LMU LUDWIG-MAXIMILIANS-UNIVERSITY MUNICH
DEPARTMENT INSTITUTE FOR INFORMATICS
DATABASE SYSTEMS GROUP

Skript zur Vorlesung:

Einführung in die Informatik: Systeme und Anwendungen

Sommersemester 2017

Kapitel 3: Datenbanksysteme

Vorlesung: Prof. Dr. Christian Böhm
Übungen: Dominik Mautz

http://www.dbs.ifi.lmu.de/cms/ Einführung_in_die_Informatik_Systeme_und_Anwendungen

Überblick

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:**

LNr	LName	LStadt	LLand	Ware	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 Attrbut LNr (ebenso LLand von LStadt).

Motivation Normalisierung

- **Relation Lieferant:**

LNr	LName	LStadt	LLand	Ware	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:

LieferantAdr	(LNr, LName, LStadt)
Stadt	(LStadt, LLand)
Angebot	(LNr, Ware, 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
```

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*:
  - Definitionen ähnlich
  - Für alle Schlüsselkandidaten $S = \{A,B,...\}$ gilt:
    Alle Attribute der Relation sind von S funktional abhängig:
    $\{A,B,...\} \rightarrow R$
- 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 des Schlüssels (nicht vom gesamten Schlüssel)
- 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 $X' \subset X$ gibt, von der Y ebenfalls funktional abhängt,
  - dann heißt $X \rightarrow Y$ eine ***volle funktionale Abhängigkeit*** ($X \dot{\rightarrow} Y$)
  - andernfalls eine ***partielle funktionale Abhängigkeit***

Partielle und *volle* FD

- Beispiele:
  - LNr $\overset{\bullet}{\rightarrow}$ Lname — voll funktional abhängig (von einem Element)
  - LNr, Ware $\rightarrow$ LName — partiell funktional abhängig (LNr $\rightarrow$ LName)
  - Ware ? Preis — nicht funktional abhängig (siehe Beispiel)
  - LNr ? Preis — nicht funktional abhängig (siehe Beispiel)
  - LNr, Ware $\overset{\bullet}{\rightarrow}$ 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 $X \overset{\bullet}{\rightarrow} Y$: 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 $X \rightarrow Y$.
  Inbesondere 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 Relationenausprägung erfüllt werden, für die auch F erfüllt ist.
- Es sind alle FDs ableitbar, die durch F impliziert sind.

Herleitung funktionaler Abhängigkeit

Triviale funktionale Abhängigkeit:

- Ein Attribut ist immer funktional abhängig:
  - von sich selbst
  - und von jeder Obermenge von sich selbst

Solche Abhängigkeiten bezeichnet man als ***trivial***

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→Z  in F do
         if Y ⊆ Erg then Erg:=Erg ∪ Z
Ausgabe X+ = Erg
```

Hülle einer Attributmenge

- **Beispiel:** AttrHülle(F, {LNr}) mit
  F = {LNr → LName; LNr → LStadt; LStadt → LLand; LNr, Ware → Preis}

- Erg_i = Erg nach i-tem Durchlauf der While-Schleife

- Erg_0 = { LNr}
- Erg_1 = { LNr, LName, LStadt }
- Erg_2 = { LNr, LName, LStadt, LLand }
- Erg_3 = { LNr, LName, LStadt, LLand } = 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: $F = \{ LNr \rightarrow LName;\ LNr \rightarrow LStadt;\ LStadt \rightarrow LLand;\ LNr, Ware \rightarrow Preis \}$.
- Zu zeigen: (LNr, Ware) eindeutig und minimal.
- Beweis:
  - (I) (LNr, Ware) eindeutig gdw.
    $(LNr, Ware \rightarrow LNr, LName, LStadt, LLand, Ware, Preis) \in F^+$,
    d.h. ist aus F herleitbar.

  - (a) $(LNr \rightarrow LName) \in F \subseteq F^+ \overset{VS}{\Rightarrow} (LNr, Ware \rightarrow LName, Ware) \in F^+$
  - (b) $(LNr \rightarrow LStadt) \in F^+ \overset{VS}{\Rightarrow} (LNr, Ware \rightarrow LStadt, Ware) \in F^+$
  - (c) $(LNr \rightarrow LStadt) \in F^+$ und $(LStadt \rightarrow LLand) \in F^+$
    $\overset{T}{\Rightarrow} (LNr \rightarrow LLand) \in F^+ \overset{VS}{\Rightarrow} (LNr, Ware \rightarrow LLand, Ware) \in F^+$
  - (d) Wegen R gilt $(LNr \rightarrow LNr) \in F^+ \overset{VS}{\Rightarrow} (LNr, Ware \rightarrow LNr, Ware) \in F^+$
  - (e) aus (a) bis (e) und $(LNr, Ware \rightarrow Preis)$ folgt (I) nach VE, qed.

Herleitung funktionaler Abhängigkeit

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

- Gegeben: F = { LNr → LName; LNr → LStadt; LStadt → LLand;
  LNr, Ware → Preis }.
- Zu zeigen: (LNr, Ware) eindeutig und minimal.
- Beweis:
  (II) (LNr, Ware) minimal gdw.
  (LNr → LNr, LName, LStadt, LLand, Ware, Preis) und
  (Ware → LNr, LName, LStadt, LLand, Ware, Preis) gelten nicht.

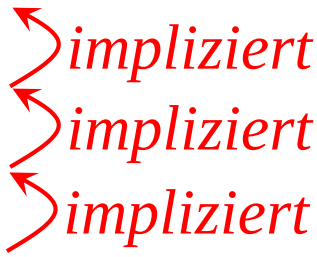
  (a) Preis ist nur von LNr und Ware gemeinsam funktional abhängig.

  (b) Weder LNr kann aus Ware hergeleitet werden, noch Ware von LNr.

  (c) aus (a) und (b) folgt (LNr, 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



- Herstellung einer Normalform durch verlustlose Zerlegung des Relationenschemas

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:

LNr	LName	LStadt	LLand	Ware	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

Anomalien?

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)
- 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

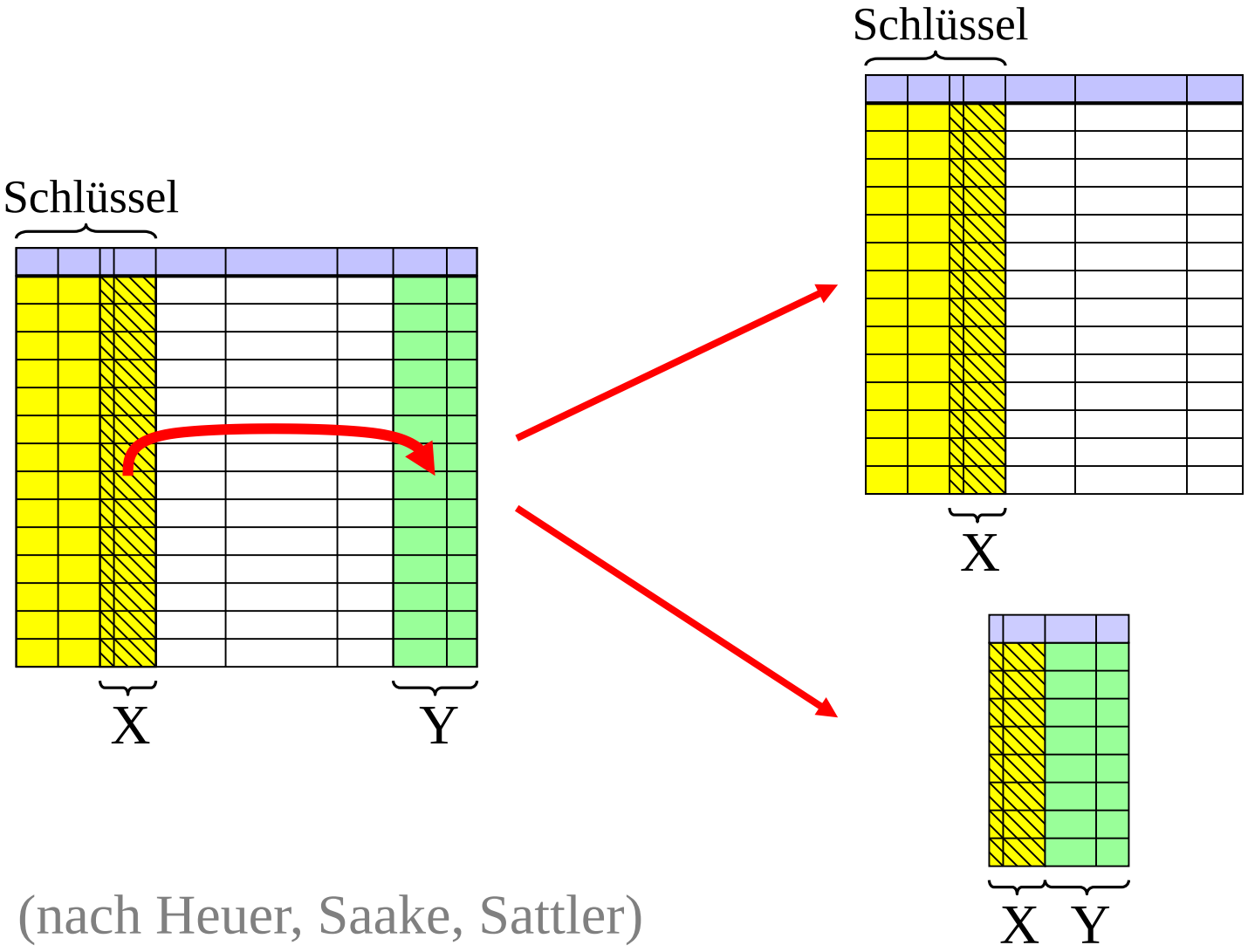
2. Normalform

- 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
    - Gruppiere 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)
- 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)

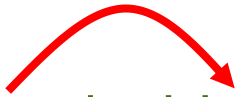
Grafische Darstellung



(nach Heuer, Saake, Sattler)

3. Normalform

- Motivation:
  Man möchte zusätzlich verhindern, dass Attribute von nicht-primen Attributen funktional abhängen.
- Beispiel:
  LieferAdr (LNr, LName, LStadt, LLand)



001	Huber	München	Deutschland
002	Meier	Berlin	Deutschland
003	Müller	Köln	Deutschland
004	Hinz	Wien	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 nichttriviale Abhängigkeit $X \rightarrow Y$ gilt:
  - X enthält einen Schlüsselkandidaten
  - oder Y ist prim
- Beobachtung: 2. Normalform ist mit impliziert

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 -> Boyce-Codd-Normalform)
- 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

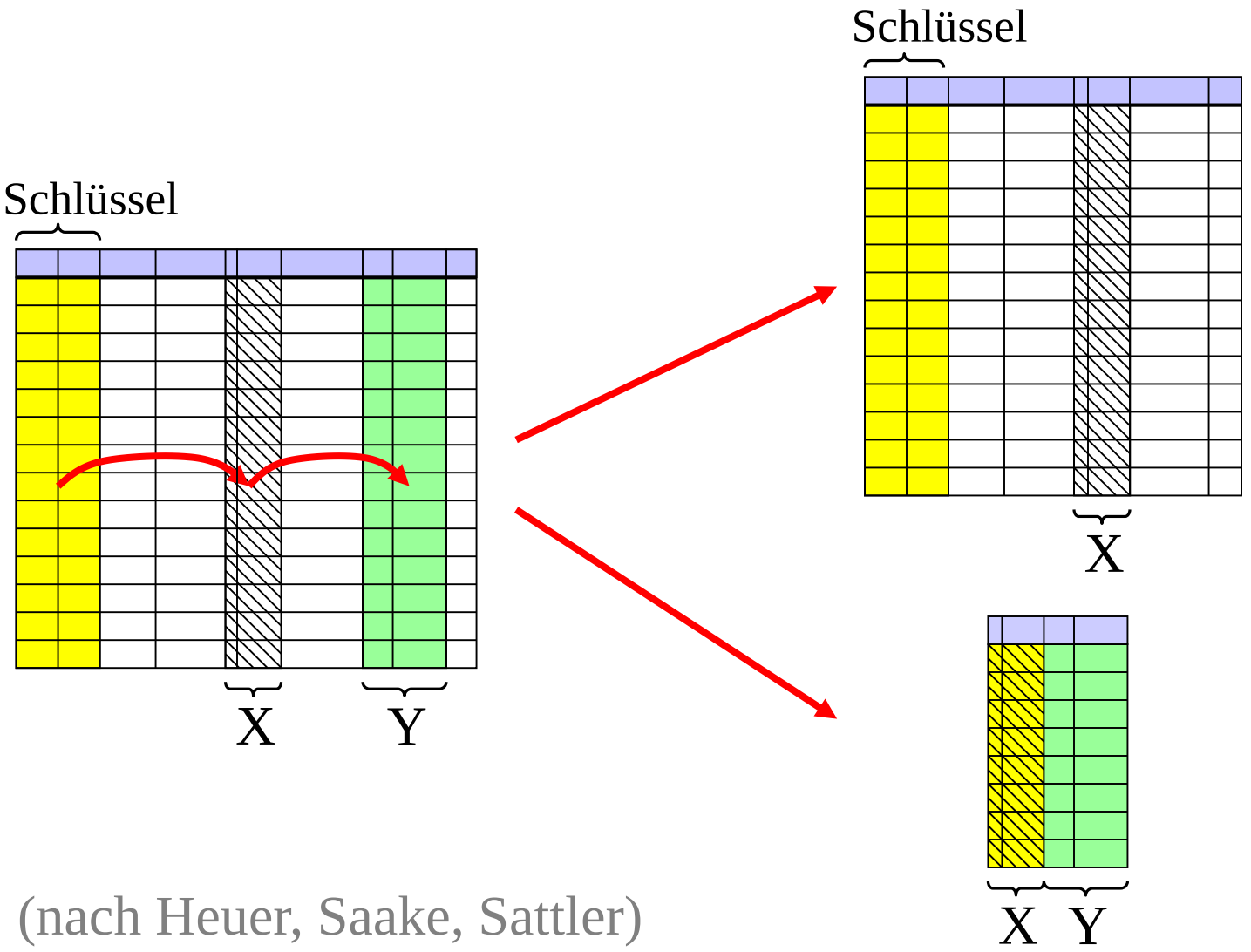
Anmerkungen zum Verständnis dieser Definition:

- 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 ($X_i \subset S$) oder von Nicht-Schlüssel-Attribut:
    - Lösche die Attribute Y_i aus R
    - Gruppiere 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

Grafische Darstellung



(nach Heuer, Saake, Sattler)

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:

$$F_R = (F_{R1} \cup F_{R2}) \quad \text{bzw.} \quad F_R{}^+ = (F_{R1} \cup F_{R2})^+$$

- 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 $R_1, \ldots, R_n$ die folgende Kriterien erfüllt:
- $R_1, \ldots, R_n$ ist eine verlustlose Zerlegung von R.
- Die Zerlegung ist abhängigkeitserhaltend.
- 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
  $Y \subseteq AttrHülle(F, X - A)$
  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
  $B \subseteq AttrHülle((F - (X \rightarrow Y)) \cup (X \rightarrow (Y - B)), X)$
  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 $X \rightarrow Y_1, \ldots, X \rightarrow Y_n$ zusammen, so dass
$X \rightarrow (Y_1 \cup \ldots \cup Y_n)$ 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)

Es gelten folgende FDs: F = {

Kunde, Ware → KLand
Kunde, WGruppe → Anbieter
Anbieter → WGruppe
Ware → WGruppe
Kunde → KOrt
KOrt → KLand

}

Schritte des Synthesealgorithmus:

1. Kanonische Überdeckung F_c der funktionalen Abhängigkeiten:
   a) Linksreduktion:
      Kunde, ~~Ware~~ → KLand,
      {KLand} ⊆ *AttrHülle*(*F*, {*Kunde, Ware*}–{*Ware*})
      weil KLand von Kunde alleine (transitiv) funktional abhängig ist
   b) Rechtsreduktion:
      Kunde → ~~KLand~~,
      {KLand} ⊆ *AttrHülle*(*F* – (Kunde → KLand) ∪ (Kunde → {}), Kunde)
      weil KLand transitiv von Kunde funktional abhängig ist
   c) Kunde → {} wird eliminiert
   d) nichts zu tun.

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 → Anbieter
   Anbieter → WGruppe
   Ware → WGruppe
   Kunde → KOrt
   KOrt → KLand
2. Erzeugen der neuen Relationenschemata und ihrer FDs:
   Bezugsquelle (Kunde, WGruppe, Anbieter) {Kunde,WGruppe → Anbieter, Anbieter → WGruppe}
   Lieferant (Anbieter, WGruppe) {Anbieter → WGruppe}
   Produkt (Ware, WGruppe) {Ware → WGruppe}
   Adresse (Kunde, KOrt) {Kunde → KOrt}
   Land (KOrt, KLand) {KOrt → 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.

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 (sog. Trigger)
- Aber es gibt auch Konzepte, Relationen so abzuspeichern, dass Join auf bestimmten Attributen unterstützt wird
  - ORACLE-Cluster

Zusammenfassung

- 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