# Databases – Database Normalisation

### Jörg Endrullis

VU University Amsterdam

### Database Normalisation :: Introduction

# Introduction

**Functional Dependencies (FDs)** 

- are a generalization of keys
- central part of relational database design theory

This theory defines when a relation is in normal form.

Usually a sign of **bad database design** if a schema contains relations that **violate the normal form**.

If a normal form is violated

- data is stored redundantly and
- information about different concepts is intermixed

Courses				
<u>courseNr</u> title instructor phone				
230	Databases I	Arthur	9002	
415	Functional Programming	Arthur	9002	
301	Graph Theory	Marvin	8020	

The phone number for each instructor is stored multiple times!

There are different normal forms. The main ones are:

- **Third Normal Form (3NF)**: the standard relational normal form used in practice (and education).
- Boyce-Codd Normal Form (BCNF):
  - a bit more restrictive
  - easier to define
  - better for our intuition of good database design

Roughly speaking, BNCF requires that all FDs are keys.

In rare circumstances, a relation might not have an equivalent BCNF form while preserving all its FDs.

The 3NF normal form always exists (and preserves the FDs).

**Normalization algorithms** can construct good relation schemas from a set of attributes and a set of functional dependencies.

In practice:

- relations are derived from entity-relationship models
- normalization is used as an additional check only

When an ER model is **well designed**, the resulting derived relational tables will **automatically be in BCNF**.

Awareness of normal forms can help to detect design errors already in the conceptual design phase.

### Database Normalisation :: First Normal Form

The **First Normal Form (1NF)** requires that all **table entries are atomic** (*not* lists, sets, records, relations).

In the relational model all table entries are already atomic.

All further normal forms assume that tables are in 1NF!

The following are **not violations of 1NF**:

- a table entry contains values with internal structure
  - e.g. a char (100) containing a comma separated list
- a list represented by several columns
  - e.g. columns value1, value2, value3

Nevertheless, these are **bad design**.

### Database Normalisation :: Functional Dependencies

# **Functional Dependencies**

Courses				
<u>courseNr</u>	title	instructor	phone	
230	Databases I	Arthur	9002	
415	Functional Programming	Arthur	9002	
301	Graph Theory	Marvin	8020	

A functional dependency (FD) in this table is

instructor  $\rightarrow$  phone

Whenever two rows agree in the instructor name, they **must** also agree in the phone number!

Intuitively, this is a FD since the phone number **only depends on the instructor**, not on the other attributes.

## **Functional Dependencies**

A functional dependency (FD)  $\{A_1, \dots, A_n\} \rightarrow \{B_1, \dots, B_m\}$ holds for a relation R in a database state I if and only if  $t.A_1 = u.A_1 \land \dots \land t.A_n = u.A_n$  $\Rightarrow t.B_1 = u.B_1 \land \dots \land t.B_m = u.B_m$ 

for all tuples  $t, u \in I(R)$ .

Usually, we do not write the set brackets { and }.

A functional dependency is like a **partial key:** uniquely determines some attributes, but not all in general.

We read functional dependencies as

•  $A_1, \ldots, A_n$  (functionally, uniquely) determine  $B_1, \ldots, B_m$ 

### **Functional Dependencies**

A functional dependency with *m* attributes on the right

$$A_1,\ldots A_n \rightarrow B_1,\ldots B_m$$

is equivalent to the m FDs:

$$egin{array}{ccccc} A_1,\ldots,A_n& o&B_1\ dots&dots&dots\ A_1,\ldots,A_n& o&B_m\ dots&dots&dots\ A_1,\ldots,A_n& o&B_m \end{array}$$

is equivalent to the combination of

$$oldsymbol{A},oldsymbol{B}
ightarrow oldsymbol{C}$$
  $oldsymbol{A},oldsymbol{B}
ightarrow oldsymbol{D}$ 

but not equivalent to

$$A 
ightarrow C, D$$
  $B 
ightarrow C, L$ 

So, in the sequel it suffices to consider functional dependencies with a single column name on the right-hand side.

Database Normalisation :: Keys vs. Functional Dependencies

# Keys are Functional Dependencies

### Keys are functional dependencies

 $\{A_1, \ldots, A_n\}$  is a key of relation  $R(A_1, \ldots, A_n, B_1, \ldots, B_m)$  $\iff$  the functional dependency  $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$  holds.

A key uniquely determines all attributes of its relation.

	Courses		
<u>courseNr</u>	title	instructor	phone
230	Databases I	Arthur	9002
415	Functional Programming	Arthur	9002
301	Graph Theory	Marvin	8020

We have the following functional dependencies:

 $\texttt{courseNr} \rightarrow \texttt{title}, \texttt{instructor}, \texttt{phone}$ 

or equivalently:

 $\begin{array}{c} \text{courseNr} \rightarrow \text{title} \\ \text{courseNr} \rightarrow \text{instructor} \\ \text{courseNr} \rightarrow \text{phone} \end{array}$ 

### Functional Dependencies are Partial Keys

Functional dependencies are partial keys

The functional dependency

$$A_1,\ldots,A_n o B_1,\ldots,B_m$$

holds for a relation *R* if  $\{A_1, \ldots, A_n\}$  is a key for the relation obtained by restricting *R* to columns  $\{A_1, \ldots, A_n, B_1, \ldots, B_m\}$ .

The restriction of the table <code>Courses</code> to { <code>instructor</code>, phone } is:

instructor	phone
Arthur	9002
Marvin	8020

Here instructor is a key. So instructor  $\rightarrow$  phone in Courses.

# The goal of database normalization is to turn FDs into keys.

The DBMS is then able to enforce the FDs for the user.

Database Normalisation :: Functional Dependencies are Constraints

### Functional Dependencies are Constraints

Functional dependencies are **constraints** (like keys).

Courses				
courseNr	title	instructor	phone	
230	Databases I	Arthur	9002	
415	Functional Programming	Arthur	9002	
301	Graph Theory	Marvin	8020	

In this example state, the functional dependency  $\texttt{title} \rightarrow \texttt{courseNr}$ 

holds. But this is probably not true in general!

For the database design, the only interesting functional dependencies are those that **hold for all intended states**.

# Example: Books and Authors

Books					
author	no	title	isbn		
Baader	1	Term Rewriting	Cambridge Uni.	0521779200	
Nipkow	2	Term Rewriting	Cambridge Uni.	0521779200	
Graham	1	Concrete Mathematics	Addison-Wesley	0201558025	
Knuth	2	Concrete Mathematics	Addison-Wesley	0201558025	
Patashnik	3	Concrete Mathematics	Addison-Wesley	0201558025	

A book may have multiple authors, one author per row. The attribute no is used to indicate the order of the authors.

- isbn → title, publisher (ISBN uniquely identifies a book)
- isbn  $\rightarrow$  author ? Does not hold.
- author  $\rightarrow$  title ? Does not hold in general.
- title  $\rightarrow \varnothing$  (There may be books with the same title)
- isbn, no  $\rightarrow$  author
- $i sbn, author \rightarrow no$  ? questionable (e.g. Smith & Smith)
- publisher, title, no → author ? questionable Authorship sequence might change in a new edition of a book!

# Example: Books and Authors

	Books					
	author	no	title	isbn		
Γ	Baader	1	Term Rewriting	Cambridge Uni.	0521779200	
	Nipkow	2	Term Rewriting	Cambridge Uni.	0521779200	
	Graham	1	Concrete Mathematics	Addison-Wesley	0201558025	
	Knuth	2	Concrete Mathematics	Addison-Wesley	0201558025	
	Patashnik	3	Concrete Mathematics	Addison-Wesley	0201558025	

A book may have multiple authors, one author per row. The attribute no is used to indicate the order of the authors.

During database design, only unquestionable conditions should be used as functional dependencies.

Database normalization **alters the table structure** depending on the specified functional dependencies.

Later hard to change: needs creation/deletion of tables!

#### A table with homework grades:

	HomeworkResults					
sid	first	last	exercise	points	maxPoints	
100	Andrew	Smith	1	9	10	
101	Dave	Jones	1	8	10	
102	Maria	Brown	1	10	10	
101	Dave	Jones	2	11	12	
102	Maria	Brown	2	10	12	

#### Which FDs should hold for this table in general?

- sid  $\rightarrow$  first, last
- exercise → maxPoints
- sid, exercise → first, last, points, maxPoints (a key)

Identify FDs that hold in this table but not in general.

- first  $\rightarrow$  last
- first, last  $\rightarrow$  sid (prevents students with same name)

Database Normalisation :: Implication of Functional Dependencies

### Implication of Functional Dependencies

If  $A \to B$  and  $B \to C$  hold, then  $A \to C$  is holds automatically.

Courses					
courseNr	title	instructor	phone		
230	Databases I	Arthur	9002		
415	Functional Programming	Arthur	9002		
301	Graph Theory	Marvin	8020		

Note that  ${\tt courseNr} \to {\tt phone} \text{ is a consequence of }$ 

 $courseNr \rightarrow instructor$ 

instructor  $\rightarrow$  phone

FDs of the form  $A \rightarrow A$  always hold.

E.g. phone  $\rightarrow$  phone holds, but is not interesting

Implication of Functional Dependencies A set of FDs  $\Gamma$  implies an FD  $\alpha \rightarrow \beta$  $\longleftrightarrow$ every DB state which satisfies all FDs in  $\Gamma$ , also satisfies  $\alpha \rightarrow \beta$ .

The DB designer is normally not interested in all FDs, but only in a **representative FD set** that implies all other FDs.

How to determine whether  $\Gamma$  implies  $\alpha \to \beta$  ?

# Armstrong Axioms

All implied FDs can be derived using the Armstrong axioms.

#### Armstrong axioms

- **Reflexivity:** if  $\beta \subseteq \alpha$ , then  $\alpha \to \beta$ .
- **Augmentation:** if  $\alpha \to \beta$ , then  $\alpha \cup \gamma \to \beta \cup \gamma$ .
- **Transitivity:** if  $\alpha \to \beta$  and  $\beta \to \gamma$ , then  $\alpha \to \gamma$ .

#### Use the Amstrong axioms to show that

- 1. isbn  $\rightarrow$  title, publisher
- **2.** isbn, no  $\rightarrow$  author
- **3**. publisher  $\rightarrow$  publisherURL

implies is bn  $\rightarrow$  publisherURL.

- 4. title, publisher  $\rightarrow$  publisher
- 5. isbn  $\rightarrow$  publisher
- 6. isbn  $\rightarrow$  publisherURL

by reflexivity by transitivity using 1. and 4. by transitivity using 5. and 3.

### Covers

### Simpler way to check if $\alpha \to \beta$ is implied by a set ${\mathcal F}$ of FDs:

- compute the **cover**  $\alpha_{\mathcal{F}}^+$  of  $\alpha$ , and
- then check if  $\beta \subseteq \alpha_{\mathcal{F}}^+$ .

#### Cover

The **cover**  $\alpha_{\mathcal{F}}^+$  of attributes  $\alpha$  with respect to a set  $\mathcal{F}$  of FDs is

$$lpha_{\mathcal{F}}^+ := \{ \ A \mid \mathcal{F} \text{ implies } lpha o A \},$$

the set of all attributes A that are uniquely determined by  $\alpha$ .

The cover  $\gamma_{\mathcal{F}}^+$  can be **computed** as follows:

• Let  $x = \gamma$ , and repeat the next step until *x* is stable.

If  $\alpha \subseteq x$  for some  $(\alpha \rightarrow \beta) \in \mathcal{F}$ , then let  $x = x \cup \beta$ .

Finally *x* is the cover  $\gamma_{\mathcal{F}}^+$  of  $\gamma$  with respect to the set  $\mathcal{F}$  of FDs.

A set of FDs  $\mathcal{F}$  implies an FD  $\alpha \rightarrow \beta$  if and only if  $\beta \subseteq \alpha_{\mathcal{F}}^+$ .

### Covers

Consider the following set  $\mathcal{F}$  of FDs:

- 1. isbn  $\rightarrow$  title, publisher
- 2. isbn, no  $\rightarrow$  author
- 3. publisher  $\rightarrow$  publisherURL

Compute the cover  $\{isbn\}_{\mathcal{F}}^+$ :

- We start with  $x = \{ isbn \}$ .
- FD 1 is applicable since  $\{isbn\} \subseteq x$ .

We get  $x = \{ isbn, title, publisher \}.$ 

• FD 3 is applicable since { publisher }  $\subseteq x$ .

We get  $x = \{ \text{ isbn, title, publisher, publisherURL} \}$ .

No further way to extend set x, thus

 $\{isbn\}_{\mathcal{F}}^{+} = \{isbn, title, publisher, publisherURL\}$ 

We may now conclude, e.g., isbn  $\rightarrow$  publisherURL.

### Database Normalisation :: Determinants

### Determinants

### Determinants (Non-trivial, minimal FDs)

 $\{A_1, \ldots, A_n\}$  is a **determinant** for  $\{B_1, \ldots, B_m\}$  if

- the FD  $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m$  holds, and
- the **left-hand side is minimal**, that is, if any  $A_i$  is removed then  $A_1, \ldots, A_{i-1}, A_{i+1}, A_n \rightarrow B_1, \ldots, B_m$  does *not* hold, and
- it is **not trivial**, that is,  $\{B_1, \ldots, B_m\} \not\subseteq \{A_1, \ldots, A_n\}$ .

(In a canonical set of FDs, all FDs are determinants.)

$$\mathcal{F} = \left\{ egin{array}{ccc} \mathsf{sid}, \mathsf{exercise} & o & \mathsf{points} \ \mathsf{exercise} & o & \mathsf{maxPoints} \end{array} 
ight\}$$

Are the following determinants?

- points, maxPoints for points, maxPoints ? No
- exercise for points, maxPoints ? No
- sid, exercise for points, maxPoints ? Yes
- exercise, points for points, maxPoints ? Yes

Database Normalisation :: Canonical Set of Functional Dependencies

# Canonical Set of Functional Dependencies

Computing a Canonical Set of Functional Dependencies Let a set of FDs  $\mathcal{F}$  be given. Determine a **minimal** (**canonical**) set of FDs that is equivalent to  $\mathcal{F}$  by transforming  $\mathcal{F}$  as follows:

#### 1. Make the right-hand sides singular

Replace every FD  $\alpha \rightarrow B_1, \ldots, B_m$  by  $\alpha \rightarrow B_i, 1 \leqslant i \leqslant m$ .

#### 2. Minimise left-hand sides

For each FD  $\alpha \to B$  and attribute  $A \in \alpha$ : If  $B \in (\alpha - \{A\})^+_{\mathcal{F}}$ , replace  $\alpha \to B$  by  $(\alpha - \{A\}) \to B$  in  $\mathcal{F}$ .

#### 3. Remove implied FDs

For each FD  $\alpha \to B$ : If  $B \in \alpha_{\mathcal{G}}^+$  for  $\mathcal{G} = \mathcal{F} - \{\alpha \to B\}$ , then drop  $\alpha \to B$  from  $\mathcal{F}$ .

Repeat steps 2 and 3 until nothing can be removed.

# Canonical Set of Functional Dependencies

Consider the relation R(A, B, C, D, E) with FDs  $A \rightarrow D, E$   $B \rightarrow C$   $B, C \rightarrow D$   $D \rightarrow E$ 

1. Make the right-hand sides singular

A 
ightarrow D A 
ightarrow E B 
ightarrow C B, C 
ightarrow D D 
ightarrow E

2. Minimise left-hand sides

 $A \rightarrow D$   $A \rightarrow E$   $B \rightarrow C$   $B \rightarrow D$   $D \rightarrow E$ We drop *C* from *B*,  $C \rightarrow D$  since  $D \in \{B\}^+$  due to  $B \rightarrow C$ . 3. Remove implied FDs

 $A \rightarrow D$   $B \rightarrow C$   $B \rightarrow D$   $D \rightarrow E$  $A \rightarrow E$  can still be derived from  $A \rightarrow D$  and  $D \rightarrow E$ . Thus we have obtained the following **canonical** set of FDs:  $A \rightarrow D$   $B \rightarrow C$   $B \rightarrow D$   $D \rightarrow E$ 

# Canonical Set of Functional Dependencies

Compute the canonical set of FDs for  $A, B, C \rightarrow D, E \quad B \rightarrow C \quad B \rightarrow E \quad C \rightarrow E \quad C, D \rightarrow D, F$  Make the right-hand sides singular  $A, B, C \rightarrow D$   $B \rightarrow C$   $B \rightarrow E$   $C \rightarrow E$   $C, D \rightarrow D$  $C, D \rightarrow F$  $A, B, C \rightarrow E$ 2. Removing implied FDs:  $A, B, C \rightarrow E$   $B \rightarrow E$   $C, D \rightarrow D$ This results in  $A, B, C \rightarrow D$   $B \rightarrow C$   $C \rightarrow E$   $C, D \rightarrow F$ Minimise left-hand sides  $A, B \rightarrow D$   $B \rightarrow C$   $C \rightarrow E$   $C, D \rightarrow F$ We drop C from A, B,  $C \rightarrow D$  since  $D \in \{A, B\}^+$ .

There is nothing more that can be removed.

### Database Normalisation :: How to Determine Keys

Given a set of FDs and the set  $\mathcal{A}$  of all attributes of a relation R:  $\alpha \subseteq \mathcal{A}$  is key of  $R \quad \iff \quad \alpha^+ = \mathcal{A}$ 

That is  $\alpha$  is a key if the cover  $\alpha^+$  contains all attributes.

We can use FDs to determine all possible keys of *R*.

Normally, we are interested in minimal keys only.

A key  $\alpha$  is **minimal** if every  $A \in \alpha$  is **vital**, that is

 $(\alpha - \{A\})^+ \neq A$ 

#### **Finding a Minimal Key**

Let A be the attributes of the relation R, and  $\mathcal{F}$  a set of FDs.

- Let x = A.
- If  $A \in (x \{A\})^+_{\mathcal{F}}$  for some  $A \in x$ , then remove A from x.
- Repeat the last step until nothing can be removed.

Finally, x is a minimal key of R.

We might get different keys depending on the order in which we remove attributes.

# Finding a Minimal Key



We determine a minimal key for the relation Results:

- 1.  $x = \{ \text{ sid, exercise, points, maxPoints} \}$
- 2. We remove points since { sid, exercise }  $\subseteq x$ :  $x = \{ \text{ sid, exercise, maxPoints } \}$
- 3. We remove maxPoints since { exercise }  $\subseteq x$ :  $x = \{ \text{ sid, exercise } \}$

Nothing else can be removed. We have a minimal key:

{ sid, exercise }

# Finding all Minimal Keys

#### **Finding all Minimal Keys**

```
Input: A_1, A_2, \ldots, A_n (all attributes of R) and \mathcal{F} (set of FDs)
Output: Results (the set of all minimal keys of R)
 Results = \emptyset:
 Candidates = { { A_i \mid A_i is not part of any right-hand side in \mathcal{F} } };
 while Candidates \neq \emptyset do
    choose and remove a smallest \kappa \in Candidates;
    if k contains no key in Results then
       if \kappa_{\tau}^{+} = \{A_{1}, A_{2}, ..., A_{n}\} then
           Results = Results \cup {\kappa}:
       else
          for all A_i \notin \kappa_F^+ do
              \kappa_i = \kappa \cup \{A_i\};
              Candidates = Candidates \cup { \kappa_i };
          end for
       end if
    end if
 end while
 return Results;
```
### How to Determine Keys: Examples



No more candidate keys to check, we terminate.

### How to Determine Keys: Examples

#### Find **all** minimal keys the relation R(A,B,C,D,E) with

A, D 
ightarrow B, D B, D 
ightarrow C A 
ightarrow E

We get

- 1. Candidates =  $\{ \{A\} \}$  since A not in any right-hand side
- 2. { *A* }<sup>+</sup> = { *A*, *E* }, so we extend with *B*, *C*, *D*: *Candidates* = { { *A*, *B* }, { *A*, *C* }, { *A*, *D* } }
- 3.  $\{A, D\}^+ = \{A, B, C, D, E\}$ . So  $\{A, D\}$  is a key.
- 4. { *A*, *B* }<sup>+</sup> = { *A*, *B*, *E* }, so we extend with *C*, *D*: *Candidates* = { { *A*, *B*, *C* }, { *A*, *B*, *D* }, { *A*, *C* } }
- 5. { *A*, *C* }<sup>+</sup> = { *A*, *C*, *E* }, so we extend with *B*, *D*: *Candidates* = { { *A*, *B*, *C* }, { *A*, *B*, *D* }, { *A*, *C*, *D* } }
- 6. Remove  $\{A, B, D\}$  and  $\{A, C, D\}$  since they contain a key.

7. 
$$\{A, B, C\}^+ = \{A, B, C, E\}$$
 is not a key!  
Extension with *D* again contains a key.

### Database Normalisation :: Consequence of Bad Design

# Consequences of Bad DB Design

Usually a severe sign of **bad DB design** if a table contains an FD (encodes a partial function) that is **not implied by a key**.

Here  $\texttt{instructor} \rightarrow \texttt{phone} \text{ is not implied by a key in:}$ 

	Courses						
<u>c</u>	ourseNr	title	instructor	phone			
	230	Databases I	Arthur	9002			
	415	Functional Programming	Arthur	9002			
	301	Graph Theory	Marvin	8020			

This leads to

redundant storage of certains facts

(here, phone numbers)

insert, update, deletion anomalies

### Consequences of Bad DB Design

Redundant storage is bad for several reasons:

- it wastes storage space
- difficult to ensure integrity when updating the database
  - all redundant copies need to be updated
  - wastes time, inefficient
- need for additional constraints to guarantee integrity
  - ensure that the redundant copies indeed agree
  - e.g. the constraint instructor  $\rightarrow$  phone

#### Problem

General FDs are not supported by relational databases.

The solution is to transform FDs into **key constraints**. This is what **database normalization** tries to do.

# Consequences of Bad DB Design

### Update anomalies

- When a single value needs to be changed (e.g., a phone number), multiple tuples must be updated. This complicates programs and updates takes longer.
- Redundant copies potentially get out of sync and it is impossible/hard to identify the correct information.

#### Insertion anomalies

- The phone number of a new instructor cannot be inserted into the DB until it is known what course she/he will teach.
- Insertion anomalies arise when unrelated concepts are stored together in a single table.

#### **Deletion anomalies**

When the last course of an instructor is deleted, his/her phone number is lost.

### Database Normalisation :: Boyce-Codd Normal Form

# Boyce-Codd Normal Form

A relation *R* is in **Boyce-Codd Normal Form (BCNF)** if all its functional dependencies are implied by its keys. That is, for every FD  $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$  of *R* we have:

- $\{A_1, \ldots, A_n\}$  contains a key of R, or
- the FD is trivial (that is,  $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$ )

The relation

Courses(<u>courseNr</u>, title, instructor, phone) with the FDs

<code>courseNr  $\rightarrow$  title, instructor, phone</code>

 $\texttt{instructor} \ \rightarrow \ \texttt{phone}$ 

is **not in BCNF** because of the FD instructor  $\rightarrow$  phone:

- { instructor } contains no key, and
- the functional dependency is not trivial.

However, the relation Courses(<u>courseNr</u>, title, instructor) without phone is in BCNF.

# Boyce-Codd Normal Form: Examples

Each course meets once per week in a dedicated room:

```
Class(courseNr, title, weekday, time, room)
```

The relation thus satisfies the following FDs (plus implied ones):

 $courseNr \rightarrow title, weekday, time, room$ 

weekday,time,room  $\rightarrow$  courseNr

#### The minimal keys of Class are

- { courseNr }
- { weekday,time,room }
- Is the relation in BCNF?
  - both FDs are implied by keys

(their left-hand sides even coincide with the keys)

Thus Class is in BCNF.

```
Consider the relation
```

```
Product(productNr, name, price)
```

and the following FDs:

productNr	$\rightarrow$	name	price,name	$\rightarrow$	name
productNr	$\rightarrow$	price	productNr,price	$\rightarrow$	name

#### Is this relation in BCNF?

- Note that { productNr } is a key.
- The FD price, name  $\rightarrow$  name is trivial.
- All other FDs are implied by the key { productNr }.

Thus the relation Product is in BCNF.

### Advantages of Boyce-Codd Normal Form

If a relation *R* is in BCNF, then...

- Ensuring its key constraints automatically satisfies all FDs. Hence, no additional constraints are needed for FDs!
- The anomalies (udpate/insertion/deletion) do not occur.

# Boyce-Codd Normal Form: Quiz

### **BCNF Quiz**

1. Consider the relation

Results(sid, exercise, points, maxPoints) with the following FDs sid, exercise  $\rightarrow$  points

exercise  $\rightarrow$  maxPoints

Is this relation in BCNF?

2. Consider the relation

#### with the following FDs

- <code>invoiceNr  $\rightarrow$  date, amount, customerNr</code>
- $\texttt{invoiceNr}, \texttt{date} \quad \rightarrow \quad \texttt{customerName}$ 
  - $\texttt{customerNr} \ \rightarrow \ \texttt{customerName}$
  - $\texttt{date}, \texttt{amount} \ \rightarrow \ \texttt{date}$

Is this relation in in BCNF?

### Database Normalisation :: Third Normal Form

# Third Normal Form

A **key attribute** is an attribute that appears in a minimal key. *Minimality is important, otherwise all attributes are key attributes.* 

Assume that every FD has a single attribute on the right-hand side. If not, expand FDs with multiple attributes on the right-hand side.

### Third Normal Form (3NF)

A relation *R* is in **Third Normal Form (3NF)** if and only if every FD  $A_1, \ldots, A_n \rightarrow B$  satisfies at least one of the conditions:

- $\{A_1, \ldots, A_n\}$  contains a key of R, or
- the FD is trivial (that is,  $B \in \{A_1, \ldots, A_n\}$ ), or
- B is a key attribute of R.

The only difference with BCNF is the last condition.

Third Normal Form (3NF) is slightly weaker than BCNF: If a relation is in BCNF, it is automatically in 3NF.

In short, we can say:

 $\mathsf{BCNF} \iff \mathsf{for every non-trivial FD}:$ 

the left-hand side contains a key

### $3NF \iff$ for every non-trivial FD:

- the left-hand side contains a key, or
- the right-hand side is an attribute of a minimal key

# Third Normal Form vs. Boyce-Codd Normal Form

### **3NF vs BCNF**

Bookings						
court	startTime	endTime	rate			
1	9:30	11:00	saver			
2	9:30	12:00	premium-a			
1	12:00	14:00	standard			

The table contains bookings for one day at a tennis club:

- there are courts 1 (hard court) and 2 (grass court)
- the rates are
  - saver for member bookings of court 1
  - standard for non-member bookings of court 1
  - premium-a for member bookings of court 2
  - premium-b for non-member bookings of court 2

Quiz:

- Find a representative set of functional dependencies.
- Is the table in BCNF? Is the table in 3NF?

### Database Normalisation :: Splitting Relations

# **Splitting Relations**

If a table *R* is not in BCNF, we can **split** it into two tables.

The violating FD determines how to split:

**Table Decomposition** 

If the FD  $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m$  violates BCNF:

- create a new relation  $S(A_1, \ldots, A_n, B_1, \ldots, B_m)$  and
- remove  $B_1, \ldots, B_m$  from the original relation R.

The table

 $\label{eq:courses} \begin{array}{l} \mbox{Courses}(\underline{\mbox{courseNr}},\mbox{ title, instructor, phone}) \\ \mbox{violates BCNF because of instructor} \rightarrow \mbox{phone}. \end{array}$ 

We split into:

Courses(courseNr, title, instructor)
Instructors(instructor, phone)

It is important that this splitting transformation is **lossless**, i.e., that the original relation can be reconstructed by a join.

Reconstruction after split Recall that we have split OldCourses(<u>courseNr</u>, title, instructor, phone) into tables Courses(<u>courseNr</u>, title, instructor) Instructors(instructor, phone)

We can reconstruct the original table as follows:

create view OldCourses(courseNr, title, instructor, phone) as select courseNr, title, C.instructor, phone from Courses C, Instructors I where C.instructor = I.instructor

# Splitting Relations: Lossless Splits

When is a split lossless?

#### **Decomposition Theorem**

The split of relations is **guaranteed to be lossless** if the set of shared attributes of the new tables is a key of at least one.

The join connects tuples depending on the shared attributes. If these values uniquely identify tuples in one relation we do not lose information.

### "Lossy" decomposition



# Splitting Relations: Lossless Splits

```
Lossless split condition satisfied
Recall that we have split
     OldCourses(courseNr, title, instructor, phone)
into tables
           Courses(courseNr, title, instructor)
              Instructors(instructor, phone)
The lossless split condition is satisfied since
     {courseNr, title, instructor} \cap {instructor, phone}
                       ={instructor}
and instructor is a key of the table Instructors.
```

All splits initiated by the **table decomposition method** satisfy the condition of the decomposition theorem.

It is **always possible** to transform a relation into BCNF by lossless splitting.

### Splitting Relations: Lossless Splits

Lossless split guarantees that the schema after splitting can represent all DB states that were possible before.

- we can translate states from the old into the new schema
- old schema can be "simulated" via views

Lossless splits can lead to more general schemas!

the new schema allows states which do not correspond to the state in the old schema

Recall that we have split

OldCourses(<u>courseNr</u>, title, instructor, phone) into tables

Courses(courseNr, title, instructor)
Instructors(instructor, phone)

We may now store instructors and phone numbers without any affiliation to courses.

# Splitting Relations: Unnecessary Splits

Not every lossless split is reasonable!

Students				
<u>sid</u>	first	last		
101	George	Orwell		
102	Elvis	Presley		

Splitting Students into

StudentsFirst(sid, first)
StudentsLast(sid, last)

is lossless, but

- the split is not necessary to enforce a normal form, and
- only requires costly joins in subsequent queries.

# Splitting Relations: Computable Columns

Although **computable columns** lead to violations of BCNF, splitting the relation is **not** the right solution.

E.g. age which is derivable from dateOfBirth.

As a consequence we have a functional dependency:

 $\texttt{dateOfBirth} \rightarrow \texttt{age}$ 

A split would yield a relation:

```
R(dateOfBirth, age)
```

which would try to materialise the computable function.

The **correct solution** is to **eliminate age** from the table and to **define a view** containing all columns plus the **computed** age.

Database Normalisation :: Preservation of Functional Dependencies

# Preservation of Functional Dependencies

Besides losslessness, a desirable property of a decomposition is the **preservation of functional dependencies**:

- A FD can refer only to attributes of a single relation.
- When splitting a relation into two, there might be FDs that can no longer be expressed (they are not preserved).

### FD gets lost during decomposition

Addresses(streetAddress, city, state, zipCode)

with functional dependencies

streetAddress,city,state  $\rightarrow$  zip

 ${\tt zip}~
ightarrow$  state

The second FD violates BCNF and would lead to the split:

- Addresses1(streetAddress, city, zip)
- Addresses2(zip, state)

But now the first FD can no longer be expressed.



### Preservation of Functional Dependencies

```
Addresses(streetAddress, city, state, zipCode)
with functional dependencies
streetAddress,city,state \rightarrow zip
zip \rightarrow state
```

Is the table in 3NF? Yes

- Most designers would not split the table since it is in 3NF.
- Pro split: if there are many addresses with the same ZIP code, there will be significant redundancy.
- **Contra split**: queries will involve more joins.

Whether or not to split depends on the intended application:

A table of ZIP codes might be of interest on its own. E.g. for the database of a mailing company.

### Database Normalisation :: BCNF Synthesis

# **BCNF** Synthesis Algorithm

**BNCF** Synthesis Algorithm

Input: a relation R and a set of FDs for R.

- 1. Compute a **canonical set** of FDs  $\mathcal{F}$ .
- 2. Maximise the right-hand sides of the FDs: Replace every FD  $X \rightarrow Y \in \mathcal{F}$  by  $X \rightarrow X^+ - X$ .
- 3. Split off violating FDs one by one: Start with  $S = \{R\}$ .

For every  $R_i \in S$  and  $X \to Y \in F$ : if

- X is contained in  $R_i$  ( $X \subseteq R_i$ ), and
- X is not a key of  $R_i$  ( $R_i \not\subseteq X^+$ ), and
- *Y* overlaps with  $R_i$  ( $Y \cap R_i \neq \emptyset$ ),

then, let  $Z = Y \cap R_i$  and

- remove attributes Z from the relation R<sub>i</sub>, and
- add a relation with attributes  $X \cup Z$  to S.

# BCNF Synthesis Algorithm: Example

Consider R = (A, B, C, D, E) with the canonical set of FDs

 $A \rightarrow D$   $B \rightarrow C$   $B \rightarrow D$   $D \rightarrow E$ 

Here  $\{A, B\}$  is the only minimal key. Is R in BCNF? No.

1. Maximise the right-hand sides of the FDs:

 $A \rightarrow D, E$   $B \rightarrow C, D, E$   $D \rightarrow E$ 

- 2. Split off violating FD's one by one:
  - $\blacksquare \mathcal{S} = \{ R_0(\underline{A}, \underline{B}, C, D, E) \}$
  - $A \rightarrow D, E$  violates BCNF of  $R_0$
  - $\blacksquare \ \mathcal{S} = \{ R_0(\underline{A}, \underline{B}, C), \ R_1(\underline{A}, D, E) \}$
  - $B \rightarrow C, D, E$  violates BCNF of  $R_0$
  - $\blacksquare \ \mathcal{S} = \{ R_0(\underline{A}, \underline{B}), \ R_1(\underline{A}, D, E), \ R_2(\underline{B}, C) \}$
  - $D \rightarrow E$  violates BCNF of  $R_1$
  - $\blacksquare \ \mathcal{S} = \{ R_0(\underline{A}, \underline{B}), \ R_1(\underline{A}, D), \ R_2(\underline{B}, C), \ R_3(\underline{D}, E) \} \text{- done!}$

Note that we lost the dependency  $B \rightarrow D!$ 

### Database Normalisation :: 3NF Synthesis

The **3NF synthesis algorithm** produces a lossless decomposition of a relation into 3NF that **preserves the FDs**.

3NF Synthesis Algorithm Input: relation *R* and a set of FDs for *R*.

- 1. Compute a **canonical set** of FDs  $\mathcal{F}$ .
- 2. Merge  $\alpha \to \beta_1$  and  $\alpha \to \beta_2$  with the same left-hand side in  $\mathcal{F}$  to the single functional dependency  $\alpha \to \beta_1 \cup \beta_2$ .
- 3. For all  $\alpha \to \beta \in \mathcal{F}$  create a relation with attributes  $\alpha \cup \beta$ .
- 4. If none of the created relations contains a **key of** *R*, add a relation with the attributes of a minimal key of *R*.
- 5. Finally, drop relations  $R_i(\alpha)$  if there is  $R_j(\beta)$  with  $\alpha \subsetneq \beta$ .

# 3NF Synthesis Algorithm: Example

Use the 3NF synthesis algorithm to normalise the relation R(A, B, C, D, E, F)

with the following canonical functional dependencies:

$$A 
ightarrow D$$
  $B 
ightarrow C$   $B 
ightarrow D$   $D 
ightarrow E$ 

1. We already have a **canonical** set of FDs  $\mathcal{F}$ .

- 2. We merge  $B \to C$  and  $B \to D$  to  $B \to C$ , D, yielding:  $A \to D$   $B \to C$ , D  $D \to E$
- 3. We create a relation for every functional dependency:  $R_1(A, D)$   $R_2(B, C, D)$   $R_3(D, E)$
- 4. Does one of these relations contains a key of *R*? No, so we add a relation with a minimal key of *R*: *R*<sub>1</sub>(*A*, *D*) *R*<sub>2</sub>(*B*, *C*, *D*) *R*<sub>3</sub>(*D*, *E*) *R*<sub>4</sub>(*A*, *B*, *F*)
- 5. Nothing to drop, no relation subsumes another.

BCNF does not retain all FDs, therefore 3NF is popular.

Database systems are good at checking **key** constraints, because they create an index on the key columns.

If we leave a table in 3NF (and not BCNF), we have non-key constraints. Namely those FDs that are not implied by keys.

Sometimes we can enforce non-key constraints as key constraints on **materialised views**.

### Database Normalisation :: Multivalued Dependencies

## Introduction

#### Recall the Decomposition Theorem

The split of relations is **guaranteed to be lossless** if the intersection (the shared set attributes) of the attributes of the new tables is a key of at least one of them.

The condition in the decomposition theorem is only

- sufficient (it guarantees losslessness),
- but not necessary (a decomposition might be lossless even if the condition is not satisfied).

Multivalued dependencies (MVDs) are constraints that give a necessary and sufficient condition for lossless decomposition

MVDs lead to the Fourth Normal Form (4NF).
## **Multivalued Dependencies**

The following table shows for each employee:

- knowledge of programming languages
- knowledge of programming DBMSs

Knowledge			
employee	programmingLanguage	dbms	
John Smith	С	Oracle	
John Smith	С	MySQL	
John Smith	C++	Oracle	
John Smith	C++	MySQL	
Maria Brown	Prolog	PostgreSQL	
Maria Brown	Java	PostgreSQL	

There are no non-trivial functional dependencies.

The table is in BCNF.

Nevertheless, there is redundant information.

# **Multivalued Dependencies**

The table contains redundant data & must be split.

KnowledgeLanguage		
employee	programmingLanguage	
John Smith	C	
John Smith	C++	
Maria Brown	Prolog	
Maria Brown	Java	

KnowledgeDBMS			
employee	<u>dbms</u>		
John Smith	Oracle		
John Smith	MySQL		
Maria Brown	PostgreSQL		

Note: table may only be decomposed if programmingLanguage and dbms are **independent**; otherwise **loss of information**.

E.g. it may not be decomposed if the semantics of the table is that the employee knows the interface between the language and the database.

## The multivalued dependency (MVD)

employee ->> programmingLanguage

means that, for each employee, the **set of values** in column programmingLanguage is **independent of all other columns**.

Knowledge		
employee	programmingLanguage	<u>dbms</u>
John Smith	С	Oracle
John Smith	С	MySQL
John Smith	C++	Oracle
John Smith	C++	MySQL
Maria Brown	Prolog	PostgreSQL
Maria Brown	Java	PostgreSQL

The table contains an **embedded function** from the employee to **sets of** programming languages.

## **Multivalued Dependencies**

Formally,  $A \rightarrow B$  holds if: whenever two tuples agree on A, one can exchange their B values and the resulting tuples are in the same table.

Due to employee ---> programmingLanguage and the two table rows

employee	programmingLanguage	<u>dbms</u>
John Smith	С	Oracle
John Smith	C++	MySQL

the table must also contain the following rows

employee	programmingLanguage	<u>dbms</u>
John Smith	C++	Oracle
John Smith	С	MySQL

This expresses the **independence** of programmingLanguage for a given employee from the rest of the table columns.

# Multivalued Dependency A multivalued dependency (MVD) $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ is satisfied in a DB state $I \iff$ for all tuples $t, u \in I(R)$ with $t.A_i = u.A_i, 1 \le i \le n$ there are two tuples $t', u' \in I(R)$ such that 1. t' agrees with t except that $t'.B_i = u.B_i, 1 \le i \le m$ , and 2. u' agrees with u except that $u'.B_i = t.B_i, 1 \le i \le m$ .

The condition means that the values of the  $B_i$  are swapped:

$$t = a_1, \dots, a_n, b_1, \dots, b_m, c_1, \dots, c_k$$
$$t' = a_1, \dots, a_n, b'_1, \dots, b'_m, c'_1, \dots, c'_k$$
$$u' = a_1, \dots, a_n, b'_1, \dots, b'_m, c'_1, \dots, c'_k$$
$$u' = a_1, \dots, a_n, b_1, \dots, b_m, c'_1, \dots, c'_k$$

Multivalued dependencies always come in pairs!

If employee ---- programmingLanguage holds, then employee ---- dbms is automatically satisfied.

More general:

For a relation  $R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k)$ , the following multivalued dependencies are equivalent

$$= A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$$
$$= A_1, \ldots, A_n \twoheadrightarrow C_1, \ldots, C_k$$

Swapping the  $B_j$  values in two tuples is the same as swapping the values for all other columns (the  $A_i$  values are identical, swapping them has no effect).

## **Multivalued Dependencies**

If the FD  $A_1, \ldots A_n \rightarrow B_1, \ldots B_m$  holds, the corresponding MVD

$$A_1,\ldots,A_n \twoheadrightarrow B_1,\ldots,B_m$$

is trivially satisfied.

The FD means that if tuples t, u agree on the  $A_i$  then also on the  $B_j$ . Swapping thus has no effect (yields t, u again).

### Deduction rules to derive all implied FDs/MVDs

- The three Armstrong Axioms for FDs.
- If  $\alpha \twoheadrightarrow \beta$  then  $\alpha \twoheadrightarrow \gamma$ , where  $\gamma$  are all remaining columns.
- If  $\alpha_1 \twoheadrightarrow \beta_1$  and  $\alpha_2 \supseteq \beta_2$  then  $\alpha_1 \cup \alpha_2 \twoheadrightarrow \beta_1 \cup \beta_2$ .
- If  $\alpha \twoheadrightarrow \beta$  and  $\beta \twoheadrightarrow \gamma$  then  $\alpha \twoheadrightarrow (\gamma \beta)$ .
- If  $\alpha \to \beta$ , then  $\alpha \twoheadrightarrow \beta$ .
- If  $\alpha \twoheadrightarrow \beta$  and  $\beta' \subseteq \beta$  and there is  $\gamma$  with  $\gamma \cap \beta = \emptyset$  and  $\gamma \to \beta'$ , then  $\alpha \to \beta'$ .

# Fourth Normal Form

Fourth Normal Form (4NF)

A relation is in Fourth Normal Form (4NF) if every MVD

$$A_1,\ldots,A_n \twoheadrightarrow B_1,\ldots,B_m$$

#### is

- either trivial, or
- implied by a key.

This definition of 4NF is very similar to BCNF but with a focus on implied MVDs (not FDs).

Since every FD is also an MVD, 4NF is stronger than BCNF. That is, if a relation is in 4NF, it is automatically in BCNF.

However, it is not very common that 4NF is violated, but BCNF is not.

The relation

Knowledge(employee, programmingLanguage, dbms)

is an example of a relation that is in BCNF, but not in 4NF. The relation has no non-trivial FDs. Database Normalisation :: Normal Forms and Entity-Relationship Models

# Introduction

If a "good" ER schema is transformed into the relational model, the result will **satisfy all normal forms** (4NF, BCNF, 3NF).

A normal form violation detected in the generated relational schema indicates a **flaw** in the input ER design.

This needs to be corrected on the ER level.

### FDs in the ER model

The ER equivalent of the very first example in this chapter:



- The FD instructor → phone leads to a violation of BCNF in the resulting table for entity Courses.
- Also in the ER model, FDs between attributes of an entity set should be implied by a key constraint.

In the ER model, the solution is the "same" as in the relational model: we have to **split** the entity set.



## Examples

Functional dependencies between **attributes of a relationship** always violate BCNF.



The FD orderNr  $\rightarrow$  date violates BCNF.

The key of the table corresponding to the relationship set "orders" consists of the attributes customerNr, productNr.

This shows that the concept "order" is an independent entity.

# Violations of BCNF might also be due to the **wrong placement** of an attribute.



The relationship is translated into

Takes(studentId, courseNr, email)

- $\blacksquare$  Then the FD studentId  $\rightarrow$  email violates BCNF.
- Obviously, email should be an attribute of Students.

## Examples

If an attribute of a ternary relationship depends only on two of the entities, this violates BCNF.



If every course is taught only once per term, then attribute room depends only on term and course (but not instructor).

Then the FD term, course  $\rightarrow$  room violates BCNF.

Relational normalization is about:

- Avoiding redundancy.
- Storing separate facts (functions) separately.
- Transforming general integrity constraints into constraints that are supported by the DBMS: keys.

## Database Normalisation :: Denormalization

**Denormalization** is the process of **adding redundant columns** to the database in order to **improve performance**.

### Redundant data storage

For example, if an application extensively access the phone number of instructors, performance-wise it may make sense to add column phone to table Courses.

Courses			
<u>courseNr</u>	title	instructor	phone

This **avoids the otherwise required joins** (on attribute instructor) between tables Courses and Instructors.

## Denormalization

Since there is still the separate table Instructors, **insertion** and deletion anomalies are avoided.

But there will be **update anomalies** (changing a single phone number requires the update of many rows).

The performance gain is thus paid for with

- a more complicated application logic (e.g., the need for triggers)
- risk that a faulty application turns the DB inconsistent

Denormalization may not only be used to avoid joins:

- Complete separate, redundant tables may be created (increasing the potential for parallel operations).
- Columns may be added which aggregate information in other columns/rows.

## Database Normalisation :: Other Constraints

### Multiple choice test

The following relation stores the solutions to a typical multiple choice test:

Answers			
question	answer	text	correct
1	а		Y
1	b		N
1	с		N
2	а		N
2	b		Y
2	С		N

### Using keys to enforce other constraints

The following is not an FD, MVD, or JD:

"Each question can only have one correct answer."

Can you suggest a transformation of table Answers such that the above constraint is already implied by a key?

# Other Constraints

## Solution 1

We could have separate tables for correct and wrong answers:

- CorrectAnswers(question, answer, text)
- WrongAnswers(question, answer, text)

Observe that the key in CorrectAnswers ensures that there is only one correct answer per question.

However, requires a new inter-relational constraint: the same question with the same answer may not appear in both tables.

### Solution 2

We could have separate tables for correct and wrong answers:

- Questions((question, correctAnswer)  $\rightarrow$  Answers, text)
- Answers(question  $\rightarrow$  Questions, answer, text)

Here the correct answer is indicated via a foreign key in Questions referencing the Answers table.