Informationssyteme[1]

Kapitel 6: Normalisierung

Wintersemester 2020/21

Prof. Dr. Peer Kröger
Institut für Informatik
Arbeitsgruppe Informationssysteme und Data Mining

[1] Die folgenden Folien basieren in großen Teilen auf Material zur Vorlesung "Datenbanksysteme", die ich in meiner Zeit an der LMU München mehrmals gehalten habe. Das Material ist von den damaligen Kollegen maßgeblich (mit-)gestaltet worden, insbes. von Prof. Dr. C. Böhm.

Übersicht

1. Einleitung
2. Funktionale Abhängigkeit
3. Normalformen

Agenda

1. Einleitung
2. Funktionale Abhängigkeit
3. Normalformen

Road Map

- Zur Erinnerung: Sequentielles Vorgehen zur Implementierung einer Datenbank:

Schritt 1:	Informelle Beschreibung	$\rightarrow$	Pflichtenheft
Schritt 2:	Konzeptioneller Entwurf	$\rightarrow$	E/R-Diagramm
Schritt 3:	Relationaler DB-Entwurf	$\rightarrow$	Relationenschema

- Wir kümmern uns nun endlich um die Frage von Kapitel 3 nach dem schlechten Design der Lieferanten-Tabelle
- Zentrale Fragestellung in diesem Kapitel: gibt es eine Bewertungsgrundlage zur Unterscheidung zwischen einem *guten* und einem *schlechten* relationalen Datenmodell?
- Antwort: Normalisierungstheorie als formale Grundlage für den relationalen DB-Entwurf

Motivation

- Gegeben sei folgendes Relationionenschema (mit entsprechender Beispiel-Ausprägung):

Lieferant

LNr	**LName**	**LStadt**	**LPLZ**	**Produkt**	**Preis**
103	Huber	Berlin	10...	Gabel	18
103	Huber	Berlin	10...	Messer	12
103	Huber	Berlin	10...	Löffel	15
104	Berger	Berlin	10...	Löffel	21
⋮					
752	Müller	München	80...	Löffel	5
752	Müller	München	80...	Messer	6

- Warum ist dies ein schlechtes Design?

Motivation

Lieferant (wie vorher, Tupelreihenfolge verändert)

LNr	**LName**	**LStadt**	**LPLZ**	**Produkt**	**Preis**
103	Huber	Berlin	10...	Gabel	18
104	Berger	Berlin	10...	Löffel	21
			⋮		
103	Huber	Berlin	10...	Löffel	15
			⋮		
752	Müller	München	80...	Löffel	5
103	Huber	Berlin	10...	Messer	12
752	Müller	München	80...	Messer	6

- Ein mögliches Problem ergibt sich u.a. beim Ändern der Adresse eines Lieferanten (z.B. Huber): werden nicht alle Adress-Einträge geändert erhält man einen inkonsistenten Datenbestand

Problem: Redundanz

Lieferant

LNr	**LName**	**LStadt**	**LPLZ**	**Produkt**	**Preis**
103	Huber	Berlin	10...	Gabel	18
103	Huber	Berlin	10...	Messer	12
103	Huber	Berlin	10...	Löffel	15
104	Berger	Berlin	10...	Löffel	21
⋮					
752	Müller	München	80...	Löffel	5
752	Müller	München	80...	Messer	6

Probleme ergeben sich grundsätzlich aus Redundanzen in Daten:

- Für jedes Tupel-Paar gilt: sind die Werte in LNr gleich, so sind die Werte in LName, LStadt und LPLZ ebenfalls identisch
- Eine ähnliche Abhängigkeit besteht zwischen LPLZ und LStadt (sogar, wenn LNr unterschiedlich ist)

Funktionale Abhängigkeit (FA)

- Diese Abhängigkeits-Beziehung zwischen Attributsmengen werden Funktionale Abhängigkeit (FA) genannt
- LStadt hängt funktional von LPLZ ab
- LName, LStadt und LPLZ sind funktional abhängig von LNr
- FAs können Redundanzen im Datenbestand erzeugen, die wiederum zu Problemen im Betrieb führen können
- Ein gutes Datenmodell vermeidet Redundanzen (indem FAs vermieden werden)

Probleme durch Redundanzen

- Redundante Daten verschwenden Speicherplatz (echt jetzt?!?)
- Das eigentliche Problem ist, dass Redundanzen sog. Anomalien verursachen können:
  - Update-Anomalie: Änderung der Adresse (LName, LStadt, LPLZ) in nur einem statt in allen Tupeln mit identischer LNr
  - Insert-Anomalie: Einfügen eines Lieferanten mit inkonsistenter Adresse; Einfügen eines Lieferanten erfordert Ware
  - Delete-Anomalie: Löschen der letzten Ware eines Lieferanten löscht dessen Adresse

Verbesserung durch Zerlegung

- Ursprüngliches Datenbankschema:

  Lieferant (**LNr**, LName, LStadt, LPLZ, **Produkt**, Preis)

- Neues Datenbankschema

  LieferantAdr (**LNr**, LName, LPLZ)
  StadtPLZ (**LPLZ**, LStadt)
  Angebot (**LNr**, **Produkt**, Preis)

- Dieses Schema ist intuitiver und hat keine Redundanz (und damit auch keine Anomalien)
- Diese Zerlegung nennt man *Normalisierung*

Ursprüngliche Relation nach Zerlegung

- Nachteil der Zerlegung:
  Um zu einer Ware die Städte der potentiellen Lieferanten zu finden, ist ein zweifacher Join nötig!
- Natürlich könnte man die ursprüngliche Relation mit Hilfe einer View simulieren

```
CREATE VIEW Lieferant AS
   SELECT L.LNr, L.LName, L.LStadt, S.LPLZ,
          A.Produkt, A.Preis
   FROM LieferantAdr L, StadtPLZ S, Angebot A
   WHERE L.LNr = A.LNr AND L.LPLZ = S.LPLZ
```

  aber um den Join kommen wir nicht herum und die View ist auch *nicht updateable* (Wozu dann die View?)

Schema-Zerlegung (Normalisierung)

- Anomalien entstehen durch Redundanzen
- Entwurfsziel:
  Vermeidung von Redundanzen (evtl. unter Einbeziehung von Effizienzüberlegungen)
- Vorgehen: Normalisierung
  Schrittweises Zerlegen des Schemas (aus dem Entwurf) in ein äquivalentes Schema ohne Redundanz
  (Extremfall *Universal-Relation-Assumption*: alles wird zunächst in einer Relation modelliert und anschließend durch Zerlegung in ein redundanzfreies Schema überführt[2])
- Formalisierung von Redundanz durch das Konzept der FAs

[2] Dafür gibt es sogar Algorithmen — siehe später

Agenda

1. Einleitung
2. Funktionale Abhängigkeit
3. Normalformen

Konventionen und Schreibweisen

- Wir verwenden im Rest dieses Kapitels folgende Notation:

  A, B, C bezeichnen einzelne Attribute

  X, Y, Z bezeichnen Mengen von Attributen

- Zur Vereinfachung gilt außerdem:

  $A, B \to C$ bezeichne $\{A, B\} \to \{C\}$

  $X \to Y, Z$ bezeichne $X \to Y \cup Z$

  $t.A$ bezeichne das Attribut A des Tupels t

  $t.X$ bezeichne die Menge der Attribute X des Tupels t

  $t.X = r.X$ bezeichne $\forall A \in X : t.A = r.A$

Funktionale Abhängigkeit: Definition

Definition (Funktionale Abhängigkeit, kurz: FA)

Seien X und Y zwei Mengen von Attributen eines Relationenschemas R ($X, Y \subseteq R$). Y ist von X funktional abhängig, geschrieben $X \rightarrow Y$, gdw. für alle Tupel t und r gilt:

$$t.X = r.X \Rightarrow t.Y = r.Y$$

Intuition:

- Für alle möglichen Ausprägungen von R gilt: zu jedem Wert von X gibt es genau einen Wert von Y

Funktionale Abhängigkeit

Nochmal die Definition:

$X \to Y$, gdw. für alle Tupel t und r gilt: $t.X = r.X \Rightarrow t.Y = r.Y$

Beispiel: Funktionale Abhängigkeiten

- Schema:
  Lieferant (**LNr**, LName, LStadt, LPLZ, **Produkt**, Preis)
- Funktionale Abhängigkeiten:
  - LNr → LName
  - LNr → LPLZ
  - LPLZ → LStadt
  - LNr, Produkt → LName
  - LNr, Produkt → LPLZ
  - LNr, Produkt → Preis

FA: Generalisierung des Schlüssel-Konzepts

- Die Definition der FA und die Definition der Eindeutigkeit des Schlüssels sehen sehr ähnlich aus.
- Tatsächlich:
  - Für jeden Schlüsselkandidaten S eines Relationenschemas R folgt aus der Eindeutigkeit von S:
    Alle Attribute von R sind funktional abhängig von S ($S \rightarrow R$)
  - Diese FAs sind gewollt (das ist der Zweck von Schlüsseln)
- Aber: es kann weitere FAs geben, z.B. kann ein Attribut B abhängig sein von
  - Nicht-Schlüssel-Attributen (z.B. LStadt von LPLZ)
  - nur einem Teil eines Schlüssel(kandidaten) (z.B. LName von LNr)

  Diese FAs wollen wir eliminieren!

FA: Generalisierung des Schlüssel-Konzepts

- Wie die Schlüsseleigenschaft ist auch die funktionale Abhängigkeit eine *semantische* Eigenschaft des Schemas
- Eine FA ist also nicht aus einer aktuellen Ausprägung entscheidbar sondern muss für alle möglichen Ausprägungen gelten

Definition (prim)

Ein Attribut heißt prim, wenn es Teil eines Schlüsselkandidaten ist

Volle und partielle FA

- Ist ein Attribut Y funktional abhängig von X ($X \to Y$), dann auch von jeder Obermenge von X
- Man interessiert sich meist für die minimale Menge, von der Y abhängt (ähnlich wie die Minimalität des Schlüssels)

Definition (Volle/partielle FA)

Sei $X \to Y$. Wenn es keine echte Teilmenge $\hat{X} \subset X$ gibt, von der Y ebenfalls funktional abhängt, so ist $X \to Y$ eine volle funktionale Abhängigkeit, andernfalls eine partielle funktionale Abhängigkeit.

Volle und partielle FA

Beispiel: Volle/partielle FA

Lieferant (**LNr**, LName, LStadt, LPLZ, **Produkt**, Preis)

- LNR $\rightarrow$ LName — voll funktional abhängig
- LNr, Produkt $\rightarrow$ LName — partiell funktional abhängig
- LNr, Produkt $\rightarrow$ Preis — voll funktional abhängig

Volle FA und die Minimalität des Schlüssels

- Wiederum sehr ähnliche Definitionen:
  - Schlüssel: minimale Menge, die Eindeutigkeit erfüllt
  - Volle FA: minimale Menge, von der Attribut abhängt
  - Eindeutigkeit ist äquivalent zu FA (siehe oben)
- Aber Vorsicht: aus der Minimalität des Schlüssels folgt leider nicht, dass alle Attribute (einzeln für sich) immer voll funktional abhängig vom Schlüssel sein müssen
- Diese Schlussfolgerung gilt nur für das gesamte Tupel (also alle Attribute zusammen sind immer voll funktional abh. vom Schlüssel)

Funktionale Abhängigkeit: Eigenschaften

- Amstrong Axiome
  - Reflexivität:
    Falls $Y \subseteq X$, dann gilt immer auch $X \to Y$
    Insbesondere gilt also auch immer die triviale FA $X \to X$ (von sich selber) sowie $X \cup Z \to X$ (von jeder Obermenge)
  - Verstärkung (Komposition):
    Falls $X \to Y$, dann gilt immer auch $X \cup Z \to Y \cup Z$
  - Transitivität:
    Falls $X \to Y$ und $Y \to Z$, dann gilt immer auch $X \to Z$
- Diese Axiome sind vollständig und korrekt, d.h. für eine Menge F von FDs
  - lassen sich aus F nur FDs ableiten, die von jeder Relationen-Ausprägung erfüllt werden, für die auch F erfüllt ist
  - sind alle FDs ableitbar, die durch F impliziert sind
- Achtung: es gilt keine Symmetrie oder Antisymmetrie

Funktionale Abhängigkeit: Eigenschaften

- Zusätzliche Eigenschaften (folgen aus den Amstrong Axiomen)
  - Vereinigung:
    Falls $X \to Y$ und $X \to Z$, dann gilt immer auch $X \to Y \cup Z$
  - De-Komposition:
    Falls $X \to Y \cup Z$, dann gilt immer auch $X \to Y$ und $X \to Z$
  - Pseudo-Transitivität:
    Falls $X \to Y$ und $Z \cup Y \to V$, dann gilt immer auch $X \cup Z \to V$

Funktionale Abhängigkeit: Hülle einer Attributmenge

Definition (Hülle einer Attributmenge)

Für eine Menge an Attributen X und eine Menge von FAs F ist die Hülle $\mathcal{H}_F(X)$ die Menge aller von X funktional abhängigen Attribute bzgl. F, d.h. es gilt $X \rightarrow \mathcal{H}_F(X)$

- Algorithmus zur Berechnung von $\mathcal{H}_F(X)$:
  Input: Attributmenge $X \subseteq R$, Menge aller FAs F von R

  Initialisierung: $\mathcal{H}_F(X) \leftarrow X$.
  Solange sich $\mathcal{H}_F(X)$ verändert:
  Für jede FA $Y \rightarrow Z \in F$:
  Wenn $Y \subseteq \mathcal{H}_F(X)$ dann $\mathcal{H}_F(X) \leftarrow \mathcal{H}_F(X) \cup Z$.
  Ergebnis: $\mathcal{H}_F(X)$.

Funktionale Abhängigkeit: Hülle einer Attributmenge

Beispiel: Hülle einer Attributmenge

- $F = \{$ LNr $\rightarrow$ LName. LNr $\rightarrow$ LPLZ. LPLZ $\rightarrow$ LStadt. LNr, Produkt $\rightarrow$ Preis $\}$
- $X = \{$ LNr $\}$
- Zustand von $\mathcal{H}_F(X)$ nach dem i-ten Durchlauf der While-Schleife ("Solange ..."):
  - 0. Durchlauf: $\mathcal{H}_F(X) = \{$ LNr $\}$
  - 1. Durchlauf: $\mathcal{H}_F(X) = \{$ LNr, LName, LPLZ $\}$
  - 2. Durchlauf: $\mathcal{H}_F(X) = \{$ LNr, LName, LPLZ, LStadt $\}$
  - 3. Durchlauf: $\mathcal{H}_F(X) = \{$ LNr, LName, LPLZ, LStadt $\}$ STOP

Agenda

1. Einleitung
2. Funktionale Abhängigkeit
3. Normalformen

Normalisierung

- In einem Relationenschema sollen also möglichst keine funktionalen Abhängigkeiten bestehen, außer vom gesamten Schlüssel (und allen weiteren Schlüsselkandidaten)
- Verschieden Normalformen (NF) beseitigen unterschiedliche Arten von funktionaler Abhängigkeit
- Die NF sind nummeriert, in der Praxis spielen die ersten drei Normalformen die größte Rolle (ein Schema, das die 3. Normalform erfüllt heißt auch 3NF-Schema)
- Die k-te NF impliziert dabei jede l-te ($l < k$) NF
- Eine spezielle NF kann durch sog *verlustlose* Zerlegung des Schemas erreicht werden

1. Normalform

- Fordert zunächst keine Einschränkungen bzgl. FAs

Definition (1. Normalform)

Ein Relationenschema ist in erster Normalform, wenn alle Attributwerte atomar sind.

- Nicht-atomare Attribute ("nested relation", "non-first normal form (NFNF)") sind in relationalen DB ohnehin nicht möglich (außer als Zwischenergebnis einer Gruppierung in SQL durch **group by** Klausel)

2. Normalform

- Verhindere, dass Attribute nur partiell vom gesamten Schlüssel(kandidaten) abhängig sind

Beispiel: Redundanz ohne 2. NF

Lieferant

LNr	**LName**	**LStadt**	**LPLZ**	**Produkt**	**Preis**
103	Huber	Berlin	10...	Gabel	18
103	Huber	Berlin	10...	Messer	12
103	Huber	Berlin	10...	Löffel	15
			⋮		

- LName ist nur von LNr abhängig und damit nur partiell vom gesamten Schlüssel
- Redundanz: die Information in LName muss immer wiederholt werden

2. Normalform

Definition (2. Normalform)

Ein Relationenschema ist in zweiter Normalform (2NF), wenn jedes Attribut

- voll funktional abhängig von allen Schlüsselkandidaten ist, oder
- selbst prim[3] ist.

Beobachtung: die 2. NF kann nur verletzt sein, wenn ...

- ... ein zusammengesetzter Schlüssel(kandidat) existiert
- ... nicht-prime Attribute existieren

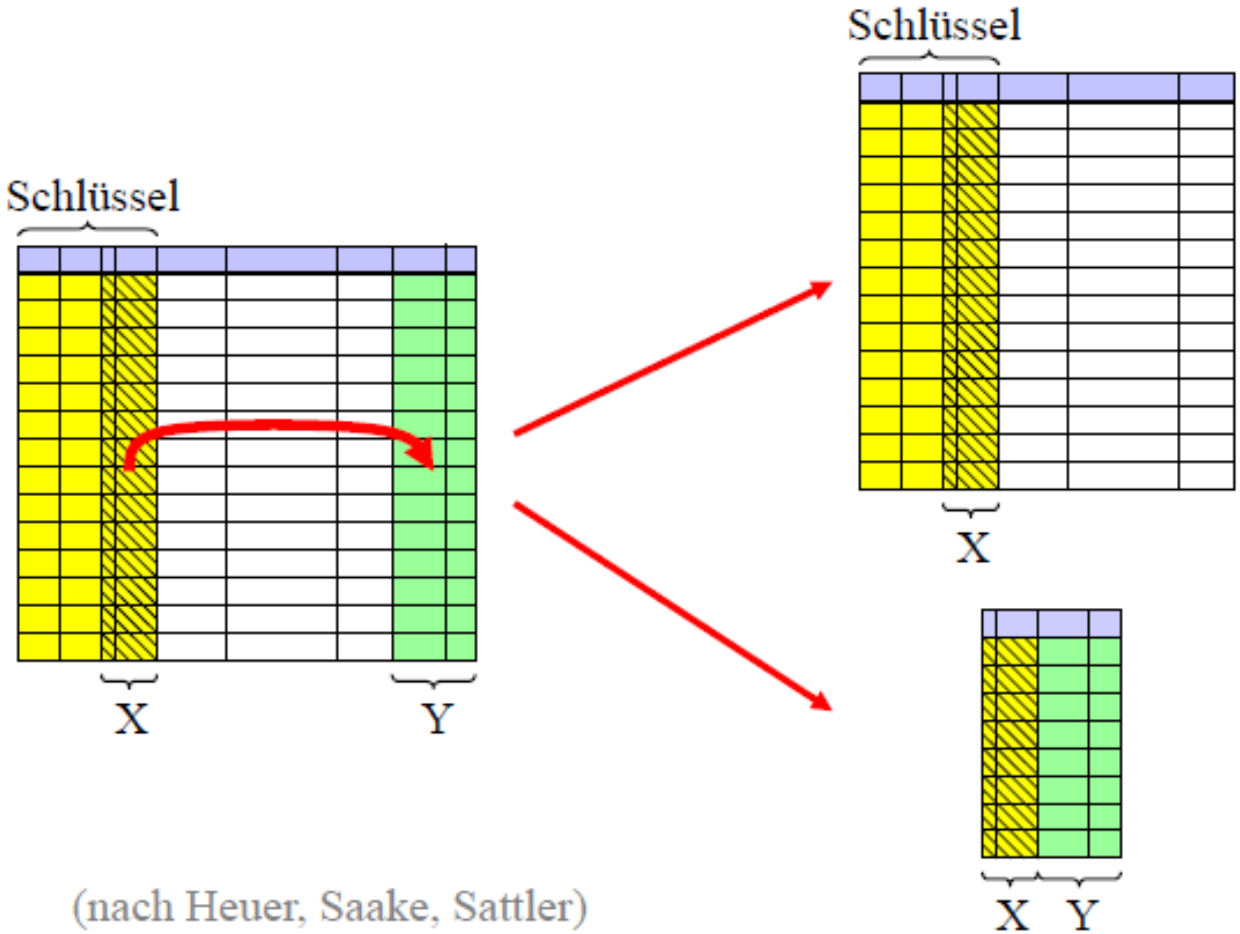
[3] Zur Erinnerung: prim = "Teil eines Schlüsselkandidaten"

2. Normalform

- Transformation eines Relationenschema R in 2NF durch folgende Zerlegung
  - Attribute, die voll funktional abhängig vom Schlüssel sind, bleiben in R
  - Für alle FAs $X_i \rightarrow Y_i$ von einem Teil des Schlüssels S ($X_i \subset S$):
    - Lösche die Attribute Y_i aus R
    - Gruppiere die FAs nach gleichen linken Seiten X_l
    - Für jede dieser Gruppen, führe eine neue Relation ein mit allen enthaltenen Attributen aus X_l und Y_l
    - X_l wird Schlüssel in der neuen Relation

2. Normalform

- Graphisch:



(nach Heuer, Saake, Sattler)

2. Normalform

Beispiel: 2. Normalform

Schema: Lieferant (**LNr**, LName, LStadt, LPLZ, **Produkt**, Preis)

- Vorgehen:
  - LName, LStadt und LPLZ werden aus Lieferant gelöscht
  - Gruppierung: Nur eine Gruppe mit LNr auf der linken Seite (es könnten Attribute von **Produkt** abhängen (2. Gruppe))
  - Erzeugen einer Relation mit **LNr**, LName, LStadt und LLand
- Ergebnis:

  Lieferung(**LNr**, **Produkt**, Preis)

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

3. Normalform

- Verhindere (zusätzlich zur 2NF), dass Attribute von nicht-primen Attributen abhängig sind

Beispiel: Redundanz ohne 3. NF

Lieferant

LNr	**LName**	**LStadt**	**LPLZ**	**Produkt**	**Preis**
103	Huber	Berlin	10115	Gabel	18
			⋮		
167	Meier	Berlin	10115	Löffel	15
			⋮		

- LStadt ist von LPLZ abhängig (umgekehrt eher nicht)
- Redundanz: die Information in LStadt wird mehrfach gespeichert

3. Normalform

Definition (3. Normalform)

Ein Relationenschema ist in dritter Normalform (3NF), wenn für jede nicht-triviale FA $X \to A$ gilt:

- X enthält einen Schlüsselkandidaten (d.h. ist ein Superschlüssel), oder
- A ist prim.

3. Normalform

Anmerkungen:

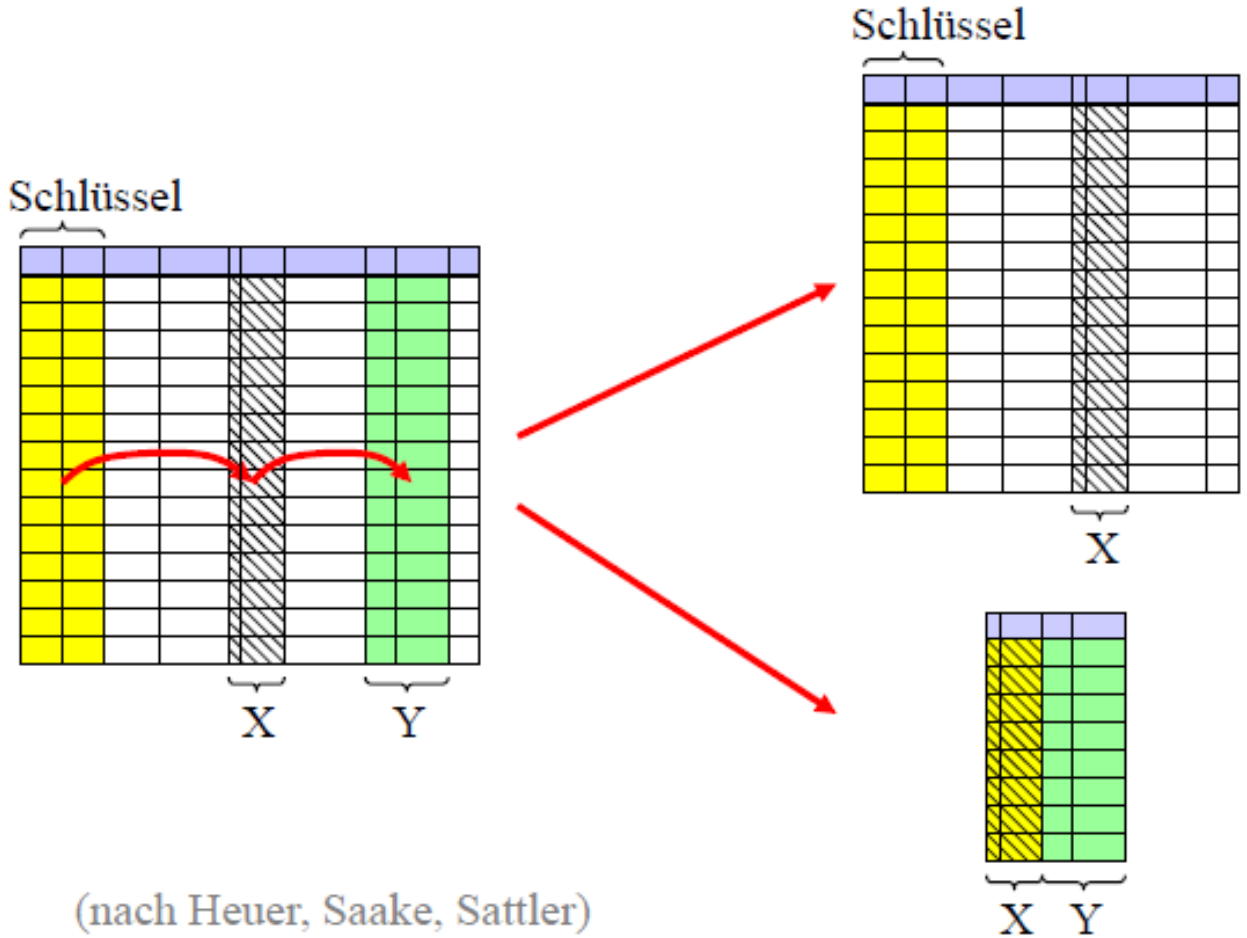
- "A ist prim" bedeutet: nur FAs interessant, bei denen rechte Seite nicht prim ist (FAs unter primen Attributen schwerer)
- Abhängigkeiten von nicht-primen Attributen bezeichnet man auch als *transitive* FAs (vom Schlüssel)
- 3NF impliziert 2NF (X ist mindestens ein Schlüsselkandidat, also keine partielle FA möglich!)
- Wegen Relfixivitäts- und Verstärkungsaxiom:
  - Triviale FAs $A \to A$ gelten natürlich immer, daher der Zusatz ".. für jede nicht-triviale FA ..."
  - Ist S Schlüssel der Relation R, dann gilt immer auch $S' \to R$ für jede Obermenge $S' \supseteq S$; daher die Formulierung "X enthält einen Schlüsselkandidaten" (statt X ist ein Schlüssel)

3. Normalform

- Transformation eines Relationenschema R in 3NF analog wie vorher Zerlegung
  - Attribute, die voll funktional abhängig vom Schlüssel sind und nicht abhängig von nicht-primen Attributen, bleiben in R
  - Für alle FAs $X_i \rightarrow Y_i$ von einem Teil des Schlüssels S ($X_i \subset S$) oder von Nicht-Schlüssel-Attributen X_i:
    - Lösche die Attribute Y_i aus R
    - Gruppiere die FAs nach gleichen linken Seiten X_l
    - Für jede dieser Gruppen, führe eine neue Relation ein mit allen enthaltenen Attributen aus X_l und Y_l
    - X_l wird Schlüssel in der neuen Relation

3. Normalform

- Graphisch:



(nach Heuer, Saake, Sattler)

Eigenschaften einer Zerlegung

- Intuitiv wollen wir natürlich bei der Normalisierung keine Informationen verlieren, d.h. durch die Zerlegung von R z.B. in zwei Relationen R_1 und R_2 sollen die gleichen Informationen gespeichert werden können wie in R

Definition (Verlustlose Zerlegung)

Die Zerlegung einer Relation R in Teilrelationen R_1 und R_2 heißt verlustlos (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:
  - $(R_1 \cap R_2) \rightarrow R_1$
  - $(R_1 \cap R_2) \rightarrow R_2$
- Achtung: ist eine Zerlegung nicht verlustlos, ergibt der natürliche Join i.d.R. *mehr* Tupel als vor der Zerlegung

Eigenschaften einer Zerlegung

- Außerdem wollen wir durch die Zerlegung von R z.B. in zwei Relationen R_1 und R_2 intuitiv keine FDs "verlieren", d.h. jede FA soll mindestens einer der Teilrelationen vollständig zugeordnet sein (alle Attr. der linken Seite und das Attr. der rechten Seite müssen in einer der Teilrelationen enthalten sein)

Definition (Abhängigkeitserhaltende Zerlegung)

Die Zerlegung einer Relation R mit FAs F_R in Teilrelationen R_1 und R_2 (mit FAs F_{R_1} und F_{R_2}) heißt abhängigkeitserhaltend (auch: "hüllentreu"), wenn gilt:

$F_R = (F_{R_1} \cup F_{R_2})$ bzw. $\mathcal{H}_F(Attr(R)) = \mathcal{H}_F(Attr(R_1) \cup Attr(R_2))$.

- 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

Synthesealgorithmus für 3NF

- Der Synthesealgorithmus erzeugt zu einem gegebenen Schema R mit funktionalen Abhängigkeiten F eine verlustlose, abhängigkeitserhaltende Zerlegung in Relationen $R_1, ..., R_n$, sodass diese R_i alle in 3NF sind
- Der Algorithmus besteht aus vier Hauptschritten
  1. Bestimme die minimale Menge der FAs F_{min} (kanonische Überdeckung), die dieselben partiellen und transitiven Abhängigkeiten wie F beschreiben durch sog. Links- und Rechtsreduktionen
  2. Erzeuge neue Relationenenschemas aus F_{min} (durch oben skizzierte Zerlegungen)
  3. Rekonstruiere Schlüsselkandidaten
  4. Eliminiere überflüssige Relationen

Synthesealgorithmus für 3NF — Details

Schritt 1: Bestimme F_{min}

1. Führe für jede FA $X \to Y \in F$ eine Linksreduktion durch:
   - Überprüfe für alle $A \in X$, ob A überflüssig ist, d.h.
   - Falls $Y \subseteq \mathcal{H}_F(X - A)$, ersetze $X \to Y$ durch $(X - A) \to Y$
2. Führe für jede (verbliebene) FA $X \to Y \in F$ eine Rechtsreduktion durch:
   - Überprüfe für alle $B \in Y$, ob B überflüssig ist, d.h.
   - Falls $B \subseteq \mathcal{H}_{(F-(X \to Y)) \cup (X \to (Y-B))}(X)$, ersetze $X \to Y$ durch $X \to (Y - B)$
3. Entferne FAs der Form $X \to \{\}$
4. Fasse FAs mit gleichen linken Seiten zusammen, d.h. aus $X \to Y_1, ..., X \to Y_n$ wird $X \to Y_1 \cup ... \cup Y_n$

Synthesealgorithmus für 3NF — Details

Schritt 2: Zerlegung auf Basis von F_{min}

- Für jede FA $X \to Y \in F_{min}$:
  - Erzeuge eine neue Relation $R_X := X \cup Y$
  - Ordne R_X die FAs $F_X := \{X' \to Y' \in F_{min} \mid X' \cup Y' \in R_X\}$ zu

Schritt 3: Rekonstruiere Schlüsselkandidaten

- Falls eines der in Schritt 2 erzeugten Schemas einen Schlüsselkandidaten aus R bzgl. F_{min}
- Sonst: wähle einen Schlüsselkandidat $S \subseteq R$ und definiere eine zusätzliches Relation $R_S := S$ mit $F_S := \{\}$

Schritt 4: Eliminiere überflüssige Relationen

- Eliminiere alle Relationen R_X, die in einer anderen Relation $R_{X'}$ enthalten sind, d.h. $R_{\subseteq} R_{X'}$

Synthesealgorithmus für 3NF — Beispiel

Beispiel: Synthesealgorithmus für 3NF

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

(Dem Schlüssel liegt folgende Annahme zugrunde: ein Kunde kauft an einem Kaufdatum nur max. einmal ein und die selbe Ware)

Es gelten folgende FAs:

- Kunde, Ware $\rightarrow$ KLand
- Kunde, WGruppe $\rightarrow$ Anbieter
- Anbieter $\rightarrow$ WGruppe
- Ware $\rightarrow$ WGruppe
- Kunde $\rightarrow$ KOrt
- KOrt $\rightarrow$ KLand (dies sei hier angenommen!)

Synthesealgorithmus für 3NF — Beispiel

Beispiel: Synthesealgorithmus für 3NF (Fortsetzung)

F besteht aus folgenden FAs:

Kunde, Ware $\rightarrow$ KLand	Kunde, WGruppe $\rightarrow$ Anbieter	Anbieter $\rightarrow$ WGruppe
Ware $\rightarrow$ WGruppe	Kunde $\rightarrow$ KOrt	KOrt $\rightarrow$ KLand

Schritt 1: Kanonische Überdeckung F_{min} berechenen:

1. Linksreduktion:

   Kunde, Ware $\rightarrow$ KLand streiche Ware ...
   Es gilt $\{\text{KLand}\} \subseteq \mathcal{H}_F(\{\text{Kunde}, \text{Ware}\} - \{\text{Ware}\})$, weil KLand von Kunde alleine (transitiv über KOrt) funktional abhängig ist

2. Rechtsreduktion:

   Kunde $\rightarrow$ KLand streiche KLand ...
   Es gilt $\{\text{KLand}\} \subseteq \mathcal{H}_{F-(\text{Kunde}\rightarrow\text{KLand})\cup(\text{Kunde}\rightarrow\{\})}(\{\text{Kunde}\})$, weil KLand von Kunde (transitiv über KOrt) funktional abhängig ist

3. Entferne FAs der Form $X \rightarrow \{\}$:

   Kunde $\rightarrow \{\}$ wird eliminiert

4. Zusammenfassen: nichts zu tun

Synthesealgorithmus für 3NF — Beispiel

Beispiel: Synthesealgorithmus für 3NF (Fortsetzung)

Schritt 1: Kanonische Überdeckung F_{min} berechenen:

Ergebnis für F_{min}:

(1) Kunde, WGruppe $\rightarrow$ Anbieter (2) Anbieter $\rightarrow$ WGruppe
(3) Ware $\rightarrow$ WGruppe (4) Kunde $\rightarrow$ KOrt (5) KOrt $\rightarrow$ KLand

Schritt 2: Zerlegung auf Basis von F_{min}

- **Bezugsquelle** (**Kunde**, **WGruppe**, Anbieter)
  FAs: (1) und (2) aus F_{min}
- **Lieferant** (**Anbieter**, WGruppe) FAs: (2) aus F_{min}
- **Produkt** (**Ware**, WGruppe) FAs: (3) aus F_{min}
- **Adresse** (**Kunde**, KOrt) FAs: (4) aus F_{min}
- **Land** (**KOrt**, KLand) FAs: (5) aus F_{min}

Synthesealgorithmus für 3NF — Beispiel

Beispiel: Synthesealgorithmus für 3NF (Fortsetzung)

Schritt 3: Rekonstruiere Schlüsselkandidaten:

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**)

Schritt 4: Eliminiere überflüssige Relationen:

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

Ergebnis:

- **Bezugsquelle** (**Kunde**, **WGruppe**, Anbieter)
- **Produkt** (**Ware**, WGruppe)
- **Adresse** (**Kunde**, KOrt)
- **Land** (**KOrt**, KLand)
- **Einkauf** (**Ware**, **Kunde**, **Kaufdatum**)

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: Abhängigkeiten unter prim Attributen

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

- Es gilt eine 1:1-Beziehung zw. Hersteller und HerstellerNr:
  - Hersteller $\rightarrow$ HerstellerNr
  - HerstellerNr $\rightarrow$ Hersteller
- Schlüsselkandidaten sind deshalb:
  - {Hersteller, ModellNr}
  - {HerstellerNr, ModellNr}
- Damit ist Schema in 3. NF, da alle Attribute prim sind
- Trotzdem Anomalien (wegen 1:1-Beziehung oben) möglich

Boyce-Codd-Normalform

Definition (Boyce-Codd-Normalform)

Ein Relationenschema ist in Boyce-Codd-Normalform (BCNF), wenn für jede nicht-triviale FA $X \rightarrow Y$ gilt:

X enthält einen Schlüsselkandidaten (d.h. ist Superschlüssel)

- Verlustlose Zerlegung ist generell immer möglich
- Abhängigkeitserhaltende Zerlegung nicht immer möglich

Zusammenfassung und Ausblick

- Normalisierung ist wichtig um Redundanzen und damit Anomalien zu vermeiden, die zu inkonsistenten Datenbeständen führen können
- Formale Grundlage für die Normalisierung sind die Normalformen
- Normalisierung erreicht man durch Zerlegung in mehrere Relationen
- Nachteil der Normalisierung: die ursprünglichen Informationen müssen durch teure Joins berechnet werden
- Abhilfe durch (materialisierte) Views bis hin zu einer gezielten De-Normalisierung (also nicht jede FA bei der Normalisierung berücksichtigen)

Zusammenfassung und Ausblick

- Neben der ersten NF sind in der Praxis insbesondere die 2NF (partielle FA von Schlüsselkandidaten) und die 3NF (transitive Abhängigkeiten) relevant
- Es gibt weitere FAs, die durch weitere Normalformen eliminiert werden können
- In der Praxis haben noch eine gewisse Relevanz (auch wenn sie meist aus Performanz-Gründen nicht umgesetzt werden):
  - Boyce-Codd-Normalform (BCNF): eliminiert FAs unter prim Attributen
  - 4. Normalform (4NF): eliminiert sog. *mehrwertige* FAs (diese können theoretisch aus mehreren unabhängigen 1:n-Beziehungen entstehen)

Überblick

- 1. Normalform:
  Alle Attribute sind atomar
- 2. Normalform:
  Keine partiellen FAs von allen Schlüsselkandidaten
- 3. Normalform:
  Zusätzlich keine transitiven FAs (von nicht-primen Attributen)
- Boyce-Codd-Normalform:
  Zusätzlich keine FAs unter primen Attributen
- 4. Normalform:
  keine Redundanz durch mehrwertige FAs