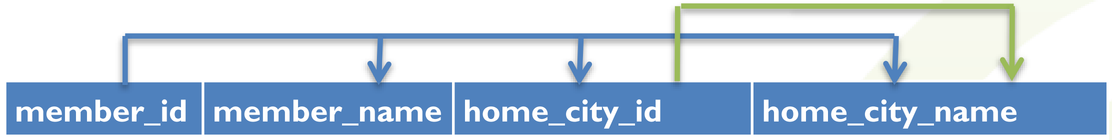
10.3 3NF

- the 3NF relies on the concept of **transitive FDs**
  - **Definition transitive FDs:**
    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
    - $Y \rightarrow Z \in F^+$.
  - No non-key attribute is transitively dependent on a key attribute
- **Example**
  - {member_id} → {member_name}
  - {member_id} → {home_city_id}
  - {member_id} → {home_city_name}
  - {home_city_id} → {home_city_name}

m._id	member_name	home_city_id	home_city_name
11	Agathe Apfel	753	Aachen
12	Cedric Citrus	112	Copenhagen
13	Paul Pflaume	983	Potsdam
14	Aurelia Ahorn	753	Aachen

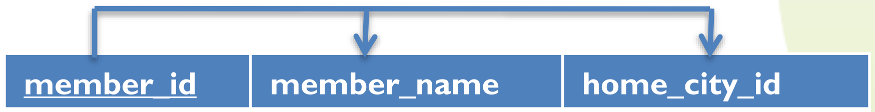
10.3 3NF

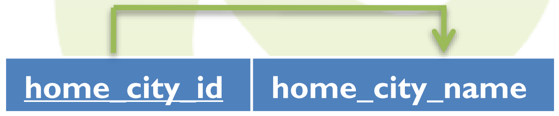
- Assume that the *non-3NF* transitive FD $X \rightarrow Z$ has been created by FDs $X \rightarrow Y$ and $Y \rightarrow Z$
- Then, **normalization** into 3NF is archived by **decomposition** according to $Y \rightarrow Z$
  - again, this decomposition is **lossless**



FDs:
{member_id} → {member_name}
{member_id} → {home_city_id}
{home_city_id} → {home_city_name}

Decompose with respect to
{home_city_id} → {home_city_name}





10.3 BCNF

- **Boyce-Codd normal form (BCNF)**
  - was actually proposed by Ian Heath (he called it 3NF) three years before Boyce and Codd
  - **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

10.3 BCNF

- BCNF is very **similar** to **3NF:**
  - **BCNF:**
    In any non-trivial FD $X \longrightarrow Y$, the set X is a superkey.
  - **3NF (alternative definition):**
    In any non-trivial FD $X \longrightarrow Y$, the set X is a superkey, or each attribute in Y is a key attribute.

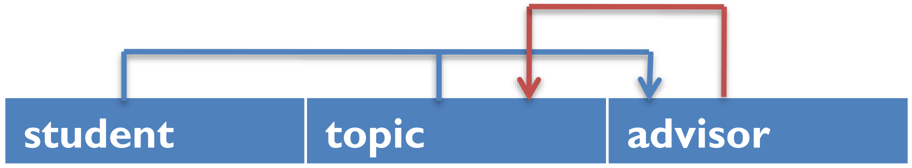
- a 3NF schema is **not in BCNF,** if it has two or more **overlapping composite keys.**
  - i.e. there are different keys X and Y such that $|X|, |Y| \geq 2$ and $X \cap Y \neq \emptyset$.

10.3 BCNF

- **Example**
  - **Students,** a **topic**, and an **advisor**
  - let's assume that the following dependencies hold
    - {student, topic} → {advisor}
    - {advisor} → {topic}
  - i.e. *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

10.3 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**
  - We have {advisor} → {topic} but {advisor} is not a superkey

10.3 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

– Deleting the last student of an advisor?

– An advisor changes his topic?

  - Because {Advisor} → {Topic}, multiple updates necessary

10.3 BCNF

- What options do we have?
  - decompose into one of
student	topic

      and

student	advisor
topic	advisor

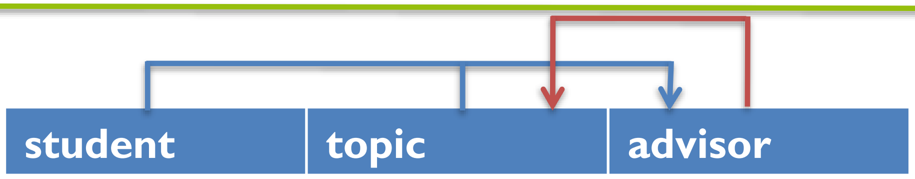
      and

topic	student
advisor	topic

      and

advisor	student
  - Which one to chose?
  - {Student, Topic} → {Advisor} is “lost“ in all options

10.3 BCNF



- In any case, we should perform a lossless decomposition
  - Apply Heath's theorem w.r.t. {advisor} → {topic}
    - => advisor topic and advisor student
  - All other decompositions can produce false tuples when rejoining
  - Multiple advisors in the same topic possible
  - Completeness of FDs was traded against a higher normal form

advisor	topic
Gauss	Math
Einstein	Physics
Leibniz	Math

advisor	student
Gauss	100
Einstein	100
Leibniz	101
Gauss	102

10.3 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

10.3 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}** $\twoheadrightarrow$ **{teacher}**)
- *For any course, there is a fixed set of textbooks, which is independent of the teacher.* (written as **{course}** $\twoheadrightarrow$ **{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!

10.3 4NF

Detour

- **Definition:**
  A relation schema is in 4NF if and only if, for any non-trivial multivalued dependency $X \twoheadrightarrow Y$, X is a superkey.

- **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

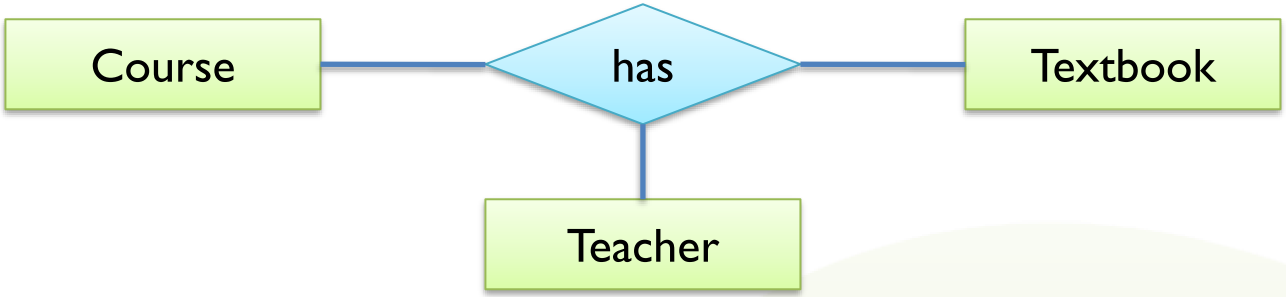
course	teacher
Physics	Prof. Green
Physics	Prof. Brown
Math	Prof. Green

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

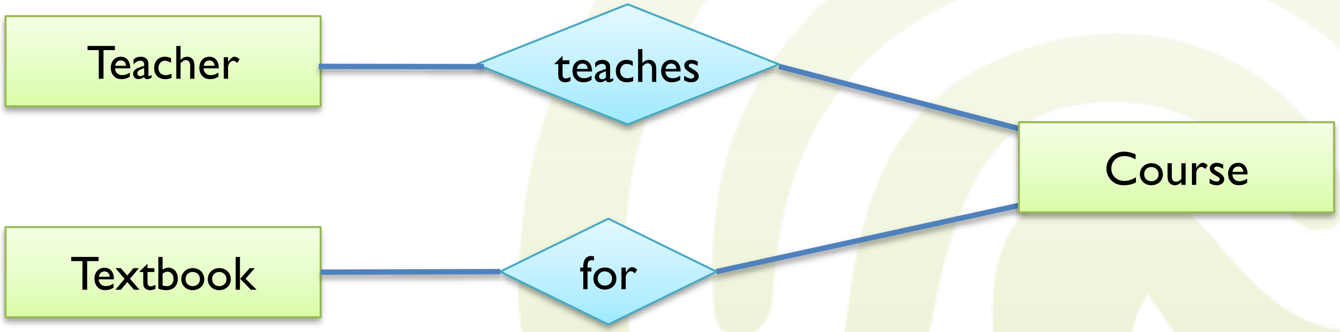
10.3 4NF

Detour

- Result from a bad conceptual schema:



- Instead of



10.3 5NF

Detour

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

10.3 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 Normalization

- Normalization
- Functional dependencies
- Normal forms
  - 1NF, 2NF, 3NF, BCNF
  - Higher normal forms
- **Denormalization**

10.4 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
    - Also: distributed and parallel databases
      - Here, redundancy is a good thing and increases data reliability and query speeds!
        - » but creates huge problems when faced with updates…

10.4 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 sometimes **increases query speed** and **decreases update efficiency** due to the introduction of redundancy

10.4 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

10.4 Denormalization

- sometimes, you can perform denormalization even at the **physical level** of the database
  - let your RDBMS know what attributes are often 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!

10 Next Week

- Advanced **database concepts** for application programming
  - **Views**
  - **Indexes**
  - **Transactions**
- **Accessing databases** from applications
  - Embedded SQL
  - SQLJ