Web and Database Technology

Functional Dependencies & Schema Normalization

Dr. Albert John Power

a.j.power@tudelft.nl

TU Delft WIS Web Information Systems

Q/A Session on Wednesday!

- Ask your Questions:

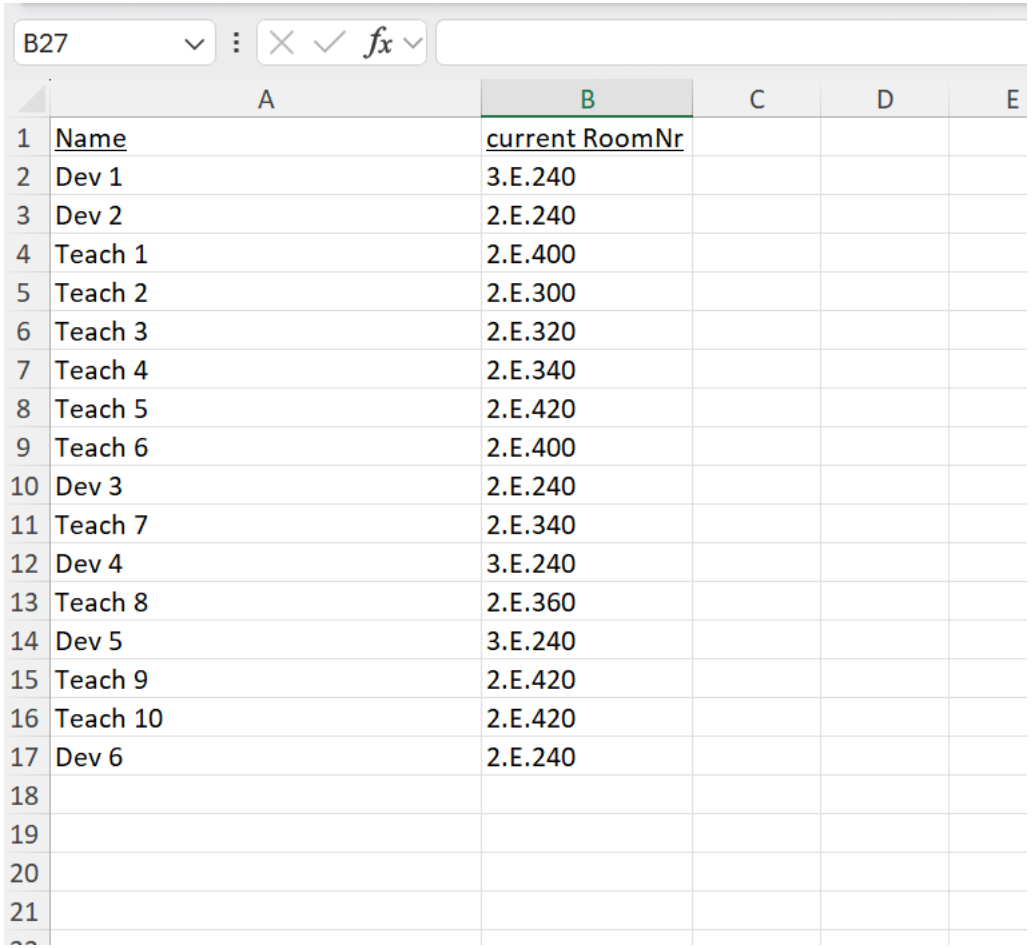
cse1500-ewi@tudelft.nl

Learning Objective

- Functional Dependencies & Schema Normalization
  - Develop logical database schema, with principled design that enforce data integrity

Example: Room Allocation

- "Make a room allocation of the teaching team" (lazy Excel approach)

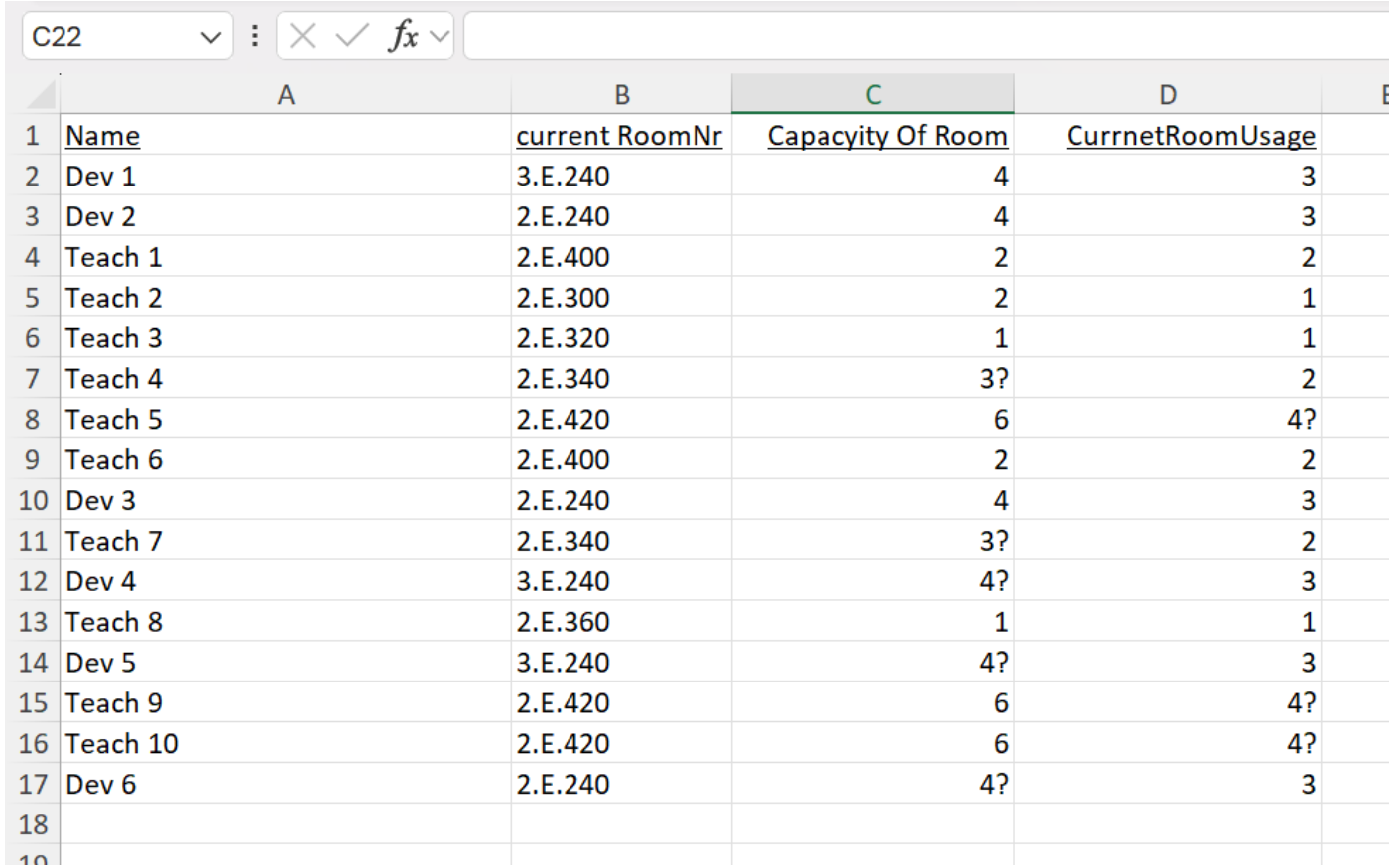


	A	B	C	D	E
1	Name	current RoomNr			
2	Dev 1	3.E.240			
3	Dev 2	2.E.240			
4	Teach 1	2.E.400			
5	Teach 2	2.E.300			
6	Teach 3	2.E.320			
7	Teach 4	2.E.340			
8	Teach 5	2.E.420			
9	Teach 6	2.E.400			
10	Dev 3	2.E.240			
11	Teach 7	2.E.340			
12	Dev 4	3.E.240			
13	Teach 8	2.E.360			
14	Dev 5	3.E.240			
15	Teach 9	2.E.420			
16	Teach 10	2.E.420			
17	Dev 6	2.E.240			
18					
19					
20					
21					

- Total number of rooms?
- Total capacity of each room?
- Current room usage?
- Rooms that are empty?

Example: Room Allocation

- “Can you add room capacity?!”



C22

	A	B	C	D
1	Name	current RoomNr	Capacyity Of Room	CurrnetRoomUsage
2	Dev 1	3.E.240	4	3
3	Dev 2	2.E.240	4	3
4	Teach 1	2.E.400	2	2
5	Teach 2	2.E.300	2	1
6	Teach 3	2.E.320	1	1
7	Teach 4	2.E.340	3?	2
8	Teach 5	2.E.420	6	4?
9	Teach 6	2.E.400	2	2
10	Dev 3	2.E.240	4	3
11	Teach 7	2.E.340	3?	2
12	Dev 4	3.E.240	4?	3
13	Teach 8	2.E.360	1	1
14	Dev 5	3.E.240	4?	3
15	Teach 9	2.E.420	6	4?
16	Teach 10	2.E.420	6	4?
17	Dev 6	2.E.240	4?	3
18				

Example: Room Allocation

- “Room 3.E.240 is for 4 people!”

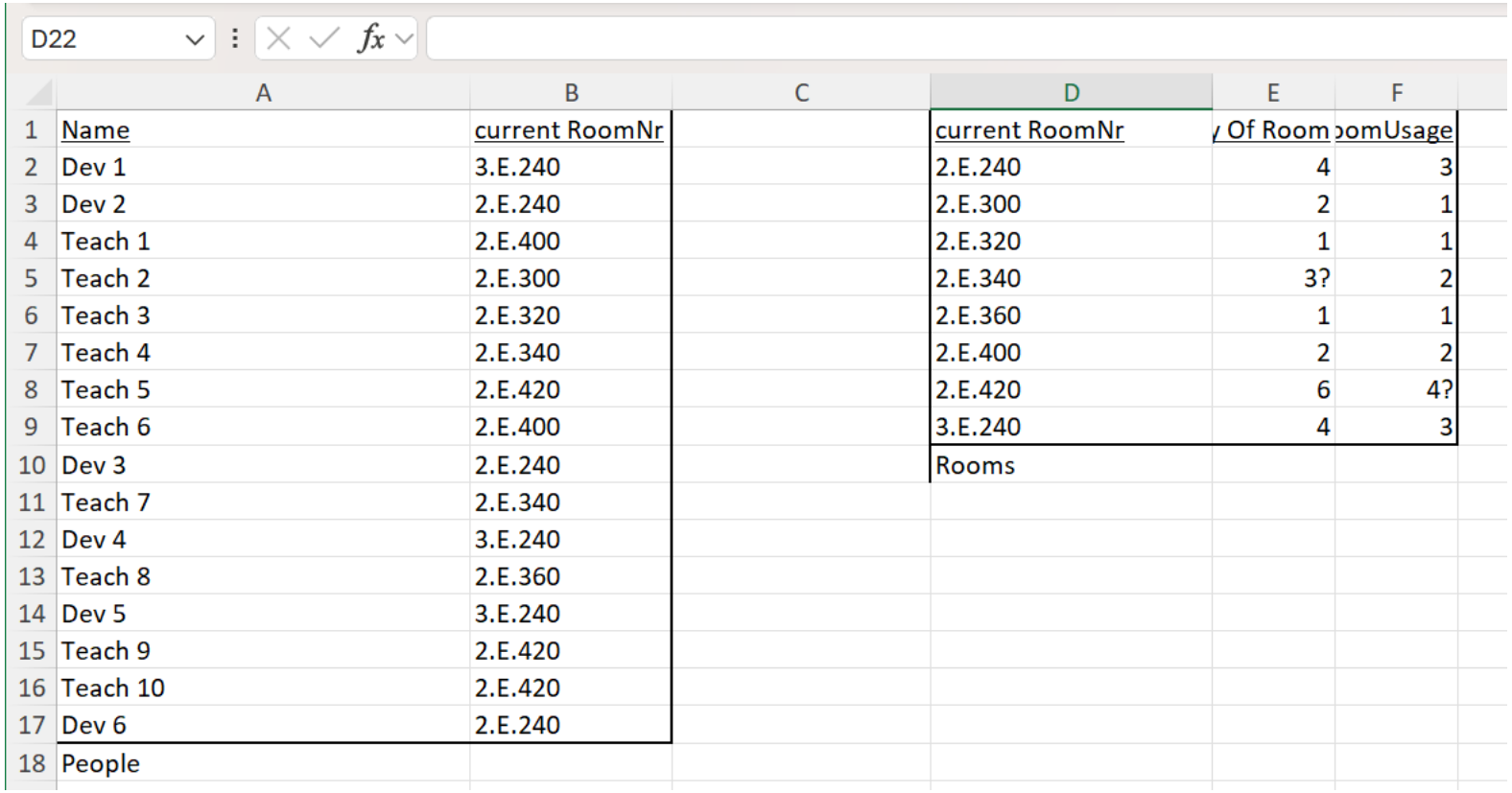
D22

	A	B	C	D	E	F
1	Name	current RoomNr	Capacyity Of Room	CurrnetRoomUsage		
2	Dev 1	3.E.240	4	3		
3	Dev 2	2.E.240	4	3		
4	Teach 1	2.E.400	2	2		
5	Teach 2	2.E.300	2	1		
6	Teach 3	2.E.320	1	1		
7	Teach 4	2.E.340	3?	2		
8	Teach 5	2.E.420	6	4?		
9	Teach 6	2.E.400	2	2		
10	Dev 3	2.E.240	4	3		
11	Teach 7	2.E.340	3?	2		
12	Dev 4	3.E.240	4?	3		
13	Teach 8	2.E.360	1	1		
14	Dev 5	3.E.240	4	3		
15	Teach 9	2.E.420	6	4?		
16	Teach 10	2.E.420	6	4?		
17	Dev 6	2.E.240	4?	3		
18						
19						
20						

- Change current room usage
- Change capacity or room name
- Empty a room
- Rename a room

Example: Room Allocation

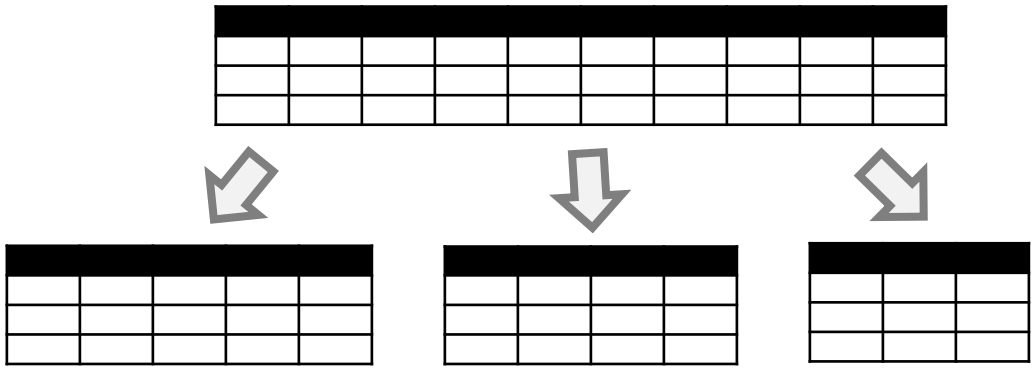
- “Oh! You ‘invented’ normalized Relational Tables…”



D22

	A	B	C	D	E	F
1	Name	current RoomNr		current RoomNr	y Of Room	oomUsage
2	Dev 1	3.E.240		2.E.240	4	3
3	Dev 2	2.E.240		2.E.300	2	1
4	Teach 1	2.E.400		2.E.320	1	1
5	Teach 2	2.E.300		2.E.340	3?	2
6	Teach 3	2.E.320		2.E.360	1	1
7	Teach 4	2.E.340		2.E.400	2	2
8	Teach 5	2.E.420		2.E.420	6	4?
9	Teach 6	2.E.400		3.E.240	4	3
10	Dev 3	2.E.240		Rooms		
11	Teach 7	2.E.340				
12	Dev 4	3.E.240				
13	Teach 8	2.E.360				
14	Dev 5	3.E.240				
15	Teach 9	2.E.420				
16	Teach 10	2.E.420				
17	Dev 6	2.E.240				
18	People					

Functional Dependencies & Schema Normalization



Recap

- Up to now, we have learned...
  - … how to model schemas from a conceptual point of view
  - … how to transform an ER schema to a logical one
  - ... how the relational model works.
- What's missing?
  - How to create a *good* database design?
  - By the way: What is a ***good* database design?**

DISCUSS

- Which table design is better?

A

hero_id	team_id	hero_name	team_name	join_year
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Black Widow	The Avengers	1975
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

B

hero_id	hero_name
1	Thor
2	Mister Fantastic
3	Iron Man
4	Black Widow
5	Captain America
6	Invisible Girl

team_id	team_name
1	The Avengers
2	Fantastic Four

hero_id	team_id	join_year
1	1	1963
2	2	1961
3	1	1963
4	1	1975
5	1	1964
6	2	1961

Introduction

A

hero_id	team_id	hero_name	team_name	join_year
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Black Widow	The Avengers	1975
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

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

Introduction

A

hero_id	team_id	hero_name	team_name	join_year
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Black Widow	The Avengers	1975
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)

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
      - Do not misunderstand this as “never ever ever have any redundancy”

  - **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

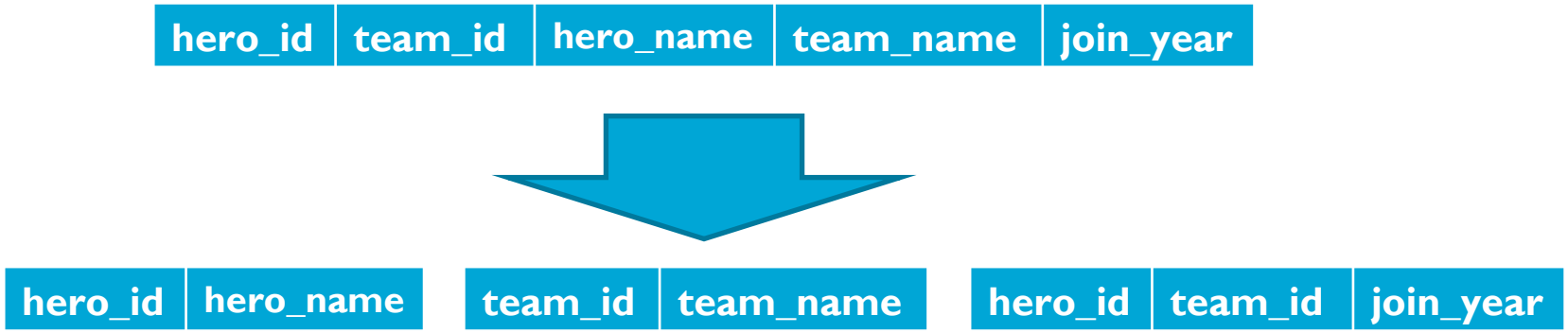
- It’s all about **splitting up** tables ... Normalization
  - remember design B

Towards 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. *hero_id is unique* and *hero_id determines hero_name*)
  - a **relational database instance** (or extension) is
    - a set of tuples adhering to the respective schemas and respecting all integrity constraints

Towards Normalization

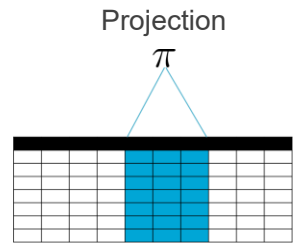
- For this lecture, let's assume the following
  - $S(R_1, ..., R_n)$ is a **relation schema**
  - $R(A_1, \ldots, A_n)$ is a **relation** in S
  - $\mathcal{C}$ is a set of **constraints** satisfied by all extensions of S
- Our ultimate goal is to enhance the database design by **decomposing** the relations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



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:

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

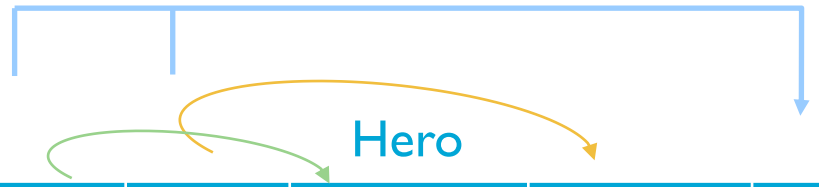


  - $\alpha_1, ..., \alpha_k$ is called a **decomposition** of R

- *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 $\mathcal{C}$

Normalization

$\mathcal{C}$ = {"{hero_id, team_id} is unique",
"hero_id determines hero_name",
"team_id determines team_name",
"{hero_id, team_id} determines join_year"}



hero_id	team_id	hero_name	team_name	join_year
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)

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

π_{α_2}(Hero)

team_id	team_name
1	The Avengers
2	Fantastic Four

π_{α_3}(Hero)

hero_id	team_id	join_year
1	1	1963
2	2	1961
3	1	1963
4	1	1963
5	1	1964
6	2	1961

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 $\alpha_i = \{A_1, ..., A_n\}$ 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*?
    - Functional Dependencies

a_1 = {hero_id, team_id, hero_name, team_name, join_year}

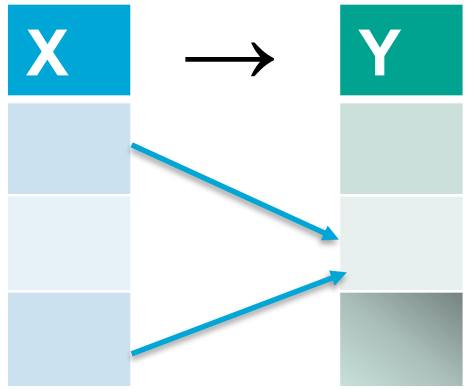
$a_1 = \{\emptyset\}$ Trivial decomposition

hero_id	team_id	hero_name	team_name	join_year
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963

Functional Dependencies

- *hero_id* defines a *hero_name*.
- if I know someone's hero_id, then I can find their hero_name.
- we will say that we have defined *hero_name* as being functionally dependent on *hero_id*.

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*
    - „the values of Y follow from the values of X“
- Examples
  - *{end_time} functionally depends on {start_time, duration}*
  - *{duration} functionally depends on {start_time, end_time}*
  - *{end_time} functionally depends on {end_time}*

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:

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

(This means: if for two given tuples the attributes in X have the same value, then also the attributes Y need to have the same value.)

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 $X \rightarrow Y$

Dependent
Y

Determinant
X

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*

Functional Dependencies

- Functional dependencies are semantic properties of the underlying domain and data m
  -
- F nce (e
  -
  - FDs are **manually defined** integrity constraints on *S*
  - all extensions respecting *S*'s functional dependencies are called **legal extensions** of *S*

Again:
- Functional Dependencies are between sets of attributes
- FD are manually defined by the DB designer
- The designer knows which FDs hold by observing the real world and thinking about it really hard….

Functional Dependencies

- Goal: Good DB design, Normalization, Functional Dependencies, Keys
- To show this, we need a short recap of **keys**
  - 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

Superkey

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

- {studentID, name, countryOfBirth, email}
- {studentID, email, name}
- {studentID, name}
- {studentID, email}
- {studentID} (minimal)
- {email} (minimal)
- etc
- { } NOT A SUPERKEY
- {name, countryOfBirth} NOT A SUPERKEY

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*

A **superkey** is a superset of a key

➢ uniqueness is required

Candidate Key

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

- {studentID} CANDIDATE KEY
- {email} CANDIDATE KEY
- {studentID, email} NOT A CANDIDATE KEY
- {email, countryOfBirth} NOT A CANDIDATE KEY

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

Primary Key

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

- {studentID}
- {email}

- In practice, if there is more than one candidate key, we usually choose one and call it the primary key

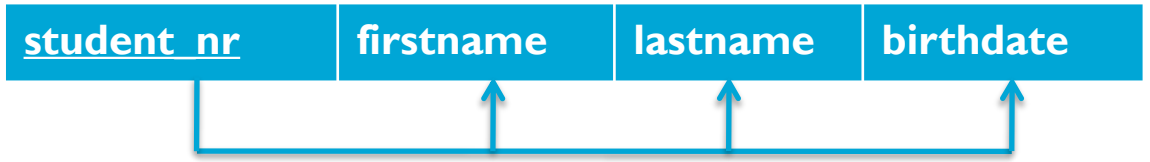
Composite Key

name	phone
Melita	111
Melita	222
John	111

- {name, phone} minimal superkey, candidate key

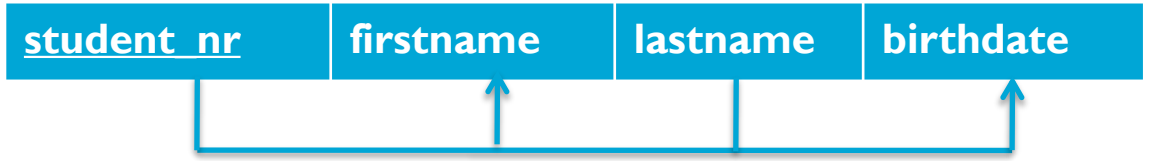
Functional Dependencies

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



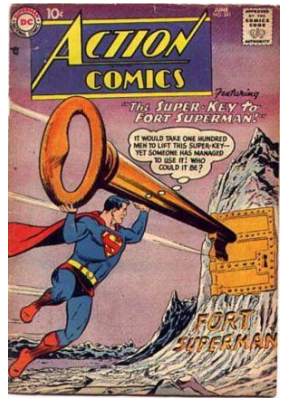
Functional Dependencies

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



Functional Dependencies

- Quick Summary on keys:
  - **Candidate Key** (or simply key)
    - An **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
      - „Identifier“
  - **Superkey** is a superset of a candidate key
    - i.e. only uniqueness is required
      - Superkey also identifies a tuple, but can be reducible
  - **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



DISCUSS WITH YOUR NEIGHBOR!

- What do you derive from knowing….
  - (“you”: actually you, an organization, people with access to more information, etc)
  - The birthday of a person?
  - The public IP address of a computer?
  - A phone number?
  - The birthplace of a person?
  - A TU Delft student number?
  - The model of a car?

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 $X \rightarrow Y$

- $S \rightarrow NC$
  - $S \rightarrow N$
  - $S \rightarrow C$

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

- Fortunately, the closure of *F* can easily be calculated using a small set of **inference rules**

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

Functional Dependencies

- **reflexivity:**
  If $X \supseteq Y$, then $X \rightarrow Y$

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

Example:

- S→S
- SNC→C

➢ If you know *studentID*, you trivially know *studentID*.

➢ If you know *studentID*, name, and *countryOfBirth*, you trivially know *countryOfBirth*.

Functional Dependencies

- **augmentation:**
  If $X \rightarrow Y$, then $X \cup Z \rightarrow Y \cup Z$
  If $X \rightarrow Y$, then $XZ \rightarrow YZ$

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

Example:

•Given S→N, we can infer SC→NC.

➢ Knowing *studentID* determines *name*, adding *countryOfBirth* to both sides preserves the dependency.

Functional Dependencies

- **transitivity:**
  If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

Example:

•Given S→N and N→E, we can infer S→E.

➢ If *studentID* determines *name*, and *name* determines *email*, then *studentID* determines *email*.

Functional Dependencies

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

Functional Dependencies

– **union:**
If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow Y \cup Z$

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

Example:

•Given S→N and S→C, we can infer S→NC.

➢ *studentID* determines both *name* and *countryOfBirth*. Therefore, *studentID* determines *name* and *countryOfBirth*.

Functional Dependencies

- **decomposition:**
  If $X \rightarrow Y \cup Z$, then $X \rightarrow Y$ and $X \rightarrow Z$

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

Example:

•Given S→NE, we can infer S→N and S→E.

➢ If *studentID* determines both *name* and *email*, it also determines each individually.

Functional Dependencies

– **pseudotransitivity:**
If $X \rightarrow Y$ and $Y \cup W \rightarrow Z$, then $X \cup W \rightarrow Z$

studentID	name	countryOfBirth	email
1	Melita	Ireland	melita@email.com
2	John	Greece	john@email.nl

Example:
•Given S→N and NC→E, we can infer SC→E.

➢ *studentID* determines *name*, and *countryOfBirth* and *name* determine *email*. Therefore, *countryOfBirth* and *studentID* together determine *email*.

Functional Dependencies

- Example
  - relational schema R(A, B, C, D, E, F)
  - FDs: $\{A\} \rightarrow \{B, C\}$ $\{B\} \rightarrow \{E\}$ $\{C, D\} \rightarrow \{E, F\}$

  What derivation can you make?

- **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 $X \rightarrow Z$
- **union:**
  If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow Y \cup Z$
- **decomposition:**
  If $X \rightarrow Y \cup Z$, then $X \rightarrow Y$ and $X \rightarrow Z$
- **pseudotransitivity:**
  If $X \rightarrow Y$ and $Y \cup W \rightarrow Z$, then $X \cup W \rightarrow Z$

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)

Functional Dependencies

- In principle, we can calculate 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 calculate the full closure
  - instead, it often suffices to calculate 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**

Functional Dependencies

- Goal of algorithm:
  - Given a set of attributes X, and functional dependencies F, what is the closure (i.e., the set of attributes which are determined by X given F)
  - Idea:
    - Recursively expand the closure, Initialize Closure=X
    - Check each functional dependency if their determinant is currently reachable, i.e., if the determinant is in X
    - If it is, use dependency and add dependent to closure
    - Repeat until you cannot repeat anymore

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

Functional Dependencies

- **Quiz**
  - *R(S,N,C,E)*
  - *F* = { {**S**} → {N}, {N} → {E}, {C} → {S} }
  - *What is the closure of {S} under F?*

Closure of S (S^+)

Start with $S^+ = \{S\}$.

- From S ⟶ N, add N to S^+. Now, $S^+ = \{S, N\}$.
- From N ⟶ E, add E to S^+. Now, $S^+ = \{S, N, E\}$.
- No other FDs are applicable.

Result: $S^+ = \{S, N, E\}$.

Functional Dependencies

- **Quiz**
  - *R(S,N,C,E)*
  - *F* = { {S} → {N}, {**N**} → {E}, {C} → {S} }
  - *What is the closure of {N} under F?*

Closure of N (N^+)

Start with N^+ = {N}.

- From N ⟶ E, add E to N^+. Now, N^+ = {N, E}.
- No other FDs are applicable.

Result: N^+ = {N, E}.

Functional Dependencies

- **Quiz**
  - *R(S,N,C,E)*
  - *F* = { {S} → {N}, {N} → {E}, {**C**} → {S} }
  - *What is the closure of {C} under F?*

Closure of C (C^+)

Start with $C^+ = \{C\}$.

- From C ⟶ S, add S to C^+. Now, $C^+ = \{C, S\}$.
- From S ⟶ N, add N to C^+. Now, $C^+ = \{C, S, N\}$.
- From N ⟶ E, add E to C^+. Now, $C^+ = \{C, S, N, E\}$.
- No other FDs are applicable.

Result: $C^+ = \{C, S, N, E\}$.

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?*

```
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
```

Functional Dependencies

- **Quiz**
  - $F = \{ \{A\} \rightarrow \{B, C\}, \{B\} \rightarrow \{E\}, \{E\} \rightarrow \{C, G\}, \{C, D\} \rightarrow \{E, G\} \}$
  - *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 G, because $\{E\} \rightarrow \{C, G\}$

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

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?**
    - Small: |G| minimal
  - Reduce redundancy and ensure efficiency

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 holds $(X, F)^+ = (X, G)^+$
  - 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

Functional Dependencies

- **Example**
  - F = { {A} → {B}, {A} → {C} }
  - G = { {A} → {B, C} }
  - are F and G equivalent?

  - we must check $(X, F)^+ = (X, G)^+$ for the following X
    - {A}
  - $(\{A\}, F)^+ = \{A, B, C\}$ $(\{A\}, G)^+ = \{A, B, C\}$

  - therefore, F and G are equivalent!

Functional Dependencies

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

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

  - therefore, F and G are not equivalent!

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^+

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^+).

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)

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*:
  $\{A\} \rightarrow \{B\}$, $\{B\} \rightarrow \{C\}$, $\{A\} \rightarrow \{D\}$

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

FDs:
{hero_id} → {hero_name}
{team_id} → {team_name}
{hero_id, team_id} → {join_year}

hero_id	team_id	hero_name	team_name	join_year

Decompose with respect to
{hero_id} → {hero_name}

hero_id	hero_name

hero_id	team_id	team_name	join_year

Functional Dependencies

- 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!

Functional Dependencies

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?

Functional Dependencies

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

Functional Dependencies

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}

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

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

Functional Dependencies

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

- Which of the following FDs are satisfied in this particular extension?
  a) {C} → {A, B}
  b) {A, D} → {C}

Functional Dependencies - Quizz

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

- Which of the following FDs are satisfied in this particular extension?

a) {C} → {A, B} : Yes

b) {A, D} → {C} : No

Functional Dependencies

- Quiz (at home):

– **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} and
  Y is an **element** of {A, B, C, D}

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

Functional Dependencies

- Why care about functional dependencies?
  - They are a tool for helping us to think about the quality of a relational schema design
    - REMEMBER the postcode/address example
  - We need FDs to find good keys
  - We need FDs for normalization
    - i.e. better schemas

Summary: Functional Dependencies

- Functional dependencies (FD)
  - If $\pi_X t_1 = \pi_X t_2$, then $\pi_Y t_1 = \pi_Y t_2$
- Armstrong's axioms and additional rules
  - **reflexivity:** If $Y \supseteq X$, 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 $X \rightarrow Z$
  - **union:** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow Y \cup Z$
  - **decomposition:** If $X \rightarrow Y \cup Z$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - **pseudotransitivity:** If $X \rightarrow Y$ and $Y \cup W \rightarrow Z$, then $X \cup W \rightarrow Z$
- Functional dependencies are the perfect tool for performing lossless decompositions

Normal Forms

TU Delft WIS Web Information Systems

Normalization

Purpose of Normalization

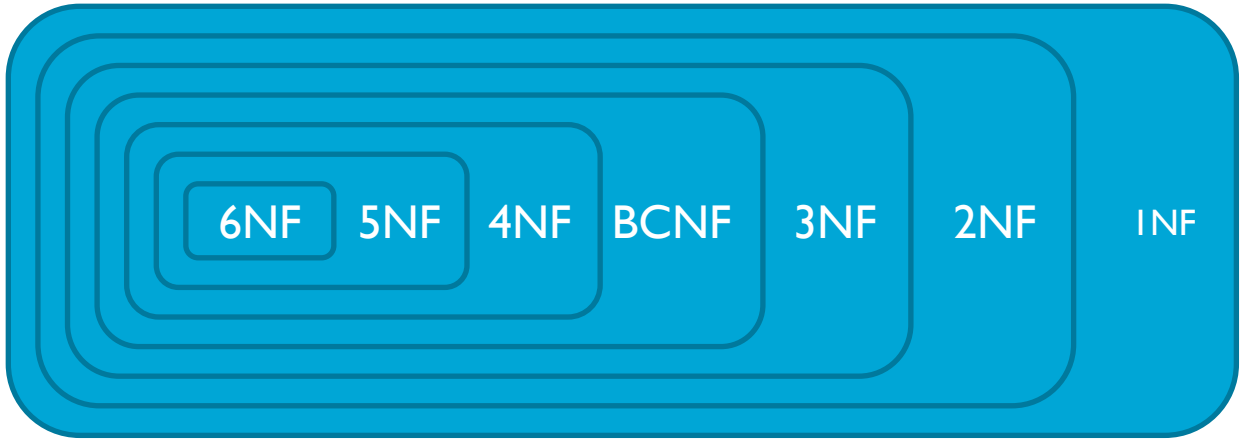
- Eliminate Data Redundancy
- Prevent Modification Anomalies

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

Normal Forms

- Part of a schema design process is to choose a desired normal form and convert the schema into that form
- There are **several normal forms**
  - the higher the number, ...
    - ... the stricter the requirements,
    - ... the less anomalies and redundancy, and
    - ... the better the *design quality*.
    - … increasing complexity in updates and queries.



1NF

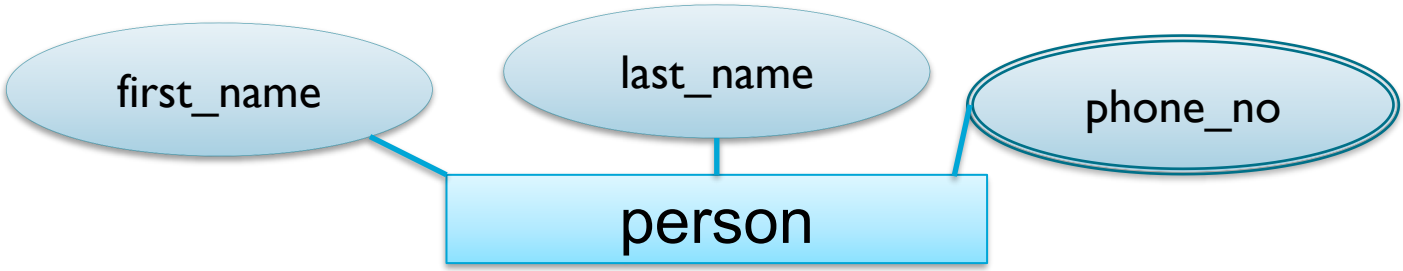
- **First normal form (1NF)**
  - has nothing to do with functional dependencies!
  - restricts relations to being *flat*
  - the value of any attribute must be **atomic,** that is, it **cannot be composed** of several other attributes
  - The table is a relation and each row is unique
    - if these properties are met, the relation is often referred to as a being in **first normal form** (1NF or minimal form)
    - in particular, **set-valued** and relation-valued attributes (tables within tables) are **prohibited**
    - **The table is a relation,**

1NF

studentID	name	countryOfBirth	phone_numbers	name	hobbies
1	Melita	Ireland	00316000001 00316000002	Mel	Tennis, Windsurfing
2	John	Greece	00306900005 00306900008	Johnny	[Football, Beach Volley]

- Not in 1NF
  - ➢ Not an RDB relation

1NF



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

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

1NF

- 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)

1NF

- a) Introducing a **new relation**

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

hero_id	hero_name
1	Storm
2	Wolverine
3	Phoenix

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

1NF

- b) **Replicating** the tuple for each multi-value

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

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

1NF

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

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

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

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 (or prime) attribute** if and only if it is contained in some candidate key; otherwise, it is a **non-key (-prime) attribute**
  - **definition (2NF):**
    A relation schema is in **2NF** iff ...
    - it is in 1NF and
    - **every non-key attribute is functionally dependent on the whole candidate key (no partial dependency on part of a composite key).**

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 **composite candidate key** 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

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 $\alpha_2 = \{A_1, ..., A_n\} \setminus Y$
  - according to Heath's Theorem, this decomposition is **lossless**

FDs:
{hero_id} → {hero_name}
{team_id} → {team_name}
{hero_id, team_id} → {join_year}

Decompose with respect to
{hero_id} → {hero_name}

2NF

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

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

FDs:
`{team_id} → {team_name}`
`{hero_id, team_id} → {join_year}`

Decompose with respect to
`{team_id} → {team_name}`

- 2NF is violated, if there is a **composite key** and some **non-key attribute** depends only on a **proper subset** of this composite key

2NF

hero_id	team_id	hero_name	team_name	join_year

hero_id	hero_name

team_id	team_name

hero_id	team_id	join_year

3NF

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

hero_id	hero_name	home_city_id	home_city_name
11	Professor X	563	New York
12	Wolverine	782	Alberta
13	Cyclops	112	Anchorage
14	Phoenix	563	New York

{hero_id} → {hero_name}
{hero_id} → {home_city_id}
{home_city_id} → {home_city_name}

Not in 3NF

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**
  - {hero_id} → {hero_name}
  - {hero_id} → {home_city_id}
  - {hero_id} → {home_city_name}
  - {home_city_id} → {home_city_name}

hero_id	hero_name	home_city_id	home_city_name
11	Professor X	563	New York
12	Wolverine	782	Alberta
13	Cyclops	112	Anchorage
14	Phoenix	563	New York

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**

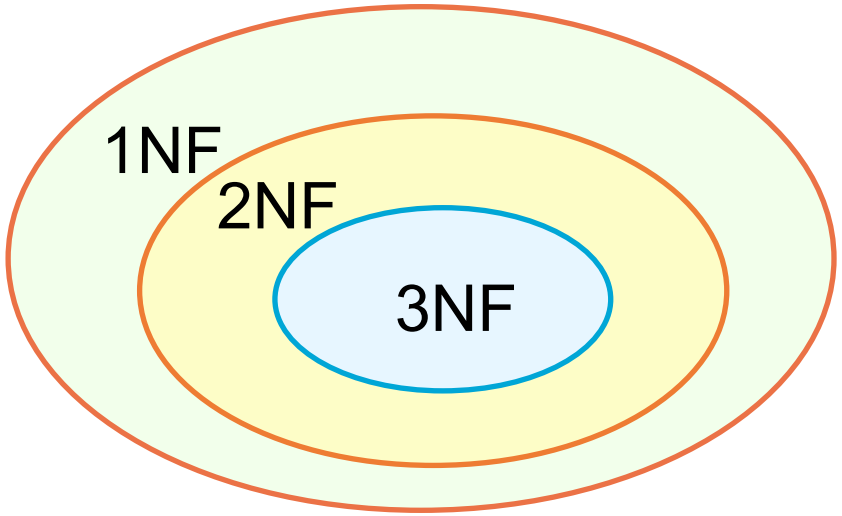
FDs:
{hero_id} → {hero_name}
{hero_id} → {home_city_id}
{home_city_id} → {home_city_name}

Decompose with respect to
{home_city_id} → {home_city_name}

Normal Forms

- Typically, we want database schemas in 3rd Normal Form
  - Or BCNF (not shown!), but in most cases 3NF->BCNF
  - 3rd Normal Form: ~ ”All non-key attributes are only determined by the key”
- There are some rare cases in which you want higher or lower normal forms
  - Know what you are doing when you choose something like that!

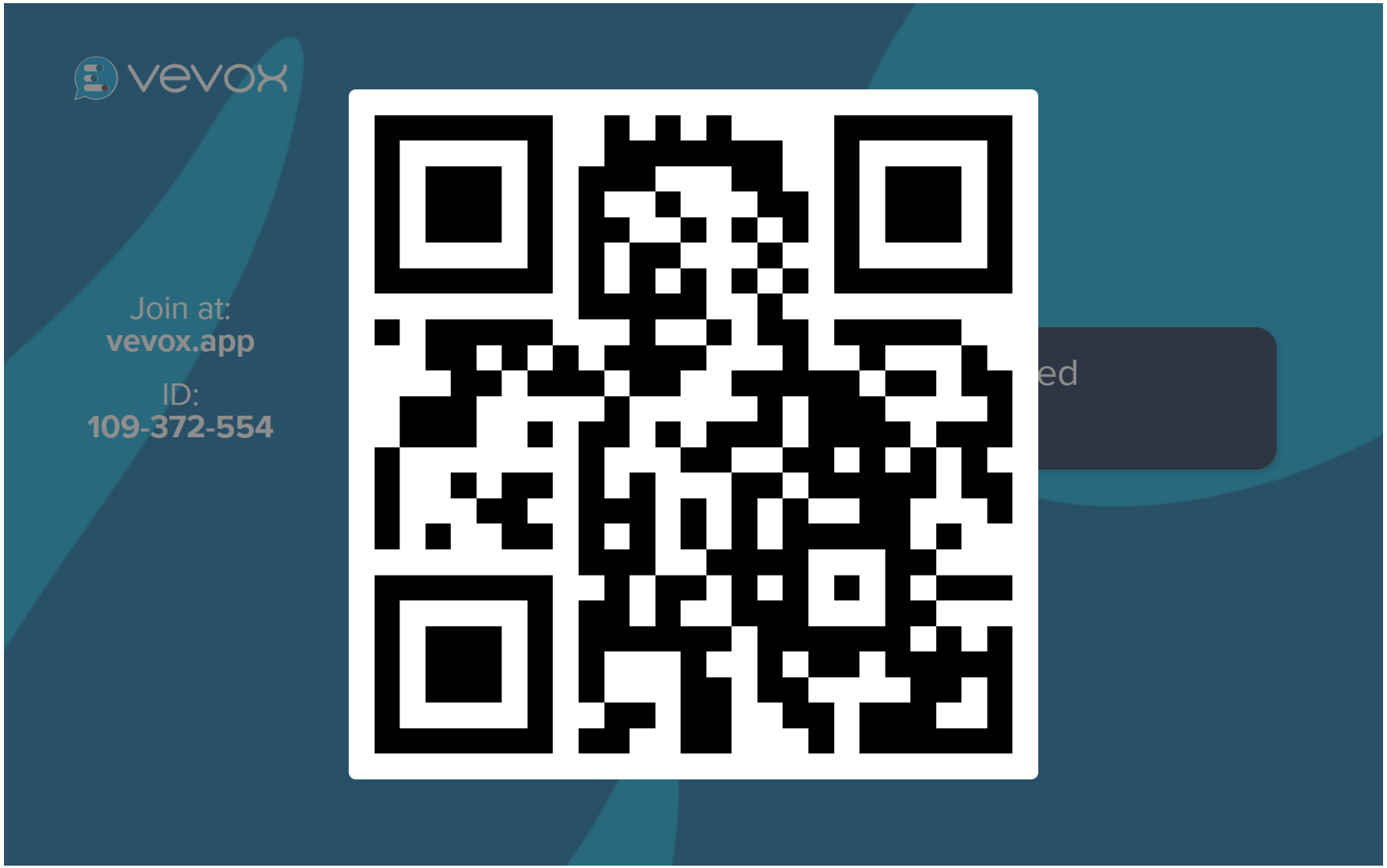
Summary: Normal forms



- Every 2NF/3NF relation is also in 1NF
- Every 3NF relation is also in 2NF

Normal Form	Criteria
1NF	Relation should not have: (1) non-atomic attributes; (2) or nested relations.
2NF	When PK contains multiple attributes: no nonkey attributes functionally dependent on a part of the PK.
3NF	Relation should not have: transitive dependency of a non-key attributes on the PK.

Quiz:



Databases & Software Engineering

See you next time!

[NOT in the Exam] Extended Reading

- **Functional dependencies**
  - Huhtala, Y., Kärkkäinen, J., Porkka, P., & Toivonen, H. (1999). TANE: An efficient algorithm for discovering functional and approximate dependencies. The Computer Journal, 42(2), 100–111.
  - Kruse, S., & Naumann, F. (2018). Efficient discovery of approximate dependencies. Proceedings of the VLDB Endowment (PVLDB), 11(7), 759–772.
  - Khamis, M. A., Ngo, H. Q., Nguyen, X., Olteanu, D., & Schleich, M. (2020). Learning models over relational data using sparse tensors and functional dependencies. ACM Transactions on Database Systems, 45(2).
- **Normalization**
  - Chen, L., Kumar, A., Naughton, J., & Patel, J. M. (2017). Towards linear algebra over normalized data. Proceedings of the VLDB Endowment, 10(11), 1214–1225.

r.hai@tudelft.nl

Extra slides

These slides are extra with alternative descriptions / examples. Just to state that again: In the assignments / Exams, we will only do 1NF-3NF; no BCNF or >4NF

Functional Dependencies

TU Delft WIS Web Information Systems

Functional Dependencies

- Functional dependencies (FDs)
  - Are used to specify *formal measures* of the "goodness" of relational designs
  - And keys are used to define **normal forms** for relations
  - Are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes
- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y

Defining Functional Dependencies

- X → Y holds if whenever two tuples have the same value for X, they must have the same value for Y
  - For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], then t1[Y]=t2[Y]
- X → Y in R specifies a constraint on all relation instances r(R)
- Written as X → Y; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

Examples of FD constraints (1)

- Social security number determines employee name
  - SSN → ENAME
- Project number determines project name and location
  - PNUMBER → {PNAME, PLOCATION}
- Employee ssn and project number determines the hours per week that the employee works on the project
  - {SSN, PNUMBER} → HOURS

Examples of FD constraints (2)

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every* relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R
  - (since we never have two distinct tuples with t1[K]=t2[K])

Defining FDs from instances

- Note that in order to define the FDs, we need to understand the meaning of the attributes involved and the relationship between them.
- An FD is a property of the attributes in the schema R
- Given the instance (population) of a relation, all we can conclude is that an FD *may exist* between certain attributes.
- What we can definitely conclude is – that certain FDs *do not exist* because there are tuples that show a violation of those dependencies.

Figure 14.7 Ruling Out FDs

Note that given the state of the TEACH relation, we can say that the FD: Text → Course may exist. However, the FDs Teacher → Course, Teacher → Text and Course → Text are ruled out.

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Figure 14.8 What FDs may exist?

- A relation *R*(A, B, C, D) with its extension.
- Which FDs *may exist* in this relation?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

Normal Forms Based on Primary Keys

- 3.1 Normalization of Relations
- 3.2 Practical Use of Normal Forms
- 3.3 Definitions of Keys and Attributes Participating in Keys
- 3.4 First Normal Form
- 3.5 Second Normal Form
- 3.6 Third Normal Form

Normalization of Relations (1)

- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

- **Normal form:**
  - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

Normalization of Relations (2)

- 2NF, 3NF, BCNF
  - based on keys and FDs of a relation schema
- 4NF
  - based on keys, multi-valued dependencies : MVDs;
- 5NF
  - based on keys, join dependencies : JDs
- Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation; see Chapter 15)

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties

- The practical utility of these normal forms becomes questionable when the constraints on which they are based are *hard to understand* or to *detect*
- The database designers *need not* normalize to the highest possible normal form
  - (usually up to 3NF and BCNF. 4NF rarely used in practice.)
- **Denormalization**:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

3.3 Definitions of Keys and Attributes Participating in Keys (1)

- A **superkey** of a relation schema R = {A1, A2, ..., An} is a set of attributes S *subset-of* R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]

- A **key** K is a **superkey** with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Definitions of Keys and Attributes Participating in Keys (2)

- If a relation schema has more than one key, each is called a **candidate** key.
  - One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **secondary keys**.
- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

3.4 First Normal Form

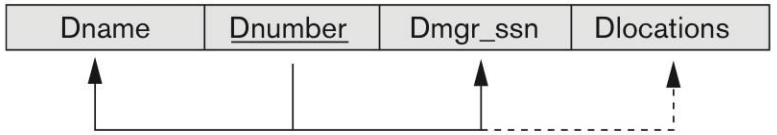
- Disallows
  - composite attributes
  - multivalued attributes
  - nested relations; attributes whose values for an *individual tuple* are non-atomic
- Considered to be part of the definition of a relation
- Most RDBMSs allow only those relations to be defined that are in First Normal Form

Figure 14.9 Normalization into 1NF

- 1NF Disallows
  - composite attributes
  - **multivalued attributes**
  - nested relations; attributes whose values for an *individual tuple* are non-atomic

(a)

DEPARTMENT



(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 14.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Figure 14.10 Normalizing nested relations into 1NF

- 1NF Disallows
  - composite attributes
  - multivalued attributes
  - **nested relations**; attributes whose values for an *individual tuple* are non-atomic

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 14.10
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a nested relation attribute PROJS. (b) Sample extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

3.5 Second Normal Form (1)

- Uses the concepts of **FDs, primary key**
- Definitions
  - **Prime attribute:** An attribute that is member of the primary key K
  - **Full functional dependency:** a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more
- Examples:
  - {SSN, PNUMBER} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold
  - {SSN, PNUMBER} -> ENAME is not a full FD (it is called a partial dependency) since SSN -> ENAME also holds

Second Normal Form (2)

- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key

- R can be decomposed into 2NF relations via the process of 2NF normalization or “second normalization”

Figure 14.11 Normalizing into 2NF and 3NF

2NF: A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key

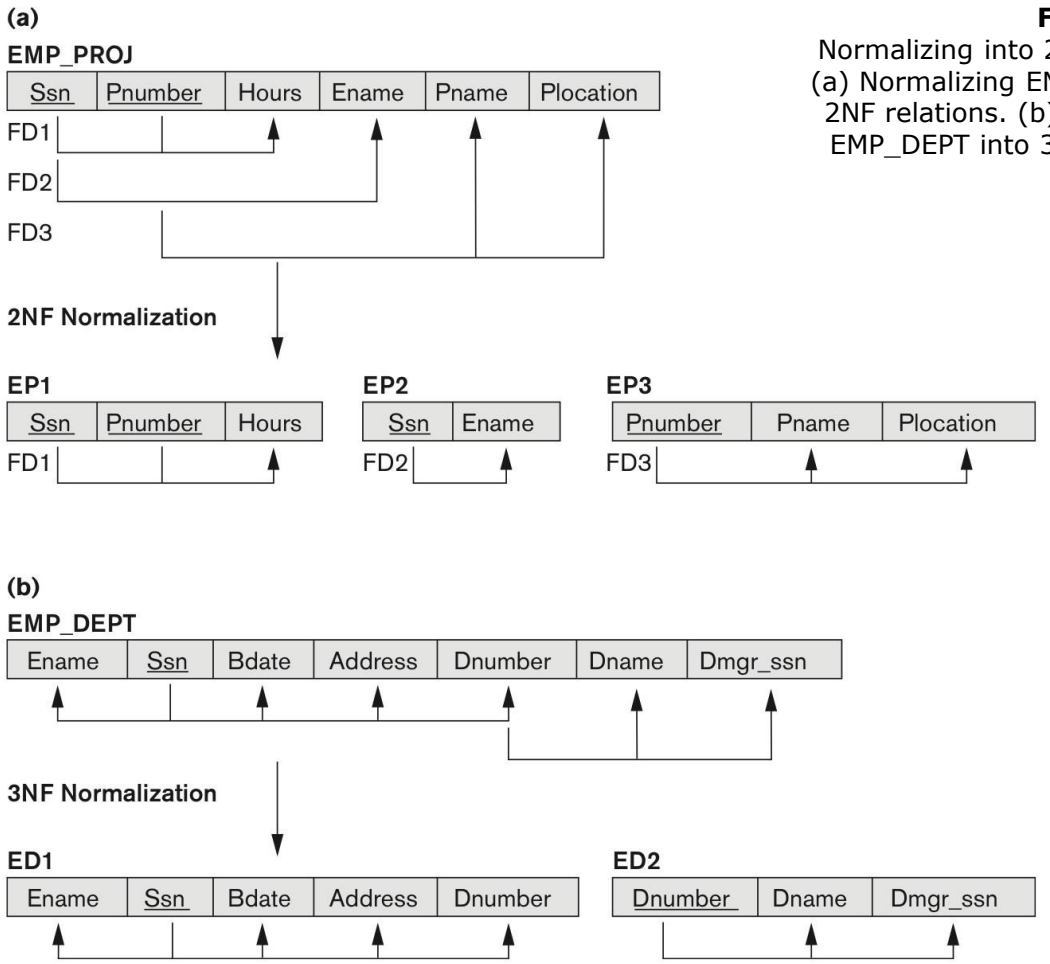
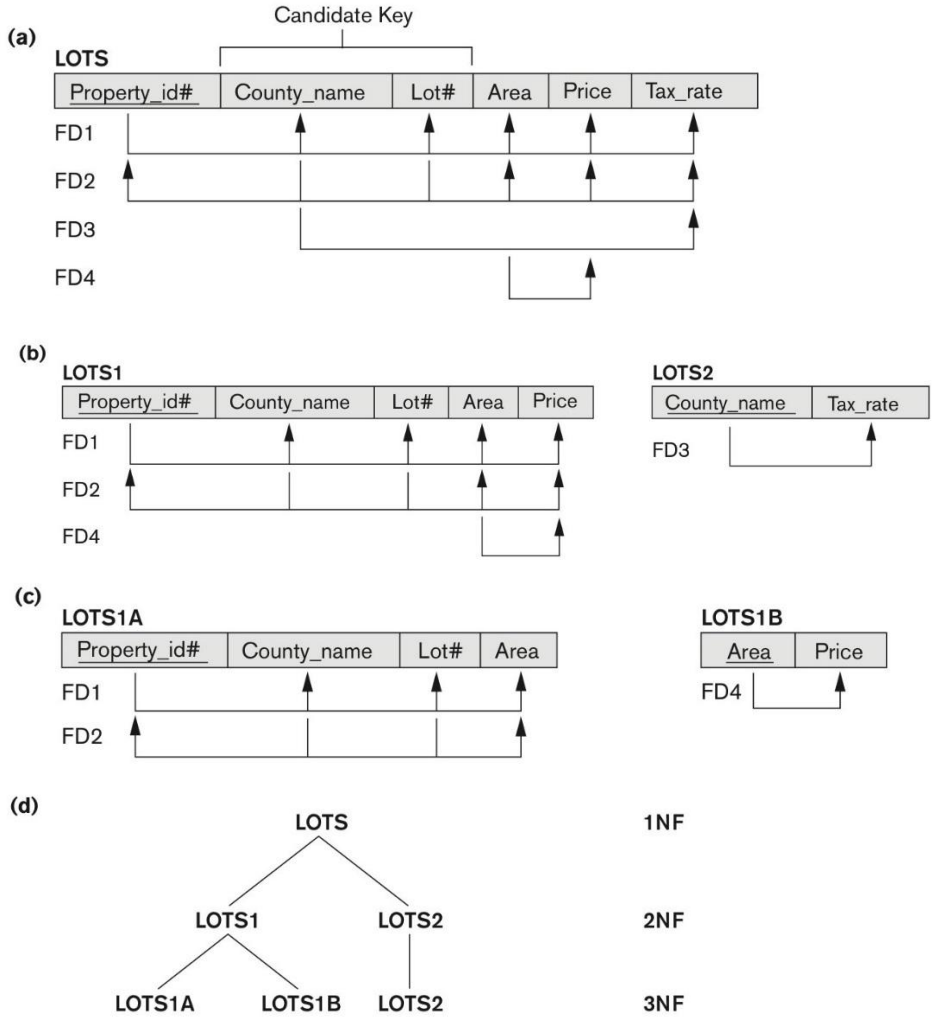


Figure 14.11
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Figure 14.12 Normalization into 2NF and 3NF

Figure 14.12
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Progressive normalization of LOTS into a 3NF design.



3.6 Third Normal Form (1)

- Definition:
  - **Transitive functional dependency:** a FD X -> Z that can be derived from two FDs X -> Y and Y -> Z
- Examples:
  - SSN -> DMGRSSN is a **transitive** FD
    - Since SSN -> DNUMBER and DNUMBER -> DMGRSSN hold
  - SSN -> ENAME is **non-transitive**
    - Since there is no set of attributes X where SSN -> X and X -> ENAME

Third Normal Form (2)

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key
- R can be decomposed into 3NF relations via the process of 3NF normalization
- NOTE:
  - In X -> Y and Y -> Z, with X as the primary key, we consider this a problem only if Y is not a candidate key.
  - When Y is a candidate key, there is no problem with the transitive dependency .
  - E.g., Consider EMP (SSN, Emp#, Salary).
    - Here, SSN -> Emp# -> Salary and Emp# is a candidate key.

Normal Forms Defined Informally

- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

General Normal Form Definitions (For Multiple Keys) (1)

- The above definitions consider the primary key only
- The following more general definitions take into account relations with multiple candidate keys
- Any attribute involved in a candidate key is a prime attribute
- All other attributes are called non-prime attributes.

General Definition of 2NF (For Multiple Candidate Keys)

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on every key of R
- In Figure 14.12 the FD
- County_name → Tax_rate violates 2NF.

- So second normalization converts LOTS into
- LOTS1 (Property_id#, County_name, Lot#, Area, Price)
- LOTS2 (County_name, Tax_rate)

General Definition of Third Normal Form

- Definition:
  - Superkey of relation schema R - a set of attributes S of R that contains a key of R
  - A relation schema R is in third normal form (3NF) if whenever a FD X → A holds in R, then either:
    - (a) X is a superkey of R, or
    - (b) A is a prime attribute of R
- LOTS1 relation violates 3NF because
- Area → Price ; and Area is not a superkey in LOTS1. (see Figure 14.12).

Interpreting the General Definition of Third Normal Form

- Consider the 2 conditions in the Definition of 3NF:
  - A relation schema R is in third normal form (3NF) if whenever a FD X → A holds in R, then either:
    - (a) X is a superkey of R, or
    - (b) A is a prime attribute of R
- Condition (a) catches two types of violations :
- - one where a prime attribute functionally determines a non-prime attribute. This catches 2NF violations due to non-full functional dependencies.
- -second, where a non-prime attribute functionally determines a non-prime attribute. This catches 3NF violations due to a transitive dependency.

Interpreting the General Definition of Third Normal Form (2)

- **ALTERNATIVE DEFINITION of 3NF: We can restate the definition as:**

  A relation schema R is in **third normal form (3NF)** if every non-prime attribute in R meets both of these conditions:

  - It is fully functionally dependent on every key of R
  - It is non-transitively dependent on every key of R

  Note that stated this way, a relation in 3NF also meets the requirements for 2NF.

- The condition (b) from the last slide takes care of the dependencies that "slip through" (are allowable to) 3NF but are "caught by" BCNF which we discuss next.

BCNF (Boyce-Codd Normal Form)

- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an **FD** **X → A** holds in R, then **X is a superkey** of R
- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF
- Hence BCNF is considered a stronger form of 3NF
- The goal is to have each relation in BCNF (or 3NF)

Figure 14.13 Boyce-Codd normal form

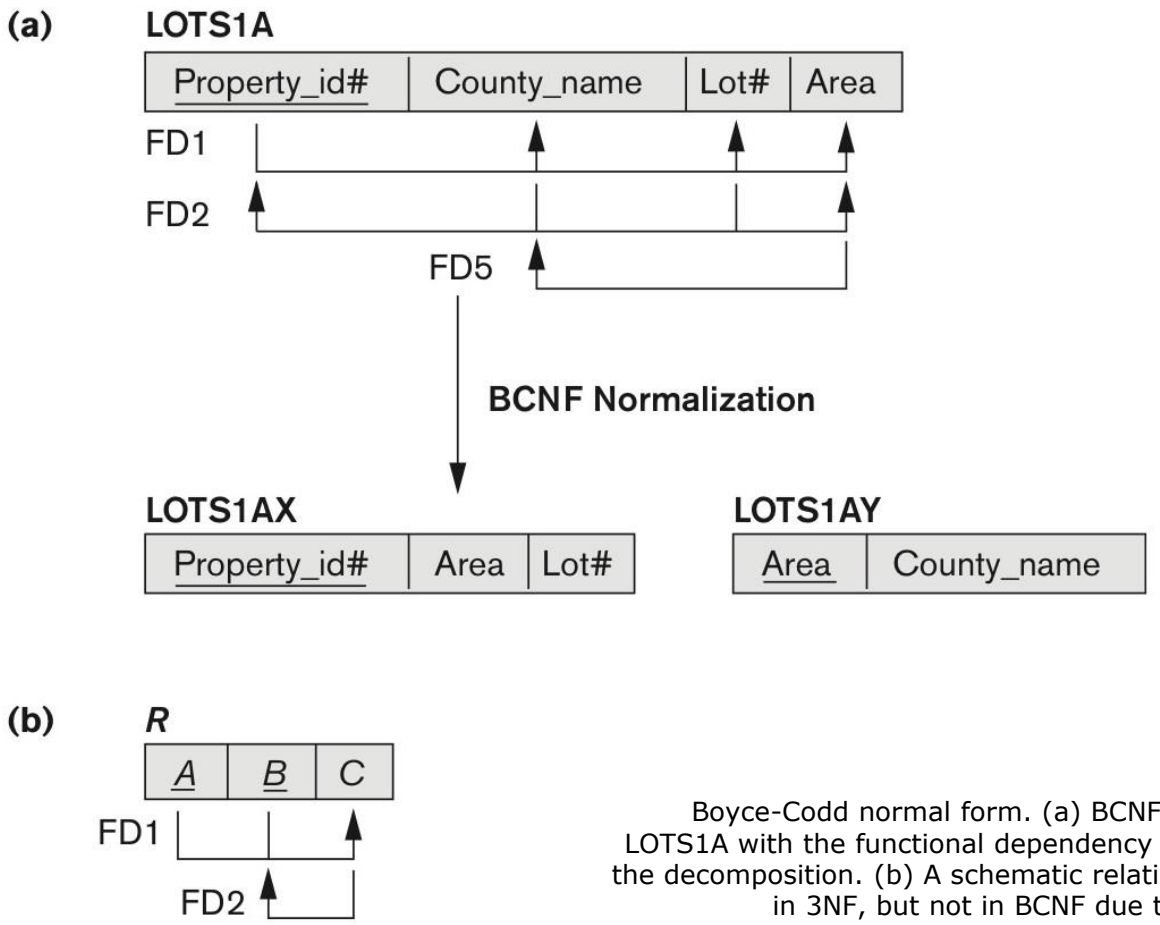


Figure 14.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. C → B.

Figure 14.14 A relation TEACH that is in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 14.14
A relation TEACH that is in 3NF but not BCNF.

Achieving the BCNF by Decomposition (1)

- Two FDs exist in the relation TEACH:
  - fd1: { student, course} -> instructor
  - fd2: instructor -> course
- {student, course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 14.13 (b).
  - So this relation is in 3NF but not in BCNF
- A relation NOT in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.
  - (See Algorithm 15.3)

Achieving the BCNF by Decomposition (2)

- Three possible decompositions for relation TEACH
  - D1: {student, instructor} and {student, course}
  - D2: {course, instructor } and {course, student}
  - D3: {instructor, course } and {instructor, student} ✓
- All three decompositions will lose fd1.
  - We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.
- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).

- A test to determine whether a binary decomposition (decomposition into two relations) is non-additive (lossless) is discussed under Property NJB on the next slide. We then show how the third decomposition above meets the property.

Test for checking non-additivity of Binary Relational Decompositions

- Testing Binary Decompositions for Lossless Join (Non-additive Join) Property
  - Binary Decomposition: Decomposition of a relation R into two relations.
  - PROPERTY NJB (non-additive join test for binary decompositions): A decomposition D = {R1, R2} of R has the lossless join property with respect to a set of functional dependencies F on R if and only if either
    - The f.d. ((R1 ∩ R2) → (R1- R2)) is in F+, or
    - The f.d. ((R1 ∩ R2) → (R2 - R1)) is in F+.

Test for checking non-additivity of Binary Relational Decompositions

- If you apply the NJB test to the 3 decompositions of the TEACH relation:
- D1 gives Student → Instructor or Student → Course, none of which is true.
- D2 gives Course → Instructor or Course → Student, none of which is true.
- However, in D3 we get Instructor → Course or Instructor → Student.
- Since Instructor → Course is indeed true, the NJB property is satisfied and D3 is determined as a non-additive (good) decomposition.

General Procedure for achieving BCNF when a relation fails BCNF

Here we make use the algorithm from Chapter 15 (Algorithm 15.5):

- Let R be the relation not in BCNF, let X be a subset-of R, and let $X \rightarrow A$ be the FD that causes a violation of BCNF. Then R may be decomposed into two relations:
- (i) $R - A$ and (ii) $X \cup A$.
- If either $R - A$ or $X \cup A$. is not in BCNF, repeat the process.

Note that the f.d. that violated BCNF in TEACH was Instructor →Course. Hence its BCNF decomposition would be :

(TEACH – COURSE) and (Instructor ∪ Course), which gives

the relations: (Instructor, Student) and (Instructor, Course) that we obtained before in decomposition D3.

Our last year's lecture on Normal Forms

TU Delft WIS Web Information Systems

Lecture Outline

- 3 Normal Forms Based on Primary Keys
- 4 General Normal Form Definitions for 2NF and 3NF (For Multiple Candidate Keys)
- 5 BCNF (Boyce-Codd Normal Form)

Informal Design Guidelines for Relational Databases (1)

- What is relational database design?
  - The grouping of attributes to form "good" relation schemas
- Two levels of relation schemas
  - The logical "user view" level
  - The storage "base relation" level
- Design is concerned mainly with base relations
- What are the criteria for "good" base relations?

Informal Design Guidelines for Relational Databases (2)

- We first discuss informal guidelines for good relational design
- Then we discuss formal concepts of functional dependencies and normal forms
  - - 1NF (First Normal Form)
  - - 2NF (Second Normal Form)
  - - 3NF (Third Normal Form)
  - - BCNF (Boyce-Codd Normal Form)
- Additional types of dependencies, further normal forms, relational design algorithms by synthesis are discussed in Chapter 15 of our book.

1.1 Semantics of the Relational Attributes must be clear

- GUIDELINE 1: Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).
  - Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
  - Only foreign keys should be used to refer to other entities
  - Entity and relationship attributes should be kept apart as much as possible.
- Bottom Line: Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.

Figure 14.1 A simplified COMPANY relational database schema

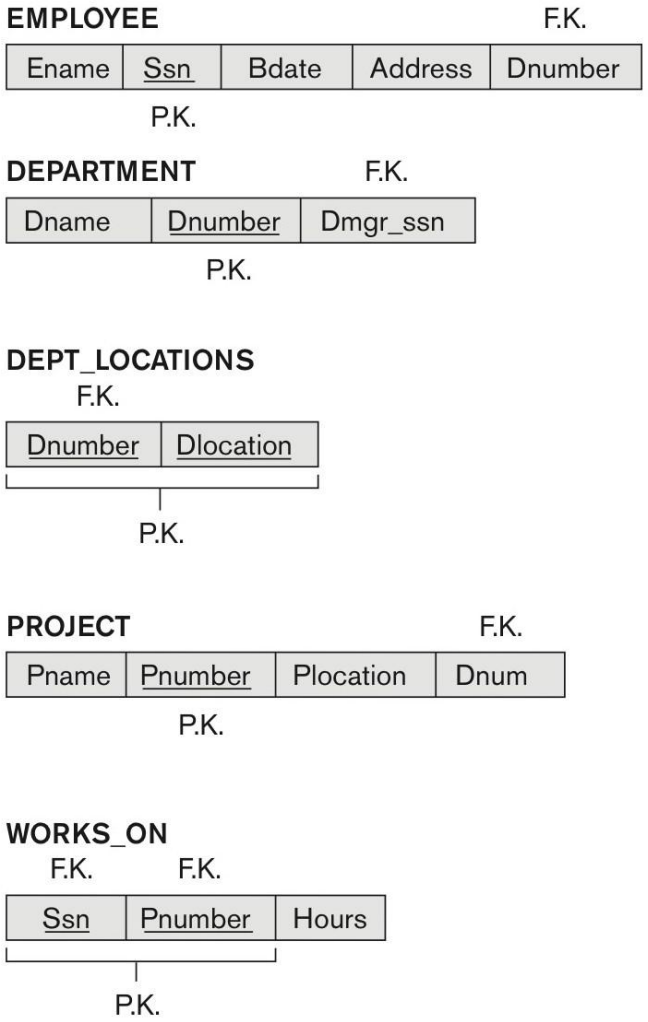


Figure 14.1 A simplified COMPANY relational database schema.

1.2 Redundant Information in Tuples and Update Anomalies

- Information is stored redundantly
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies

EXAMPLE OF AN UPDATE ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Update Anomaly:
  - Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

EXAMPLE OF AN INSERT ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Insert Anomaly:
  - Cannot insert a project unless an employee is assigned to it.
- Conversely
  - Cannot insert an employee unless an he/she is assigned to a project.

EXAMPLE OF A DELETE ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Delete Anomaly:
  - When a project is deleted, it will result in deleting all the employees who work on that project.
  - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

Figure 14.3 Two relation schemas suffering from update anomalies

Figure 14.3
Two relation schemas suffering from update anomalies. (a) EMP_DEPT and (b) EMP_PROJ.

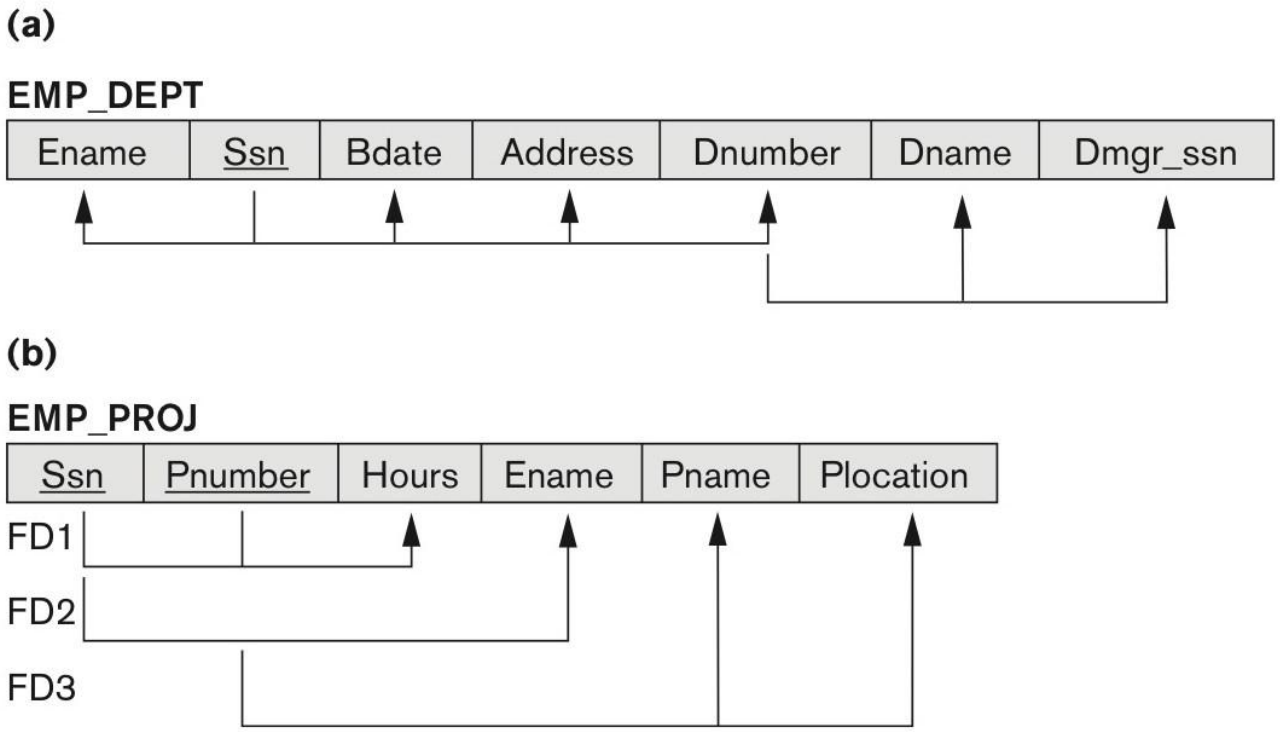


Figure 14.4 Sample states for EMP_DEPT and EMP_PROJ

Figure 14.4
Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 14.2. These may be stored as base relations for performance reasons.

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Guideline for Redundant Information in Tuples and Update Anomalies

- GUIDELINE 2:
  - Design a schema that does not suffer from the insertion, deletion and update anomalies.
  - If there are any anomalies present, then note them so that applications can be made to take them into account.

Null Values in Tuples

- GUIDELINE 3:
  - Relations should be designed such that their tuples will have as few NULL values as possible
  - Attributes that are NULL frequently could be placed in separate relations (with the primary key)
- Reasons for nulls:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable

Generation of Spurious Tuples – avoid at any cost

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

- GUIDELINE 4:
  - The relations should be designed to satisfy the lossless join condition.
  - No spurious tuples should be generated by doing a natural-join of any relations.

Spurious Tuples (2)

- There are two important properties of decompositions:
  a) Non-additive or losslessness of the corresponding join
  b) Preservation of the functional dependencies.

- Note that:
  - Property (a) is extremely important and *cannot* be sacrificed.
  - Property (b) is less stringent and may be sacrificed. (See Chapter 15).

4th Normal Form and Beyond

Optional reading

TU Delft WIS Web Information Systems

Multivalued Dependencies and Fourth Normal Form (1)

- Definition:
- A multivalued dependency (MVD) X —>> Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t1 and t2 exist in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties, where we use Z to denote (R 2 (X ∪ Y)):
  - t3[X] = t4[X] = t1[X] = t2[X].
  - t3[Y] = t1[Y] and t4[Y] = t2[Y].
  - t3[Z] = t2[Z] and t4[Z] = t1[Z].
- An MVD X —>> Y in R is called a trivial MVD if (a) Y is a subset of X, or (b) X ∪ Y = R.

Multivalued Dependencies and Fourth Normal Form (3)

- Definition:
- A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every nontrivial multivalued dependency X —>> Y in F+, X is a superkey for R.
  - Note: F+ is the (complete) set of all dependencies (functional or multivalued) that will hold in every relation state r of R that satisfies F. It is also called the closure of F.

Fourth and fifth normal forms.

(a) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(b) EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

(c) SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d) R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Figure 14.15
Fourth and fifth normal forms. (a) The EMP relation with two MVDs: Ename –>> Pname and Ename –>> Dname. (b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS. (c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R1, R2, R3). (d) Decomposing the relation SUPPLY into the 5NF relations R1, R2, R3.

Join Dependencies and Fifth Normal Form (1)

- Definition:
- A join dependency (JD), denoted by JD(R1, R2, ..., Rn), specified on relation schema R, specifies a constraint on the states r of R.
  - The constraint states that every legal state r of R should have a non-additive join decomposition into R1, R2, ..., Rn; that is, for every such r we have
  - * (□R1(r), □R2(r), ..., □Rn(r)) = r
- Note: an MVD is a special case of a JD where n = 2.
- A join dependency JD(R1, R2, ..., Rn), specified on relation schema R, is a trivial JD if one of the relation schemas Ri in JD(R1, R2, ..., Rn) is equal to R.

Join Dependencies and Fifth Normal Form (2)

Definition:

- A relation schema *R* is in **fifth normal form** (**5NF**) (or **Project-Join Normal Form** (**PJNF**)) with respect to a set *F* of functional, multivalued, and join dependencies if,
  - for every nontrivial join dependency JD(R_1, R_2, ..., R_n) in F^+ (that is, implied by *F*),
    - every R_i is a superkey of *R*.
- Discovering join dependencies in practical databases with hundreds of relations is next to impossible. Therefore, 5NF is rarely used in practice.

BCNF

These are slides Christoph made a while ago on BCNF (we do NOT do BNCF in the exam/assignments of this course). Just in case you want to see them…

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
- A 3NF table that does not have multiple overlapping candidate keys is guaranteed to be in BCNF

BCNF

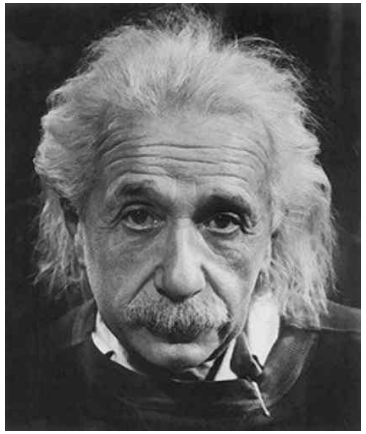
- BCNF is very **similar** to **3NF:**
  - **BCNF:**
    In any non-trivial FD $X \rightarrow Y$, the set X is a superkey.
  - **3NF (alternative definition):**
    In any non-trivial FD $X \rightarrow Y$, the set X is a superkey, or the set Y is a subset of some key.

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

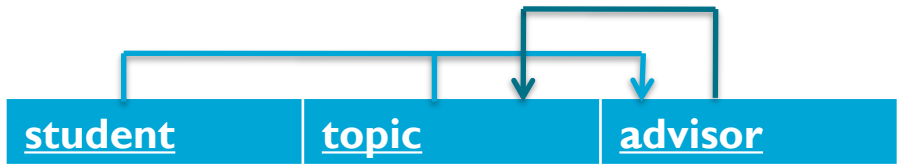
BCNF

- **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



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

160

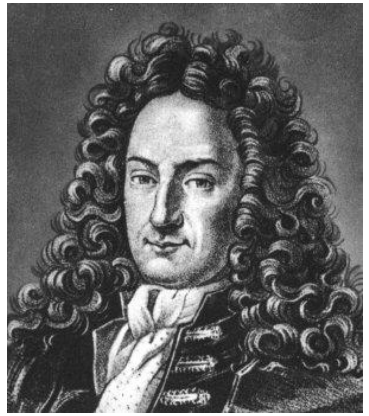
BCNF

- 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

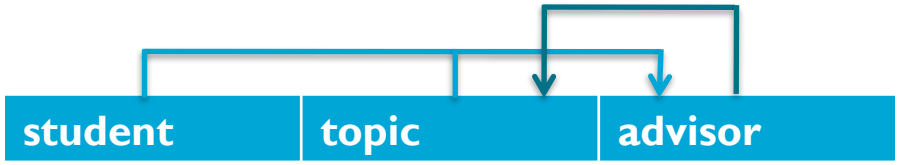
  - Deleting the last student of an advisor?
  - An advisor changes his topic?
    - Because $\{\text{Advisor}\} \rightarrow \{\text{Topic}\}$, multiple updates necessarry



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?
  - $\{\text{Student}, \text{Topic}\} \rightarrow \{\text{Advisor}\}$ is “lost“ in all options

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