universität wien

VU Datenbanksysteme, Kapitel 6

Normalformen

Christian Böhm

Motivation

Beispiel-Relation:

Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis
	103	Qualle	555333	Schränkchen	020280	nachtblau	828,00
	103	Qualle	000123	Tisch	020299	nachtblau	599,00
	103	Qualle	000124	Stuhl	4040FF	himmelblau	129,90
	626	Bschiss	555333	Schränkchen	020280	nachtblau	835,00
	626	Bschiss	000138	Bett	020299	nachtblau	599,90

Redundanzen: für alle Tupel-Paare (t, t') gilt:

- Bei gleicher LNr → gleicher LName
- Bei gleicher WarenNr → gleiche Bezeichnung, Farbcode, Farbname
- Bei gleichem Farbcode → gleicher Farbname

Was ist Redundanz?

Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis
	103	Qualle	555333	Schränkchen	020280	nachtblau	828,00
	103	Qualle	000123	Tisch	020299	nachtblau	599,00
	103	Qualle	000124	Stuhl	4040FF	himmelblau	129,90
	626	Bschiss	555333	Schränkchen	020280	nachtblau	835,00
	626	Bschiss	000138	Bett	020299	nachtblau	599,90

Vermeidbare mehrfache Speicherung desselben Faktums

- Z.B. Lieferant 103 heißt Qualle (3x gespeichert)
- Speicherverschwendung!

Redundanzen führen zu Anomalien:

- Insert-Anomalie:
  z.B. INSERT INTO Angebot VALUES (103, 'Quäle', …)
- Update-Anomalie:
  z.B. UPDATE Angebot SET Farbname = 'rot' WHERE WarenNr = 000123
- Delete-Anomalie:
  z.B. Löschen der letzten Ware eines Lieferanten → Name geht verloren

Formalisierung der Redundanz: Funktionale Abhängigkeit (siehe später)

Was tun? Relation zerlegen!

Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis
	103	Qualle	555333	Schränkchen	020280	nachtblau	828,00
	103	Qualle	000123	Tisch	020299	nachtblau	599,00
	103	Qualle	000124	Stuhl	4040FF	himmelblau	129,90
	626	Bschiss	555333	Schränkchen	020280	nachtblau	835,00
	626	Bschiss	000138	Bett	020299	nachtblau	599,90

(Zerlegung)

Lieferant	LNr	LName
	103	Qualle
	626	Bschiss

Ware	WarenNr	Bezeichnung	Farbcode
	555333	Schränkchen	020280
	000123	Tisch	020299
	000124	Stuhl	4040FF
	000138	Bett	020299

Farben	Farbcode	Farbname
	020280	nachtblau
	020299	nachtblau
	4040FF	himmelblau

Preise	LNr	WarenNr	Preis
	103	555333	828,00
	103	000123	599,00
	103	000124	129,90
	626	555333	835,00
	626	000138	599,90

Ursprungsrelation als View: $\text{Angebot} = \text{Lieferant} \bowtie \text{Ware} \bowtie \text{Farben} \bowtie \text{Preise}$

Definition: Funktionale Abhängigkeit

Eine Attributmenge Y ist von X **funktional abhängig** $(X, Y \subseteq R)$

(Notation: $X \rightarrow Y$)

$\Leftrightarrow$

für jede gültige Ausprägung von R und jedes Paar (t, t') von Tupeln gilt:

$$t.X = t'.X \quad \Rightarrow \quad t.Y = t'.Y$$

(zu jeder Kombination von X-Werten existiert genau eine Kombination von Y-Werten)

Vergleich: Funktionale Abhängigkeit / Schlüssel

Wenn die funktionale Abhängigkeit $X \rightarrow Y$ gilt, dann gilt:

$$t.X = t'.X \quad \Rightarrow \quad t.Y = t'.Y$$

Ist X ein (Super-) **Schlüssel** von R, dann gilt die Eindeutigkeits-Eigenschaft:

$$t.X = t'.X \quad \Rightarrow \quad t = t'$$

(d.h. alle Werte gleich)

→ Alle Attribute der Relation sind von **jedem Schlüssel** funktional abhängig
→ Aber es gibt u.U. weitere (z.T. störende) funktionale Abhängigkeiten
→ Funktionale Abhängigkeit ist **Verallgemeinerung** des Schlüssel-Konzepts.

$$t.X = t'.X \quad \Rightarrow \quad t.Y = t'.Y$$

Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis
	103	Qualle	555333	Schränkchen	020280	nachtblau	828,00
	103	Qualle	000123	Tisch	020299	nachtblau	599,00
	103	Qualle	000124	Stuhl	4040FF	himmelblau	129,90
	626	Bschiss	555333	Schränkchen	020280	nachtblau	835,00
	626	Bschiss	000138	Bett	020299	nachtblau	599,90

Beispiele:

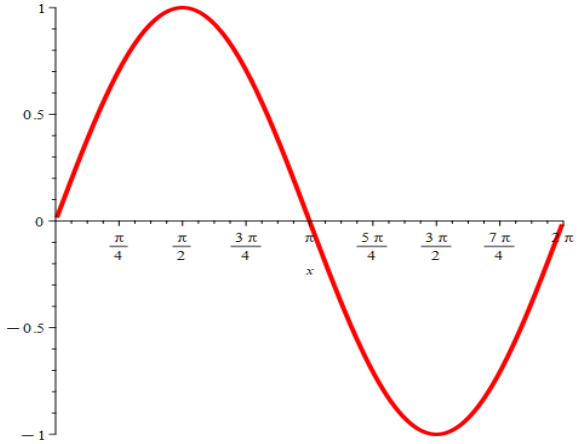
- Alle Attribute von jedem Schlüssel funktional abhängig:
  LNr, WarenNr → LNr, LName, WarenNr, Bezeichnung, Farbcode, Farbname, Preis
- Darüber hinaus funktionale Abhängigkeiten von *Teilen* eines Schlüssels:
  LNr → LName
  WarenNr → Bezeichnung, Farbcode, Farbname
- Funktionale Abhängigkeiten von Nicht-Schlüssel-Attributen:
  Farbcode → Farbname
- Und viele weitere:
  Preis → Preis
  LNr, Preis → LName
  LNr, Farbcode → Farbname
  …

  } Armstrong-Axiome (siehe später)

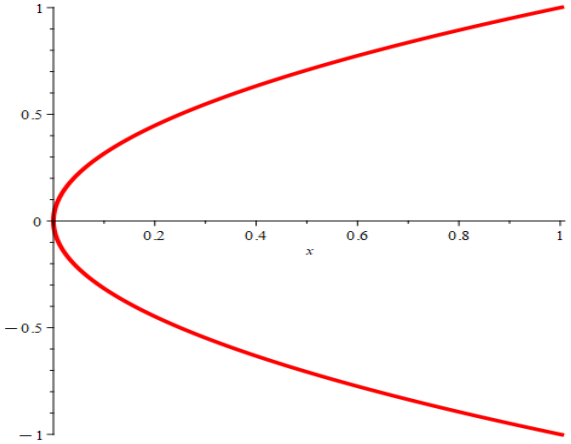
Warum „funktionale Abhängigkeit?“

- Namensgebend ist die Rechts-Eindeutigkeit einer Funktion $y = f(x)$:
  Jedem x-Wert ist eindeutig ein y-Wert zugeordnet.

Funktion:



Keine Funktion:



- Funktionale Abhängigkeit X $\rightarrow$ Y:
  Jedem X-Wert ist eindeutig ein Y-Wert zugeordnet.

Definition: Volle funktionale Abhängigkeit

Eine funktionale Abhängigkeit

$$X \rightarrow Y \qquad (X, Y \subseteq R)$$

heißt **volle** funktionale Abhängigkeit (Notation: $X \overset{\bullet}{\rightarrow} Y$)

$$\Leftrightarrow$$

$$\neg\exists\, X' \subsetneq X, \text{ so dass } X' \rightarrow Y$$

X ist **minimale Menge**: $X \rightarrow Y$

Andernfalls heißt $X \rightarrow Y$ **partielle** funktionale Abhängigkeit.

$$\neg\exists\ X' \subsetneq X, \text{ so dass } X' \rightarrow Y$$

Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis
	103	Qualle	555333	Schränkchen	020280	nachtblau	828,00
	103	Qualle	000123	Tisch	020299	nachtblau	599,00
	103	Qualle	000124	Stuhl	4040FF	himmelblau	129,90
	626	Bschiss	555333	Schränkchen	020280	nachtblau	835,00
	626	Bschiss	000138	Bett	020299	nachtblau	599,90

Beispiele:

(1) LNr $\overset{\bullet}{\rightarrow}$ LName — **voll** funktional abhängig:
Ein-elementige Mengen sind immer minimal

(2) LNr, WarenNr $\rightarrow$ LName — **partiell** funktional abhängig, wegen (1)

(3) LNr $\not\rightarrow$ Preis — **nicht** funktional abhängig

(4) WarenNr $\not\rightarrow$ Preis — **nicht** funktional abhängig

(5) LNr, WarenNr $\overset{\bullet}{\rightarrow}$ Preis — **voll** funktional abhängig, wegen (3), (4)

Volle Funktionale Abhängigkeit vs. Schlüssel

- Warum folgt aus der Minimalität eines Schlüssels S nicht, dass jedes Attribut **voll** funktional abhängig vom Schlüssel ist?

  Schlüssel S: → jedes Attribut $A \in R$ ist funktional abhängig von S
  → S ist die minimale Menge, von der alle Attribute funktional abhängig sind (zur Tupel-Identifikation).

- aber S ist nicht unbedingt minimal für jede einzelne funktionale Abhängigkeit $S \rightarrow A$ für jedes Attribut $A \in R$.
- z.B. kann es bei R(A, B, C, D) sein, dass gilt:
  $A \rightarrow B$ und $C \rightarrow D$.
  Schlüssel {A, C} ist für R minimal, aber es gibt partielle Abhängigkeiten.

Armstrong-Axiome

Reflexivität (R): $Y \subseteq X \Rightarrow X \rightarrow Y$ insbes.: $X \rightarrow X$

Verstärkung (V): $X \rightarrow Y \Rightarrow X \cup Z \rightarrow Y \cup Z$

Transitivität (T): $X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$

Die Armstrong-Axiome sind **korrekt** und **vollständig**.

(Beweis: Definition der funktionalen Abhängigkeit einsetzen!)

Symmetrie der funktionalen Abhängigkeit

Die funktionale Abhängigkeit erfüllt i.A. weder die Eigenschaft der Symmetrie noch die Eigenschaft der Anti-Symmetrie,

d.h. alle vier Fälle sind bei Attributmengen $X, Y \subseteq R$ möglich:

- $X \to Y$, aber nicht $Y \to X$
- $Y \to X$, aber nicht $X \to Y$
- sowohl $X \to Y$, als auch $Y \to X$
- weder $X \to Y$, noch $Y \to X$

Weitere nützliche Gesetze

Vereinigungsregel (VE): $X \to Y \ \wedge \ X \to Z \quad \Rightarrow \quad X \to YZ$

Dekompositionsregel (D): $X \to YZ \quad \Rightarrow \quad X \to Y, \ X \to Z$

Pseudo-Transitivität (P): $X \to Y \ \wedge \ YV \to Z \quad \Rightarrow \quad XV \to Z$

Hierbei steht YZ für $Y \cup Z$.
Diese Gesetze lassen sich aus den Armstrong-Axiomen herleiten.

Algorithmus AttrHülle

Gegeben: Menge F von funktionalen Abhängigkeiten,
Menge X von Attributen (Startmenge).

Welche Attribute X^+ sind nach Armstrong funktional abhängig von X?

algorithmus AttrHülle (F, X)
    $X^+ := X$; // Reflexivitätsaxiom
    while (Änderungen an X^+) **do**
        foreach $Y \rightarrow Z \in F$ **do** // funktionale Abhängigkeit $Y \rightarrow Z$ ist Iterator-Variable
            if ($Y \subseteq X^+$) **then** // ganze linke Seite von $Y \rightarrow Z$ schon in X^+ vorhanden
                $X^+ := X^+ \cup Z$; // rechte Seite von $Y \rightarrow Z$ zu X^+ dazunehmen

Algorithmus AttrHülle

```
algorithmus AttrHülle (F, X)
    X⁺ := X;
    while (Änderungen an X⁺) do
        foreach Y → Z ∈ F do
            if (Y ⊆ X⁺) then
                X⁺ := X⁺ ∪ Z;
```

Beispiel: Gegeben folgende Abhängigkeiten:

F = {

LNr, Preis	→	LName
Farbcode	→	Farbname
LNr, WarenNr	→	LNr, LName, WarenNr, Bezeichnung, Farbcode, Farbname, Preis
WarenNr	→	Bezeichnung, Farbcode
LNr	→	LName

}

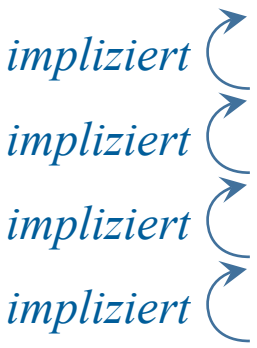
Ergebnis von AttrHülle(F, {WarenNr}) nach i-tem Durchlauf der Schleife:

0. X^+ = {WarenNr} // Initialisierung
1. X^+ = {WarenNr, Bezeichnung, Farbcode}
2. X^+ = {WarenNr, Bezeichnung, Farbcode, Farbname} // transitiv: WarenNr → Farbname
3. X^+ = {WarenNr, Bezeichnung, Farbcode, Farbname} // keine Änderung, fertig!

Normalisierung

Maxime: Beseitigung aller funktionaler Abhängigkeiten, außer es steht auf der linken Seite ein gesamter Schlüssel.

Verschiedene Normalformen beseitigen unterschiedliche Arten funktionaler Abhängigkeiten durch Zerlegung des Relationenschemas:

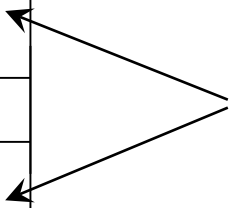


1. Normalform:	(immer gegeben)
2. Normalform:	partielle Abhängigkeiten
3. Normalform:	transitive Abhängigkeiten
Boyce-Codd-Normalform:	Abhängigkeiten zwischen Schlüsselkandidaten
4. Normalform:	mehrwertige Abhängigkeiten

Erste Normalform (1. NF)

- Keine Einschränkungen bezüglich funktionaler Abhängigkeiten
- Alle Attributwerte sind atomar, d.h. nicht mehrwertig
- In relationalen Datenbanken: mehrwertige Attribute nicht möglich
  → wir werden im weiteren stillschweigend 1. NF voraussetzen.
- Z.B. Zwischenergebnis von GROUP BY vor Aggregation:

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



Mehrwertig: C und D

Definition Zweite Normalform (2. NF):

Ein Relationen-Schema ist in 2. Normalform, wenn jedes Attribut A

- **voll** funktional abhängig von **allen** Schlüsselkandidaten ist

oder

- selbst Teil eines Schlüsselkandidaten (**prim**) ist.

Beobachtung: Die Zweite Normalform kann nicht verletzt sein, wenn…

- kein **zusammengesetzter** Schlüsselkandidat existiert
- oder alle Attribute prim sind.

Definition Zweite Normalform (2. NF):

Ein Relationen-Schema ist in 2. Normalform, wenn jedes Attribut A

- **voll** funktional abhängig von **allen** Schlüsselkandidaten ist

oder

- selbst Teil eines Schlüsselkandidaten (**prim**) ist.

Warum sollen Attribute nicht partiell von Schlüsselkandidaten abhängig sein?

- Partielle Abhängigkeiten von Schlüsselkandidaten verursachen Redundanz, z.B.:

  LNr → LName
  WarenNr → Bezeichnung, Farbcode, Farbname

- Jeder Schlüssel verursacht diese Redundanz → „von **allen** Schlüsselkandidaten“

Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis
	103	Qualle	555333	Schränkchen	020280	nachtblau	828,00
	103	Qualle	000123	Tisch	020299	nachtblau	599,00
	103	Qualle	000124	Stuhl	4040FF	himmelblau	129,90
	626	Bschiss	555333	Schränkchen	020280	nachtblau	835,00
	626	Bschiss	000138	Bett	020299	nachtblau	599,90

Definition Zweite Normalform (2. NF):

Ein Relationen-Schema ist in 2. Normalform, wenn jedes Attribut A

- **voll** funktional abhängig von **allen** Schlüsselkandidaten ist

oder

- selbst Teil eines Schlüsselkandidaten (**prim**) ist.

Warum sind **prime** Attribute bei der 2. (und 3.) Normalform ausgenommen?

- Partielle Abhängigkeiten zwischen **unterschiedlichen** Schlüsselkandidaten verursachen ebenfalls Redundanz,
- sind aber problematischer zu beseitigen.

→ Boyce-Codd-Normalform, siehe später!

Definition Zweite Normalform (2. NF):

Ein Relationen-Schema ist in 2. Normalform, wenn jedes Attribut A

- **voll** funktional abhängig von **allen** Schlüsselkandidaten ist

oder

- selbst Teil eines Schlüsselkandidaten (**prim**) ist.

LNr, WarenNr → Preis
LNr → LName
WarenNr → Bezeichnung, Farbcode, Farbname
Farbcode → Farbname (nicht relevant für 2. NF)

Um 2. Normalform herzustellen, zerlegen wir das Relationenschema:

- Attribute, die **voll** funktional abhängig von **allen** Schlüsseln sind, bleiben in der Relation

Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis

- Partiell abhängige Attribute werden aus dem Relationenschema gelöscht

Angebot	LNr	~~LName~~	WarenNr	~~Bezeichnung~~	~~Farbcode~~	~~Farbname~~	Preis

- Die partiellen Abhängigkeiten werden nach gleichen linken Seiten gruppiert

Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis

- Für jede Gruppe wird eine Relation erzeugt mit den Attributen der linken und rechten Seite (linke Seite wird Schlüssel)

(siehe nächste Folie)

Definition Zweite Normalform (2. NF):

Ein Relationen-Schema ist in 2. Normalform, wenn jedes Attribut A

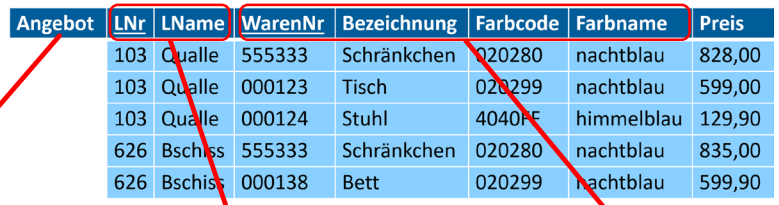
- **voll** funktional abhängig von **allen** Schlüsselkandidaten ist

oder

- selbst Teil eines Schlüsselkandidaten (**prim**) ist.

LNr, WarenNr → Preis
LNr → LName
WarenNr → Bezeichnung, Farbcode, Farbname
Farbcode → Farbname (nicht relevant für 2. NF)

- Für jede Gruppe wird eine Relation erzeugt mit den Attributen der linken und rechten Seite (linke Seite wird Schlüssel)



Angebot	LNr	LName	WarenNr	Bezeichnung	Farbcode	Farbname	Preis
	103	Qualle	555333	Schränkchen	020280	nachtblau	828,00
	103	Qualle	000123	Tisch	020299	nachtblau	599,00
	103	Qualle	000124	Stuhl	4040FF	himmelblau	129,90
	626	Bschiss	555333	Schränkchen	020280	nachtblau	835,00
	626	Bschiss	000138	Bett	020299	nachtblau	599,90

Angeb	LNr	WarenNr	Preis
	103	555333	828,00
	103	000123	599,00
	103	000124	129,90
	626	555333	835,00
	626	000138	599,90

Lieferant	LNr	LName
	103	Qualle
	626	Bschiss

Ware	WarenNr	Bezeichnung	Farbcode	Farbname
	555333	Schränkchen	020280	nachtblau
	000123	Tisch	020299	nachtblau
	000124	Stuhl	4040FF	himmelblau
	000138	Bett	020299	nachtblau

Definition Dritte Normalform (3. NF):

Ein Relationen-Schema ist in Dritter Normalform, wenn für jede funktionale Abhängigkeit $X \rightarrow A$ eine der folgenden Aussagen gilt:

(1) $X \rightarrow A$ ist **trivial**, d.h. $A \in X$

oder

(2) X ist ein Super-Schlüssel

oder

(3) A ist selbst Teil eines Schlüsselkandidaten (**prim**).

Definition Dritte Normalform (3. NF):

Ein Relationen-Schema ist in Dritter Normalform, wenn für jede funktionale Abhängigkeit $X \rightarrow A$ eine der folgenden Aussagen gilt:

(1) $X \rightarrow A$ ist **trivial**, d.h. $A \in X$

oder

(2) X ist ein Super-Schlüssel

oder

(3) A ist selbst Teil eines Schlüsselkandidaten (**prim**).

Intuition dieser Definition:

„Es gibt nur Abhängigkeiten, wo **links** ein **ganzer Schlüssel** steht“

Aber in der Definition steht der Allquantor „**für jede** funktionale Abhängigkeit“:

(1) Das Reflexivitätsaxiom erzeugt viele unvermeidbare („**triviale**“) Abhängigkeiten

(2) Wegen Verstärkungsaxiom sind alle Attribute auch von **Super**-Schlüssel abhängig

(3) Abhängigkeiten zwischen primen Attributen sind problematisch zu beseitigen

→ Boyce-Codd-Normalform, siehe später

Definition Dritte Normalform (3. NF):

Ein Relationen-Schema ist in Dritter Normalform, wenn für jede funktionale Abhängigkeit $X \rightarrow A$ eine der folgenden Aussagen gilt:

(1) $X \rightarrow A$ ist **trivial**, d.h. $A \in X$

oder

(2) X ist ein Super-Schlüssel

oder

(3) A ist selbst Teil eines Schlüsselkandidaten (**prim**).

Ware	WarenNr	Bezeichnung	Farbcode	Farbname
	555333	Schränkchen	020280	nachtblau
	000123	Tisch	020299	nachtblau
	000124	Stuhl	4040FF	himmelblau
	000138	Bett	020299	nachtblau

Die Dritte Normalform schließt aus:

- **Partielle** Abhängigkeiten von Schlüsselkandidaten
  Nach (2) muss linke Seite X ein **ganzer Superschlüssel** sein (außer A ist prim/trivial)

  → Zweite Normalform ist in dieser Definition mit impliziert

- Abhängigkeiten zwischen **nicht-primen** Attributen,
  ebenfalls nach (2). Diese erzeugen ebenfalls Redundanzen (siehe Beispiel)
  Diese Abhängigkeiten heißen missverständlich auch „transitive Abhängigkeiten“

Definition Dritte Normalform (3. NF):

Ein Relationen-Schema ist in Dritter Normalform, wenn für jede funktionale Abhängigkeit $X \rightarrow A$ eine der folgenden Aussagen gilt:

(1) $X \rightarrow A$ ist **trivial**, d.h. $A \in X$

oder

(2) X ist ein Super-Schlüssel

oder

(3) A ist selbst Teil eines Schlüsselkandidaten (**prim**).

Ware	WarenNr	Bezeichnung	Farbcode
	555333	Schränkchen	020280
	000123	Tisch	020299
	000124	Stuhl	4040FF
	000138	Bett	020299

Farben	Farbcode	Farbname
	020280	nachtblau
	020299	nachtblau
	4040FF	himmelblau

Zerlegungsalgorithmus für Dritte Normalform:

Ausgehend von und analog zur 2. NF (ohne partielle Abhängigkeiten):

- **Nicht-prime** Attribute werden aus dem Relationenschema gelöscht, wenn sie von **nicht-primen** Attributen abhängig sind.*
- Die Abhängigkeiten werden nach gleichen linken Seiten gruppiert.
- Für jede Gruppe wird eine Relation erzeugt mit den Attributen der linken und rechten Seite (linke Seite wird Schlüssel).

*mit Ausnahme der trivialen Abhängigkeiten

Kanonische Überdeckung

Ziel: Minimierung der Menge F von funktionalen Abhängigkeiten auf einen Kern ohne überflüssige Elemente
(die sich durch die Armstrong-Axiome bzw. AttrHülle ergeben).

Links-Reduktion: Überprüfe jedes Attribut A auf jeder linken Seite einer jeden funktionalen Abhängigkeit in F, ob es sich löschen lässt, ohne dass funkt. Abhängigkeiten

dazukommen

Rechts-Reduktion: Überprüfe jedes Attribut B auf jeder rechten Seite, ob es sich löschen lässt, ohne dass Abhängigkeiten

wegfallen

Links-Reduktion

Ein Attribut $A \in X$ (= linke Seite) der funktionalen Abhängigkeit

$$X \rightarrow Y \quad \in \quad F$$

kann gelöscht werden, wenn Y (z.B. transitiv) durch die funktionalen Abhängigkeiten im **bisherigen** F ohnehin von $X \setminus \{A\}$ abhängig ist.

Also, wenn: $Y \subseteq \text{AttrHülle}(F, X \setminus \{A\})$

Ggf. wird A aus $X \rightarrow Y$ in der Menge F gelöscht:

$$F := F \setminus \{X \rightarrow Y\} \cup \{X \setminus \{A\} \rightarrow Y\}$$

Rechts-Reduktion

Ein Attribut $B \in Y$ (= rechte Seite) der funktionalen Abhängigkeit

$$X \rightarrow Y \quad \in \quad F$$

kann gelöscht werden, wenn B auch durch funktionale Abhängigkeiten im **neuen** F' (nach dem Löschen von B) von X funktional abhängig ist.

Wir löschen B probeweise:

$$F' := F \setminus \{X \rightarrow Y\} \cup \{X \rightarrow Y \setminus \{B\}\}$$

Für die Entscheidung, F' beizubehalten überprüfen wir:

$$B \in \text{AttrHülle}\,(F', X) \qquad \Rightarrow F := F';$$

Synthese-Algorithmus für Dritte Normalform

(1) Ermittle die **kanonische Überdeckung** F_c der funkt. Abh. F von R:
   (a) Führe alle möglichen Links-Reduktionen durch;
   (b) Führe alle möglichen Rechts-Reduktionen durch;
   (c) Entferne funktionale Abhängigkeiten $X \rightarrow \{\}$;
   (d) Vereinige funktionale Abhängigkeiten mit gleicher linker Seite;

(2) Für jede funktionale Abhängigkeit $X \rightarrow Y$ in F_c:
Erzeuge Relation $R_X := X \cup Y$; (X wird Schlüssel von R_X);

(3) Falls keine Relation R_X aus Schritt (2) einen Schlüssel von R enthält:
Erzeuge eine Relation R_S mit einem Schlüsselkandidaten S von R;

(4) Lösche R_X wenn Teilmenge einer anderen Relation $R_{X'}$: $\underbrace{X \cup Y}_{R_X} \subseteq \underbrace{X' \cup Y'}_{R_{X'}}$.

Beispiel

(a) Führe alle möglichen Links-Reduktionen durch;
(b) Führe alle möglichen Rechts-Reduktionen durch;
(c) Entferne funktionale Abhängigkeiten $X \rightarrow \{\}$;
(d) Vereinige funktionale Abhängigkeiten mit gleicher linker Seite;

Wir betrachten die folgenden, gültigen funktionalen Abhängigkeiten:

(1) LNr, WarenNr → LName, Bezeichnung, Farbcode, Farbname, Preis
(2) LNr → LName
(3) WarenNr → Bezeichnung, Farbcode, Farbname
(4) WarenNr, Preis → Farbcode
(5) Farbcode → Farbname

Links-Reduktionen:

(4) WarenNr, ~~Preis~~ → Farbcode wegen (3)

Rechts-Reduktionen:

(1) LNr, WarenNr → ~~LName~~, Bezeichnung, Farbcode, Farbname, Preis wegen (2)
(1) LNr, WarenNr → ~~Bezeichnung~~, ~~Farbcode~~, ~~Farbname~~, Preis wegen (3)
(3) WarenNr → Bezeichnung, Farbcode, ~~Farbname~~ wegen (5)
(4) WarenNr → ~~Farbcode~~ wegen (3)

Beispiel

(a) Führe alle möglichen Links-Reduktionen durch;
(b) Führe alle möglichen Rechts-Reduktionen durch;
(c) Entferne funktionale Abhängigkeiten $X \rightarrow \{\}$;
(d) Vereinige funktionale Abhängigkeiten mit gleicher linker Seite;

Nach Schritt (b) haben wir folgende funktionale Abhängigkeiten:

(1) LNr, WarenNr $\rightarrow$ Preis
(2) LNr $\rightarrow$ LName
(3) WarenNr $\rightarrow$ Bezeichnung, Farbcode
(4) WarenNr $\rightarrow$ {}
(5) Farbcode $\rightarrow$ Farbname

In Schritt (c) wird (4) eliminiert:

(4) ~~WarenNr $\rightarrow$ {}~~

In Schritt (d) ist nichts zu tun.

Beispiel

(2) Für jede funktionale Abhängigkeit $X \rightarrow Y$ in F_c:
Erzeuge Relation $R_X := X \cup Y$; (X wird Schlüssel von R_X);
(3) Falls keine Relation R_X aus Schritt (2) einen Schlüssel von R enthält:
Erzeuge eine Relation R_S mit einem Schlüsselkandidaten S von R;
(4) Lösche R_X wenn Teilmenge einer anderen Relation $R_{X'}$

Die kanonische Überdeckung F_c enthält jetzt folgende funktionale Abhängigkeiten:

(1) LNr, WarenNr → Preis
(2) LNr → LName
(3) WarenNr → Bezeichnung, Farbcode
(5) Farbcode → Farbname

In Schritt (2) wird für jede Abhängigkeit eine Relation erzeugt:

R_1 (LNr, WarenNr, Preis)
R_2 (LNr, LName)
R_3 (WarenNr, Bezeichnung, Farbcode)
R_5 (Farbcode, Farbname)

In Schritt (3) ist nichts zu tun, da R_1 bereits den Schlüssel (LNr, WarenNr) enthält.

In Schritt (4) ist nichts zu tun, da keine Relation Teilmenge einer anderen ist.

Das Ergebnis des Synthesealgorithmus ist…

- **Verlustlos** (oder auch: *Verbund-treu*):

$$R = R_1 \bowtie R_2 \bowtie \ldots$$

  Beweis-Idee: Für (R_i, R_j) gilt immer: $(R_i \cap R_j) \rightarrow R_i$ **oder** $(R_i \cap R_j) \rightarrow R_j$

- **Abhängigkeitserhaltend**:

  Für jede Abhängigkeit $X \rightarrow Y$: $\exists$ Relation R_i: $X, Y \subseteq R_i$

  d.h. jede funktionale Abhängigkeit ist einer Relation zugeordnet.
  Beweis-Idee: Algorithmus erzeugt jede Relation aus einer Abhängigkeit.

- In **Dritter Normalform**.

Boyce-Codd-Normalform

Welche funktionalen Abhängigkeiten können in 3. NF noch auftreten?

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

Beispiel: Autoverzeichnis (<u>Hersteller</u>, HerstellerNr, <u>ModellNr</u>)

mit Hersteller $\rightarrow$ HerstellerNr
und HerstellerNr $\rightarrow$ Hersteller

} d.h. 1:1-Beziehung

Schlüsselkandidaten: {Hersteller, ModellNr},
{HerstellerNr, ModellNr}.

Schema ist in **Dritter Normalform**, weil alle Attribute prim sind.

Definition Boyce-Codd-Normalform (BCNF):

Ein Relationen-Schema ist in Boyce-Codd-Normalform, wenn für jede funktionale Abhängigkeit $X \rightarrow A$ eine der folgenden Aussagen gilt:

(1) $X \rightarrow A$ ist **trivial**, d.h. $A \in X$

oder

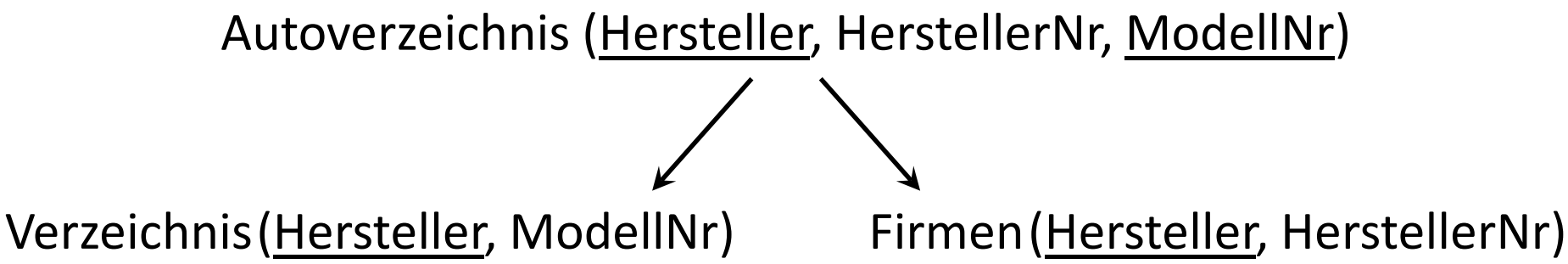
(2) X ist ein Super-Schlüssel

Herstellen der Boyce-Codd-Normalform

Zerlegungsalgorithmus analog zur/ausgehend von 2./3. Normalform:

- Löschen der partiell abhängigen, primen Attribute aus dem Relationenschema
- Gemeinsam mit der zugehörigen linken Seite: Erzeuge neue Relation!

Beispiel:



→ Zerlegung verlustlos, aber nicht immer Abhängigkeits-erhaltend

ggf. Verschlechterung der Situation: Abhängigkeiten nicht mehr überprüfbar ohne Join

Fazit

- Gut durchdachtes E/R-Diagramm → weitgehend normalisierte Tabellen
- Normalformen sind Alternative und Ergänzung zum E/R-Modell
- Extrem-Ansatz ohne E/R-Modell: Universal Relation Assumption:
  - Alles in einer Tabelle modellieren
  - funktionale Abhängigkeiten ermitteln
  - Synthesealgorithmus.

Normalisierung kann schädlich für Performanz sein (Join)

- Nicht jede funktionale Abhängigkeit berücksichtigen (Postleitzahl, …)
- Man kann auch komplexe Integritätsbedingungen in SQL formulieren
- Man kann SQL-Tabellen so abspeichern, dass Join unterstützt wird (Cluster)