



Skript zur Vorlesung  
**Datenbanksysteme I**  
Wintersemester 2012/2013

# Kapitel 7: Normalformen

Vorlesung: Prof. Dr. Christian Böhm  
Übungen: Sebastian Goebel, Nina Hubig

Skript © 2004 Christian Böhm  
ergänzt von Matthias Schubert 2005

[http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme\\_I](http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme_I)



# Relationaler Datenbank-Entwurf

- Schrittweises Vorgehen:
  - Informelle Beschreibung: **Pflichtenheft**
  - Konzeptioneller Entwurf: **E/R-Diagramm**
  - Relationaler DB-Entwurf: **Relationenschema**
- In diesem Kapitel:
  - Normalisierungstheorie als formale Grundlage für den relationalen DB-Entwurf**
- Zentrale Fragestellungen:
  - Wie können Objekte und deren Beziehungen ins relationale Modell überführt werden
  - Bewertungsgrundlagen zur Unterscheidung zwischen „guten“ und „schlechten“ relationalen DB-Schemata



# Motivation Normalisierung

- **Relation Lieferant:**

<u>LNr</u>	LName	LStadt	LLand	<u>Ware</u>	Preis
103	Huber	Berlin	Deutschland	Schraube	50
103	Huber	Berlin	Deutschland	Draht	20
103	Huber	Berlin	Deutschland	Nagel	40
...	...	...	...	...	...
762	Maier	Zürich	Schweiz	Nagel	45
762	Maier	Zürich	Schweiz	Schraube	55

- **Hier gibt es offensichtlich einige Redundanzen:**

- Wenn bei mehreren Tupeln der Attributwert LNr gleich ist, dann müssen auch die Attributwerte von LName, LStadt und LLand gleich sein.
- Wenn (auch bei verschiedenen LNr) LStadt gleich ist, dann muss auch LLand gleich sein

- **Redundanzen durch funktionale Abhängigkeiten**

- Wir sagen: Die Attribute LName, LStadt und LLand sind **funktional abhängig** vom Attribut LNr (ebenso LLand von LStadt);  
Exakte Definition siehe Seite 8ff.



# Motivation Normalisierung

- **Relation Lieferant:**

<u>LNr</u>	LName	LStadt	LLand	<u>Ware</u>	Preis
103	Huber	Berlin	Deutschland	Schraube	50
103	Huber	Berlin	Deutschland	Draht	20
103	Huber	Berlin	Deutschland	Nagel	40
...	...	...	...	...	...
762	Maier	Zürich	Schweiz	Nagel	45
762	Maier	Zürich	Schweiz	Schraube	55

- Redundanzen führen zu Speicherplatzverschwendung;
- Das eigentliche Problem sind aber Anomalien (Inkonsistenzen durch Änderungsoperationen) und dass das Schema nicht intuitiv ist:
  - **Update-Anomalie:** Änderung der Adresse (LName, LStadt, LLand) in nur *einem* Tupel statt in *allen* Tupeln zu einer LNr.
  - **Insert-Anomalie:** Einfügen eines Tupels mit inkonsistenter Adresse; Einfügen eines Lieferanten erfordert Ware.
  - **Delete-Anomalie:** Löschen der letzten Ware löscht die Adresse.



# Verbesserung

- Neues Datenbankschema:

<b>LieferantAdr</b>	( <u>LNr</u> , LName, LStadt)
<b>Stadt</b>	( <u>LStadt</u> , LLand)
<b>Angebot</b>	( <u>LNr</u> , <u>Ware</u> , Preis)

- Vorteile:
  - keine Redundanz
  - keine Anomalien
- Nachteil:
  - Um zu einer Ware die Länder der Lieferanten zu finden, ist ein zweifacher Join nötig (teuer auszuwerten und umständlich zu formulieren)



# Ursprüngliche Relation

- Die ursprüngliche Relation Lieferant kann mit Hilfe einer View simuliert werden:

```
create view Lieferant as
  select    L.LNr, LName, L.LStadt, LLand, Ware, Preis
  from      LieferantAdr L, Stadt S, Angebot A
  where     L.LNr = A.LNr and L.LStadt = S.LStadt
```

- Dies spart die Schreiarbeit beim Formulieren von Anfragen und macht die Anfragen übersichtlicher.
- In diesem Fall ist die View *nicht updatable* (die Änderungsoperationen verursachen ja die Probleme).
- Da die View nur eine *virtuelle Relation* ist, muss der Join trotzdem ausgewertet werden (weniger effizient).



# Schema-Zerlegung

- Anomalien entstehen durch Redundanzen
- Entwurfsziele:
  - Vermeidung von Redundanzen
  - Vermeidung von Anomalien
  - evtl. Einbeziehung von Effizienzüberlegungen
- Vorgehen:  
Schrittweises Zerlegen des gegebenen Schemas (Normalisierung) in ein äquivalentes Schema ohne Redundanz und Anomalien
- Formalisierung von Redundanz und Anomalien:  
*Funktionale Abhängigkeit*



# Funktionale Abhängigkeit

(engl. Functional Dependency, FD)

- beschreibt Beziehungen zwischen den Attributen einer Relation
- Schränkt das Auftreten gleicher bzw. ungleicher Attributwerte innerhalb einer Relation ein  
→ spezielle Integritätsbedingung (nicht in SQL)

Wiederholung Integritätsbedingungen in SQL:

- Primärschlüssel
- Fremdschlüssel (referenzielle Integrität)
- **not null**
- **check**



# Wiederholung *Schlüssel*

## Definition:

- Eine Teilmenge  $S$  der Attribute eines Relationenschemas  $R$  heißt *Schlüssel*, wenn gilt:
  - *Eindeutigkeit*  
Keine Ausprägung von  $R$  kann zwei verschiedene Tupel enthalten, die sich in *allen* Attributen von  $S$  gleichen.
  - *Minimalität*  
Keine echte Teilmenge von  $S$  erfüllt bereits die Bedingung der Eindeutigkeit



# Konventionen zur Notation

- Ab jetzt gilt die Notation:
  - $A, B, C$  bezeichnen einzelne Attribute
  - $X, Y, Z$  bezeichnen Mengen von Attributen
- Zur Vereinfachung gilt außerdem:
  - $A, B \rightarrow C$  bezeichne  $\{A, B\} \rightarrow \{C\}$
  - $X \rightarrow Y, Z$  bezeichne  $X \rightarrow Y \cup Z$
  - $t.A$  bezeichne das Attribut  $A$  des Tupels  $t$
  - $t.X$  bezeichne die Menge  $X$  von Attributen des Tupels  $t$
  - $t.X = r.X$  bezeichne  $\forall A \in X : t.A = r.A$



# Definition: *funktional abhängig*

- **Gegeben:**
  - Ein Relationenschema  $R$
  - $X, Y$ : Zwei Mengen von Attributen von  $R$  ( $X, Y \subseteq R$ )

- **Definition:**

$Y$  ist von  $X$  funktional abhängig ( $X \rightarrow Y$ )

gdw.  $\forall$  Tupel  $t$  und  $r : t.X = r.X \Rightarrow t.Y = r.Y$

(für alle möglichen Ausprägungen von  $R$  gilt:

Zu jedem Wert in  $X$  existiert genau ein Wert von  $Y$ .)

- Bsp.: **Lieferant** (LNr, LName, LStadt, LLand, Ware, Preis):
  - LNr  $\rightarrow$  LName
  - LNr  $\rightarrow$  LStadt
  - LStadt  $\rightarrow$  LLand
  - LNr, Ware  $\rightarrow$  LName
  - LNr, Ware  $\rightarrow$  LStadt
  - LNr, Ware  $\rightarrow$  Preis



# Vergleich mit *Schlüssel*

- Gemeinsamkeiten zwischen dem *Schlüssel* im relationalen Modell und *Funktionaler Abhängigkeit*:
  - Für jeden Schlüsselkandidaten  $S = \{A, B, \dots\}$  gilt: Alle Attribute der Relation sind von  $S$  funktional abhängig (wegen der Eindeutigkeit):
$$S \rightarrow R$$
  - Jede Menge, von der  $R$  FD ist, ist Superschlüssel.
- Unterschied:
  - Aber es gibt u.U. weitere funktionale Abhängigkeiten: Ein Attribut  $B$  kann z.B. auch funktional abhängig sein
    - von Nicht-Schlüssel-Attributen
    - von nur einem Teil eines Schlüssels

**→ FD ist Verallgemeinerung des Schlüssel-Konzepts.**



# Vergleich mit *Schlüssel*

- Wie der Schlüssel ist auch die funktionale Abhängigkeit eine **semantische Eigenschaft** des Schemas:
  - FD nicht aus aktueller DB-Ausprägung entscheidbar
  - sondern muss für alle möglichen Ausprägungen gelten

## Prime Attribute

- Definition:  
Ein Attribut heißt **prim**,  
wenn es Teil eines Schlüsselkandidaten ist



# Partielle und volle FD

- Ist ein Attribut  $B$  funktional von  $A$  abhängig, dann auch von jeder Obermenge von  $A$ .  
Man ist interessiert, minimale Mengen zu finden, von denen  $B$  abhängt (vgl. Schlüsseldefinition)
- **Definition:**
  - Gegeben: Eine funktionale Abhängigkeit  $X \rightarrow Y$
  - Wenn es keine echte Teilmenge  $X' \subsetneq X$  gibt, von der  $Y$  ebenfalls funktional abhängt,
  - dann heißt  $X \rightarrow Y$  eine **volle funktionale Abhängigkeit** ( $X \twoheadrightarrow Y$ )
  - andernfalls eine **partielle funktionale Abhängigkeit**



# Partielle und volle FD

- Beispiele:
  - $LNr \rightarrow LName$  **voll funktional abhängig**  
(ein-elementige Menge ist immer minimal)
  - $LNr, Ware \rightarrow LName$  **partiell funktional abhängig**  
(da schon  $LNr \rightarrow LName$  gilt)
  - $Ware ? Preis$  **nicht funktional abhängig** (siehe Beispiel)
  - $LNr ? Preis$  **nicht funktional abhängig** (siehe Beispiel)
  - $LNr, Ware \rightarrow Preis$  **voll funktional abhängig** (weder von  $LNr$  noch von  $Ware$  alleine funktional abhängig)



# Volle FD vs Minimalität des Schlüssels

- Definitionen sehr ähnlich:  
Schlüssel: Minimale Menge, die Eindeutigkeit erfüllt  
Volle FD: Minimale Menge, von der ein Attribut abhängt  
(Und die Eindeutigkeit ist äquivalent zur FD, s. S. 12)
- Frage: Folgt aus der Minimalität des Schlüssels, dass alle Attribute (immer) voll funktional abhängig sind?
- Antwort: Leider nicht. (Warum eigentlich „*leider*“?)
- Aber wo liegt der Denkfehler?  
Der Allquantor ist bei beiden Definitionen verschieden:
  - Schlüssel: Minimale Menge, von der alle Attribute funktional abhängig sind.
  - Volle FD  $X \twoheadrightarrow Y$ : Für jedes Attribut  $A \in Y$  gilt:  $X$  ist die minimale Menge von der  $A$  funktional abhängig ist.



# Herleitung funktionaler Abhängigkeit

## Armstrong Axiome

- Reflexivität (R): Falls  $Y$  eine Teilmenge von  $X$  ist ( $Y \subseteq X$ ), dann gilt immer  $X \rightarrow Y$ . Insbesondere gilt also immer  $X \rightarrow X$ .
- Verstärkung (VS): Falls  $X \rightarrow Y$  gilt, dann gilt auch  $XZ \rightarrow YZ$ . Hierbei steht  $XZ$  für  $X \cup Z$ .
- Transitivität (T): Falls  $X \rightarrow Y$  und  $Y \rightarrow Z$  gilt, dann gilt auch  $X \rightarrow Z$ .

Diese Axiome sind **vollständig** und **korrekt** :

Sei  $F$  eine Menge von FDs:

- Es lassen sich nur FDs von  $F$  ableiten, die von jeder Relationen-Ausprägung erfüllt werden, für die auch  $F$  erfüllt ist.
- Es sind alle FDs ableitbar, die durch  $F$  impliziert sind.



# Herleitung funktionaler Abhängigkeit

## Triviale funktionale Abhängigkeit:

- Wegen Reflexivität ist jedes Attribut funktional abhängig:
  - von sich selbst
  - von jeder Obermenge von sich selbst

Solche Abhängigkeiten bezeichnet man als **trivial**.

## Symmetrieeigenschaften funktionaler Abhängigkeiten

- Bei den funktionalen Abhängigkeiten gibt es kein Gesetz zur Symmetrie oder Antisymmetrie, d.h. bei zwei bel. Attribut (-Mengen)  $X, Y$  sind alle 4 Fälle möglich:
  - Es gilt nur  $X \rightarrow Y$ ,
  - Es gilt  $X \rightarrow Y$  und  $Y \rightarrow X$ ,
  - Es gilt nur  $Y \rightarrow X$ ,
  - Es gibt keine FD. zw.  $X$  und  $Y$ .



# Hülle einer Attributmenge

- Eingabe: eine Menge  $F$  von FDs und eine Menge von Attributen  $X$ .
- Ausgabe: die vollständige Menge von Attributen  $X^+$ , für die gilt  $X \rightarrow X^+$ .

AttrHülle( $F, X$ )

Erg :=  $X$

**while** (Änderungen an Erg) **do**

**foreach** FD  $Y \rightarrow Z \in F$  **do**

**if**  $Y \subseteq \text{Erg}$  **then** Erg := Erg  $\cup Z$

Ausgabe  $X^+ = \text{Erg}$



# Hülle einer Attributmeng

- **Beispiel:**  $\text{AttrH\u00fclle}(F, \{\text{LNr}\})$  mit  
 $F = \{\text{LNr} \rightarrow \text{LName}; \text{LNr} \rightarrow \text{LStadt}; \text{LStadt} \rightarrow \text{LLand};$   
 $\text{LNr}, \text{Ware} \rightarrow \text{Preis}\}$
- $\text{Erg}_i = \text{Erg}$  nach  $i$ -tem Durchlauf der **while**-Schleife
- $\text{Erg}_0 = \{ \text{LNr} \}$
- $\text{Erg}_1 = \{ \text{LNr}, \text{LName}, \text{LStadt} \}$
- $\text{Erg}_2 = \{ \text{LNr}, \text{LName}, \text{LStadt}, \text{LLand} \}$
- $\text{Erg}_3 = \{ \text{LNr}, \text{LName}, \text{LStadt}, \text{LLand} \} = \text{Erg}_2$



# Herleitung funktionaler Abhängigkeit

## Weitere Axiome

- Vereinfachen Herleitungsprozess, aber nicht notwendig (da Armstrong-Axiome vollständig)
- Vereinigungsregel (VE):  
Falls  $X \rightarrow Y$  und  $X \rightarrow Z$  gilt, dann gilt auch  $X \rightarrow YZ$ .
- Dekompositionsregel (D):  
Falls  $X \rightarrow YZ$  gilt, dann gilt auch  $X \rightarrow Y$  und  $X \rightarrow Z$ .
- Pseudotransitivitätsregel (P):  
Falls  $X \rightarrow Y$  und  $ZY \rightarrow V$  gilt, dann gilt auch  $XZ \rightarrow V$ .



# Herleitung funktionaler Abhängigkeit

**Beispiel:** Zeige, dass (LNr, Ware) Schlüsselkandidat ist.

• Gegeben:  $F = \{ \text{LNr} \rightarrow \text{LName}; \text{LNr} \rightarrow \text{LStadt}; \text{LStadt} \rightarrow \text{LLand}; \text{LNr, Ware} \rightarrow \text{Preis} \}$ .

• Zu zeigen: (LNr, Ware) eindeutig und minimal.

• Beweis:

(I) (LNr, Ware) eindeutig                      gdw.

$(\text{LNr, Ware} \rightarrow \text{LNr, LName, LStadt, LLand, Ware, Preis}) \in F^+$ ,

d.h. ist aus F herleitbar.

(a)  $(\text{LNr} \rightarrow \text{LName}) \in F \subseteq F^+ \xrightarrow{\text{VS}} (\text{LNr, Ware} \rightarrow \text{LName, Ware}) \in F^+$

(b)  $(\text{LNr} \rightarrow \text{LStadt}) \in F^+ \xrightarrow{\text{VS}} (\text{LNr, Ware} \rightarrow \text{LStadt, Ware}) \in F^+$

(c)  $(\text{LNr} \rightarrow \text{LStadt}) \in F^+$  und  $(\text{LStadt} \rightarrow \text{LLand}) \in F^+$

$\xrightarrow{\text{T}} (\text{LNr} \rightarrow \text{LLand}) \in F^+ \xrightarrow{\text{VS}} (\text{LNr, Ware} \rightarrow \text{LLand, Ware}) \in F^+$

(d) Wegen R gilt  $(\text{LNr} \rightarrow \text{LNr}) \in F^+ \xrightarrow{\text{VS}} (\text{LNr, Ware} \rightarrow \text{LNr, Ware}) \in F^+$

(e) aus (a) bis (d) und  $(\text{LNr, Ware} \rightarrow \text{Preis})$  folgt (I) nach VE, qed.



# Herleitung funktionaler Abhängigkeit

**Beispiel:** Zeige, dass  $(\text{LNr}, \text{Ware})$  Schlüsselkandidat ist.

- Gegeben:  $F = \{ \text{LNr} \rightarrow \text{LName}; \text{LNr} \rightarrow \text{LStadt}; \text{LStadt} \rightarrow \text{LLand}; \text{LNr}, \text{Ware} \rightarrow \text{Preis} \}$ .
- Zu zeigen:  $(\text{LNr}, \text{Ware})$  eindeutig und minimal.
- Beweis:
  - (II)  $(\text{LNr}, \text{Ware})$  minimal  $\quad \text{gdw.}$   
 $(\text{LNr} \rightarrow \text{LNr}, \text{LName}, \text{LStadt}, \text{LLand}, \text{Ware}, \text{Preis})$  und  
 $(\text{Ware} \rightarrow \text{LNr}, \text{LName}, \text{LStadt}, \text{LLand}, \text{Ware}, \text{Preis})$  gelten nicht.
    - (a) Preis ist nur von LNr und Ware gemeinsam funktional abhängig.
    - (b) Weder LNr kann aus Ware hergeleitet werden, noch Ware von LNr.
    - (c) aus (a) und (b) folgt  $(\text{LNr}, \text{Ware})$  minimal, qed.



# Normalisierung

- In einem Relationenschema sollen also möglichst keine funktionalen Abhängigkeiten bestehen, außer vom gesamten Schlüssel
- Verschiedene Normalformen beseitigen unterschiedliche Arten von funktionalen Abhängigkeiten bzw. Redundanzen/Anomalien
  - 1. Normalform
  - 2. Normalform
  - 3. Normalform
  - Boyce-Codd-Normalform
  - 4. Normalform
- Herstellung einer Normalform durch verlustlose Zerlegung des Relationenschemas

*impliziert*

*impliziert*

*impliziert*

*impliziert*



# 1. Normalform

- Keine Einschränkung bezüglich der FDs
- Ein Relationenschema ist in erster Normalform, wenn alle Attributwerte *atomar* sind
- In relationalen Datenbanken sind nicht-atomare Attribute ohnehin nicht möglich
- Nicht-atomare Attribute z.B. durch **group by**

A	B	C	D
1	2	3 4	4 5
2	3	3	4
3	3	4 6	5 7

„nested relation“  
non first normal form  
In SQL nur temporär  
erlaubt



## 2. Normalform

- Motivation:  
Man möchte verhindern, dass Attribute nicht vom gesamten Schlüssel voll funktional abhängig sind, sondern nur von einem Teil davon.
- Beispiel:



<u>LNr</u>	LName	LStadt	LLand	<u>Ware</u>	Preis
103	Huber	Berlin	Deutschland	Schraube	50
103	Huber	Berlin	Deutschland	Draht	20
103	Huber	Berlin	Deutschland	Nagel	40
...	...	...	...	...	...
762	Maier	Zürich	Schweiz	Nagel	45
762	Maier	Zürich	Schweiz	Schraube	55

Konsequenz: In den abhängigen Attributen muss dieselbe Information immer wiederholt werden



## 2. Normalform

- Dies fordert man vorerst nur für Nicht-Schlüssel-Attribute (für die anderen z.T. schwieriger, siehe Seite 44ff)
- Definition  
Ein Schema ist in zweiter Normalform, wenn jedes Attribut
  - voll funktional abhängig von *allen* Schlüsselkandidaten
  - oder
  - prim ist
- Beobachtung:  
Zweite Normalform kann nur verletzt sein, wenn...
  - ...ein zusammengesetzter Schlüssel (-Kandidat) existiert
  - ...und wenn nicht-prime Attribute existieren



## 2. Normalform

- Zur Transformation in 2. Normalform spaltet man das Relationenschema auf:
  - Attribute, die voll funktional abhängig vom Schlüssel sind, bleiben in der Ursprungsrelation  $R$
  - Für alle Abhängigkeiten  $X_i \rightarrow Y_i$  von einem Teil eines Schlüssels ( $X_i \subsetneq S$ ) geht man folgendermaßen vor:
    - Lösche die Attribute  $Y_i$  aus  $R$
    - Gruppier die Abhängigkeiten nach gleichen linken Seiten  $X_i$
    - Für jede Gruppe führe eine neue Relation ein mit allen enthaltenen Attributen aus  $X_i$  und  $Y_i$
    - $X_i$  wird Schlüssel in der neuen Relation



## 2. Normalform

- Beispiel:

**Lieferant** (LNr, LName, LStadt, LLand, Ware, Preis)

partielle Abhängigkeiten

- Vorgehen:

- LName, LStadt und LLand werden aus Lieferant gelöscht

- Gruppierung:

Nur eine Gruppe mit LNr auf der linken Seite

- es könnten Attribute von Ware abhängen (2. Gruppe)

- Erzeugen einer Relation mit LNr, LName, LStadt und LLand

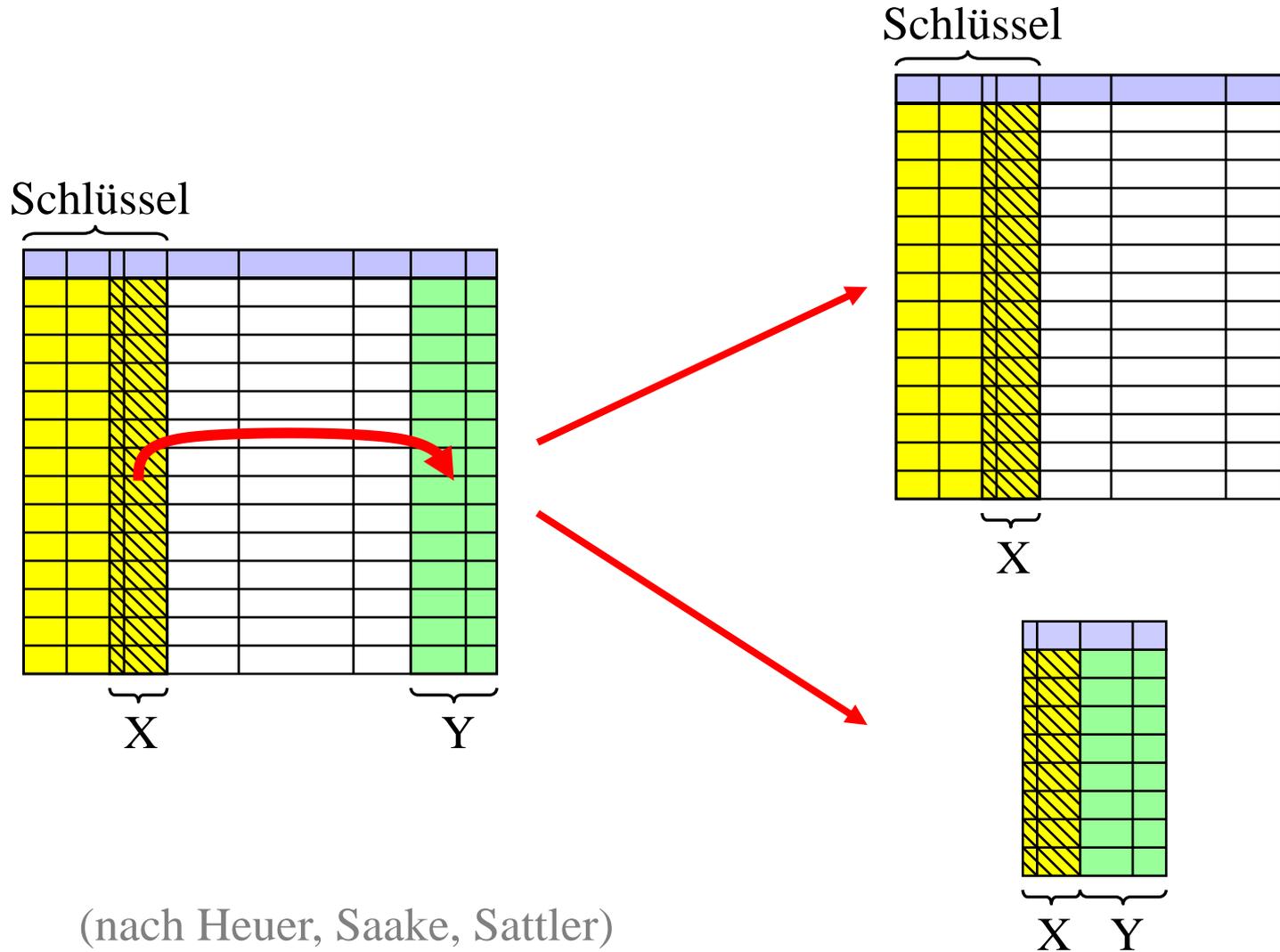
- Ergebnis: **Lieferung** (LNr, Ware, Preis)

**LieferAdr** (LNr, LName, LStadt, LLand)



# Grafische Darstellung

Datenbanksysteme I  
Kapitel 7: Normalformen



(nach Heuer, Saake, Sattler)



# 3. Normalform

- Motivation:  
Man möchte zusätzlich verhindern, dass Attribute von nicht-primen Attributen funktional abhängig sind.

- Beispiel:

**LieferAdr** (LNr, LName, LStadt, LLand)



001	Huber	München	Deutschland
002	Meier	Berlin	Deutschland
003	Müller	Berlin	Deutschland
004	Hinz	Salzburg	Österreich
005	Kunz	Salzburg	Österreich

- Redundanz: Land mehrfach gespeichert
- Anomalien?



# 3. Normalform

- Abhängigkeit von Nicht-Schlüssel-Attribut bezeichnet man häufig auch als *transitive* (d.h. durch Armstrong-Transitivitätsaxiom hergeleitete) *Abhängigkeit* vom Primärschlüssel

- weil Abhängigkeit *über* ein drittes Attribut besteht:



- Definition:  
Ein Relationenschema ist in 3. Normalform, wenn für jede nicht-triviale funktionale Abhängigkeit  $X \rightarrow A$  gilt:
  - $X$  enthält einen Schlüsselkandidaten
  - oder  $A$  ist prim.



# 3. Normalform

Anmerkungen zum Verständnis dieser Definition:

- Wieder Beschränkung auf die FDs, bei denen die rechte Seite nicht-prim ist  
(Abhängigkeiten unter Schlüsselkandidaten sind schwieriger, siehe Boyce-Codd-Normalform S. 44ff)
- Intuitiv möchte die Definition sagen:  
Nicht-prime Attribute sind nur von (ganzen) Schlüsselkandidaten funktional abhängig, also:  
Für jede FD gilt: Linke Seite *ist* ein Schlüsselkandidat.
  - Keine Partiellen FDs von Schlüsselkandidaten:  
→ 2. Normalform ist mit 3. Normalform impliziert
  - Keine FDs, bei denen linke Seite kein Schlüssel ist bzw. Teile enthält, die nicht prim sind.



# 3. Normalform

## Anmerkungen zum Verständnis dieser Definition:

- Intuitiv möchte die Definition sagen:  
Nicht-prime Attribute sind nur vom (ganzen) Schlüsselkandidaten funktional abhängig, also:  
Für jede FD gilt: Linke Seite *ist* ein Schlüsselkandidat.
- Wegen Reflexivitäts- und Verstärkungsaxiom und Allquantor bei den FDs muss man aber berücksichtigen:
  - Es gelten immer auch die trivialen FDs, z.B.  $A \rightarrow A$  (für jedes beliebige Attribut  $A$ ). Deshalb Ergänzung der Definition: „Für jede *nicht-triviale* FD gilt...“
  - Ist  $S$  Schlüssel, dann gilt immer auch  $S' \rightarrow R$  für jede Obermenge  $S' \supseteq S$ . Deshalb heißt es in der Definition:  $X$  *enthält* einen Schlüssel. (statt  $X$  *ist* ein Schlüssel.)



# 3. Normalform

- Transformation in 3. Normalform wie vorher
  - Attribute, die voll funktional abhängig vom Schlüssel sind, und nicht abhängig von Nicht-Schlüssel-Attributen sind, bleiben in der Ursprungsrelation  $R$
  - Für alle Abhängigkeiten  $X_i \rightarrow Y_i$  von einem Teil eines Schlüssels ( $X_i \subsetneq S$ ) oder von Nicht-Schlüssel-Attribut:
    - Lösche die Attribute  $Y_i$  aus  $R$
    - Gruppier die Abhängigkeiten nach gleichen linken Seiten  $X_i$
    - Für jede Gruppe führe eine neue Relation ein mit allen enthaltenen Attributen aus  $X_i$  und  $Y_i$
    - $X_i$  wird Schlüssel in der neuen Relation





# Verlustlosigkeit der Zerlegung

- Die Zerlegung einer Relation  $R$  in die beiden Teilrelationen  $R_1$  und  $R_2$  heißt *verlustlos* (oder auch *verbundtreu*), wenn gilt:

$$R = R_1 \bowtie R_2$$

- Man kann zeigen, dass die Zerlegung verlustlos ist, wenn mindestens eine der beiden folgenden FDs gilt:
  - $(R_1 \cap R_2) \rightarrow R_1$
  - $(R_1 \cap R_2) \rightarrow R_2$
- Intuitiv ist klar, dass Verlustlosigkeit von größter Wichtigkeit ist, weil andernfalls gar nicht dieselbe Information in den Relationenschemata  $R_1$  und  $R_2$  gespeichert werden kann, wie in  $R$ .



# Abhängigkeitserhaltung

- Die Zerlegung einer Relation  $R$  in die beiden Teilrelationen  $R_1$  und  $R_2$  heißt *abhängigkeitserhaltend* (oder auch *hüllentreu*), wenn gilt:

$$F_R = (F_{R_1} \cup F_{R_2}) \quad \text{bzw.} \quad F_R^+ = (F_{R_1} \cup F_{R_2})^+$$

- Das heißt, jede funktionale Abhängigkeit soll mindestens einer der Teilrelationen vollständig zugeordnet sein (also alle Attribute der linken Seite und das Attribut der rechten Seite müssen in einer der Teilrelationen enthalten sein).
- Zwar führt die Verletzung der Abhängigkeitserhaltung nicht zu einer Veränderung der speicherbaren Information, aber eine „verlorene“ FD ist nicht mehr überprüfbar, ohne dass der Join durchgeführt wird.  
→ Verschlechterung der Situation (Redundanzen)



# Synthesealgorithmus für 3NF

## Synthesealgorithmus für 3NF

- Der sogenannte *Synthesealgorithmus* ermittelt zu einem gegebenen Relationenschema  $R$  mit funktionalen Abhängigkeiten  $F$  eine Zerlegung in Relationen  $R_1, \dots, R_n$  die folgende Kriterien erfüllt:
  - $R_1, \dots, R_n$  ist eine verlustlose Zerlegung von  $R$ .
  - Die Zerlegung ist abhängigkeiterhaltend.
  - Alle  $R_i$  ( $1 \leq i \leq n$ ) sind in dritter Normalform.



# Synthesealgorithmus für 3NF

Der Synthese-Algorithmus arbeitet in 4 Schritten:

1. Bestimme die kanonische Überdeckung  $F_c$  zu  $F$ , d.h. eine minimale Menge von FDs, die dieselben (partiellen und transitiven) Abhängigkeiten wie  $F$  beschreiben
2. Erzeuge neue Relationenschemata aus  $F_c$
3. Rekonstruiere einen Schlüsselkandidaten
4. Eliminiere überflüssige Relationen



# Synthesealgorithmus für 3NF

1. Bestimme die **kanonische Überdeckung**  $F_c$  zu einer gegebenen Menge  $F$  von funktionalen Abhängigkeiten (FDs):
  - a) Führe für jede FD  $X \rightarrow Y \in F$  die Linksreduktion durch, also:
    - Überprüfe für alle  $A \in X$ , ob  $A$  überflüssig ist, d.h. ob
$$Y \subseteq \text{AttrHülle}(F, X - A)$$
gilt. Falls dies der Fall ist, ersetze  $X \rightarrow Y$  durch  $(X - A) \rightarrow Y$ .
  - b) Führe für jede (verbliebene) FD  $X \rightarrow Y$  die Rechtsreduktion durch, also:
    - Überprüfe für alle  $B \in Y$ , ob
$$B \subseteq \text{AttrHülle}(F - (X \rightarrow Y) \cup (X \rightarrow (Y - B)), X)$$
gilt. In diesem Fall ist  $B$  auf der rechten Seite überflüssig und kann eliminiert werden, d.h.  $X \rightarrow Y$  wird durch  $X \rightarrow (Y - B)$  ersetzt.
  - c) Entferne die FDs der Form  $X \rightarrow \{\}$ , die in (b) möglicherweise entstanden sind.
  - d) Fasse FDs der Form  $X \rightarrow Y_1, \dots, X \rightarrow Y_n$  zusammen, so dass  $X \rightarrow (Y_1 \cup \dots \cup Y_n)$  verbleibt.



# Synthesealgorithmus für 3NF

2. Für jede FD  $X \rightarrow Y \in F_c$ 
  - Erzeuge ein Relationenschema  $R_X := X \cup Y$ .
  - Ordne  $R_X$  die FDs  $F_X := \{X' \rightarrow Y' \in F_c \mid X' \cup Y' \in R_X\}$  zu.
3. Rekonstruiere einen Schlüsselkandidaten
  - Falls eines der in Schritt 2 erzeugten Schemata einen Schlüsselkandidaten von  $R$  bzgl.  $F_c$  enthält, sind wir fertig.
  - Andernfalls wähle einen Schlüsselkandidaten  $S \subseteq R$  aus und definiere folgendes zusätzliche Schema:
    - $R_S := S$
    - $F_S := \{\}$
4. Eliminiere überflüssige Relationen
  - Eliminiere diejenigen Schemata  $R_X$ , die in einem anderen Relationenschema  $R_{X'}$  enthalten sind, d.h.
    - $R_X \subseteq R_{X'}$



# Synthesealgorithmus für 3NF

## Beispiel:

**Einkauf** (Anbieter, Ware, WGruppe, Kunde, KOrt, KLand, Kaufdatum)

### Schritte des Synthesealgorithmus:

1. Kanonische Überdeckung  $F_c$  der funktionalen Abhängigkeiten:

Kunde, WGruppe  $\rightarrow$  Anbieter

Anbieter  $\rightarrow$  WGruppe

Ware  $\rightarrow$  WGruppe

Kunde  $\rightarrow$  KOrt

KOrt  $\rightarrow$  KLand

2. Erzeugen der neuen Relationenschemata und ihrer FDs:

**Bezugsquelle** (Kunde, WGruppe, Anbieter) {Kunde, WGruppe  $\rightarrow$  Anbieter,  
Anbieter  $\rightarrow$  WGruppe}

**Lieferant** (Anbieter, WGruppe) {Anbieter  $\rightarrow$  WGruppe}

**Produkt** (Ware, WGruppe) {Ware  $\rightarrow$  WGruppe}

**Adresse** (Kunde, KOrt) {Kunde  $\rightarrow$  KOrt}

**Land** (KOrt, KLand) {KOrt  $\rightarrow$  KLand}

3. Da keine dieser Relationen einen Schlüsselkandidaten der ursprünglichen Relation enthält, muß noch eine eigene Relation mit dem ursprünglichen Schlüssel angelegt werden:

**Einkauf** (Ware, Kunde, Kaufdatum)

4. Da die Relation *Lieferant* in *Bezugsquelle* enthalten ist, können wir *Lieferant* wieder streichen.



# Boyce-Codd-Normalform

- Welche Abhängigkeiten können in der dritten Normalform noch auftreten?

Abhängigkeiten unter Attributen, die prim sind,  
aber noch nicht vollständig einen Schlüssel bilden

- Beispiel:

**Autoverzeichnis** (Hersteller, HerstellerNr, ModellNr)

- es gilt 1:1-Beziehung zw. Hersteller und HerstellerNr:

Hersteller  $\rightarrow$  HerstellerNr

HerstellerNr  $\rightarrow$  Hersteller

- Schlüsselkandidaten sind deshalb:

{Hersteller, ModellNr}

{HerstellerNr, ModellNr}

- Schema in 3. NF, da alle Attribute prim sind.



# Boyce-Codd-Normalform

- Trotzdem können auch hier Anomalien auftreten
- *Definition:*  
Ein Schema  $R$  ist in Boyce-Codd-Normalform, wenn für alle nichttrivialen Abhängigkeiten  $X \rightarrow Y$  gilt:
  - $X$  enthält einen Schlüsselkandidaten von  $R$
- Verlustlose Zerlegung ist generell immer möglich.
- Abhängigkeitserhaltende Zerlegung nicht immer möglich.



# Mehrwertige Abhängigkeiten (MVD)

- Mehrwertige Abhängigkeiten entstehen, wenn mehrere **unabhängige 1:n-Beziehungen** in einer Relation stehen (was nach Kapitel 6 eigentlich nicht sein darf):
- Mitarbeiter ( Name, Projekte, Verwandte )  
Huber, {P1, P2, P3} {Heinz, Hans, Hubert}  
Müller, {P2, P3} {Manfred}
- In erster Normalform müsste man mindestens 3 Tupel für Huber und 2 Tupel für Müller speichern:
- Mitarbeiter ( Name, Projekte, Verwandte )  
Huber, P1, Heinz,  
Huber, P2, Hans,  
Huber, P3, Hubert,  
Müller, P2, Manfred  
Müller, P3, NULL



# Mehrwertige Abhängigkeiten (MVD)

- Um die Anfrage zu ermöglichen, wer die Verwandten von Mitarbeitern in Projekt P2 sind, müssen pro Mitarbeiter sogar sämtliche Kombinationstupel gespeichert werden:

Mitarbeiter (Name, Projekte, Verwandte)

Huber,	P1,	Heinz,
Huber,	P1,	Hans,
Huber,	P1,	Hubert,
Huber,	P2,	Heinz,
Huber,	P2,	Hans,
Huber,	P2,	Hubert,
Huber,	P3,	Heinz,
Huber,	P3,	Hans,
Huber,	P3,	Hubert,
Müller,	P2,	Manfred,
Müller,	P3,	Manfred.

- Wir nennen dies eine Mehrwertige Abhängigkeit zwischen Name und Projekte (auch zwischen Name und Verwandte)



# Mehrwertige Abhängigkeiten (MVD)

Gegeben:  $X, Y, Z \subseteq R$ , mit  $X, Y$  disjunkt und  $Z = R - (X \cup Y)$ .  
 $Y$  ist *mehrwertig abhängig* von  $X$  ( $X \twoheadrightarrow Y$ ), wenn für jede gültige Ausprägung von  $R$  gilt:

- Für jedes Paar aus Tupeln  $t_1, t_2$  mit  $t_1.X = t_2.X$ , aber (natürlich)  $t_1 \neq t_2$ , existieren 2 weitere Tupel  $t_3$  und  $t_4$  mit folgenden Eigenschaften:

$$t_1.X = t_2.X = t_3.X = t_4.X$$

$$t_3.Y = t_1.Y$$

$$t_3.Z = t_2.Z$$

$$t_4.Y = t_2.Y$$

$$t_4.Z = t_1.Z$$

Jede FD ist auch eine MVD !!!



# Beispiel MVD

$R$			
	$X$ $A_1 \dots A_i$	$Y$ $A_{i+1} \dots A_j$	$Z$ $A_{j+1} \dots A_n$
$t_1$	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
$t_2$	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
$t_3$	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
$t_4$	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$



# Weiteres Beispiel

Relation: Modelle

ModellNr	Farbe	Leistung
E36	blau	170 PS
E36	schwarz	198 PS
E36	blau	198 PS
E36	schwarz	170 PS
E34	schwarz	170 PS

{ModellNr}  $\twoheadrightarrow$  {Farbe} und {ModellNr}  $\twoheadrightarrow$  {Leistung}

Farben

ModellNr	Farbe
E36	blau
E36	schwarz
E34	Schwarz

Leistung

ModellNr	Leistung
E36	170 PS
E36	198 PS
E34	170 PS

Modelle =  $\Pi_{\text{ModellNr, Sprache}}(\text{Farben}) \bowtie \Pi_{\text{ModellNr, Leistung}}(\text{Leistung})$



# Verlustlose Zerlegung MVD

Ein Relationenschema  $R$  mit einer Menge  $D$  von zugeordneten funktionalen mehrwertigen Abhängigkeiten kann genau dann verlustlos in die beiden Schemata  $R_1$  und  $R_2$  zerlegt werden, wenn gilt:

- $R = R_1 \cup R_2$
- mindestens eine von zwei MVDs gilt:
  1.  $R_1 \cap R_2 \twoheadrightarrow R_1$  oder
  2.  $R_1 \cap R_2 \twoheadrightarrow R_2$



# Triviale MVD und 4. Normalform

Eine MVD  $X \twoheadrightarrow Y$  bezogen auf  $R \supseteq X \cup Y$  ist *trivial*, wenn jede mögliche Ausprägung  $r$  von  $R$  diese MVD erfüllt. Man kann zeigen, dass  $X \twoheadrightarrow Y$  trivial ist, genau dann wenn:

- $Y \subseteq X$  oder
- $Y = R - X$ .

Eine Relation  $R$  mit zugeordneter Menge  $F$  von funktionalen und mehrwertigen Abhängigkeiten ist in 4. Normalform (**4NF**), wenn für jede MVD  $X \twoheadrightarrow Y \in F^+$  eine der folgenden Bedingungen gilt:

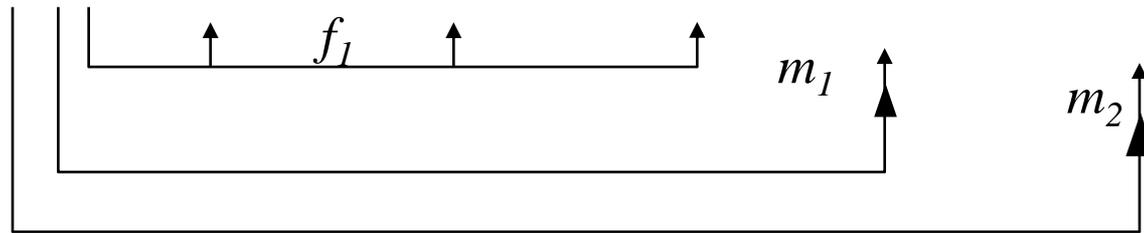
- Die MVD ist trivial oder
- $X$  ist ein Superschlüssel von  $R$ .



# Beispiel

Assistenten:

{[PersNr,Name,Fachgebiet, Boss,Sprache,ProgSprache]}



- Assistenten: {[PersNr,Name,Fachgebiet, Boss]}
- Sprachen: {[PersNr, Sprache]}
- ProgSprach: {[PersNr,ProgSprache]}



# Schlussbemerkungen

- Ein gut durchdachtes E/R-Diagramm liefert bereits weitgehend normalisierte Tabellen
- Normalisierung ist in gewisser Weise eine Alternative zum E/R-Diagramm
- Extrem-Ansatz: Universal Relation Assumption:
  - Modelliere alles zunächst in einer Tabelle
  - Ermittle die funktionalen Abhängigkeiten
  - Zerlege das Relationenschema entsprechend (der letzte Schritt kann auch automatisiert werden: Synthesealgorithmus für die 3. Normalform)



# Schlussbemerkungen

- Normalisierung kann schädlich für die Performanz sein, weil Joins sehr teuer auszuwerten sind
- Nicht *jede* FD berücksichtigen:
  - Abhängigkeiten zw. Wohnort, Vorwahl, Postleitzahl
  - Man kann SQL-Integritätsbedingungen formulieren, um Anomalien zu vermeiden (Trigger, siehe später)
- Aber es gibt auch Konzepte, Relationen so abzuspeichern, dass Join auf bestimmten Attributen unterstützt wird
  - ORACLE-Cluster



# Zusammenfassung

## Implikation

- 1. Normalform:  
Alle Attribute atomar
- 2. Normalform:  
Keine funktionale Abhängigkeit eines Nicht-Schlüssel-Attributs von **Teil** eines Schlüssels
- 3. Normalform:  
Zusätzlich keine nichttriviale funktionale Abhängigkeit eines Nicht-Schlüssel-Attributs von Nicht-Schlüssel-Attributen
- Boyce-Codd-Normalform:  
Zusätzlich keine nichttriviale funktionale Abhängigkeit unter den Schlüssel-Attributen
- 4. Normalform:  
keine Redundanz durch MVDs.