Ludwig Maximilians Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Skript zur Vorlesung

Datenbanksysteme I

Wintersemester 2008/2009

Kapitel 7: Normalformen

Vorlesung: Prof. Dr. Christian Böhm
Übungen: Annahita Oswald, Bianca Wackersreuther

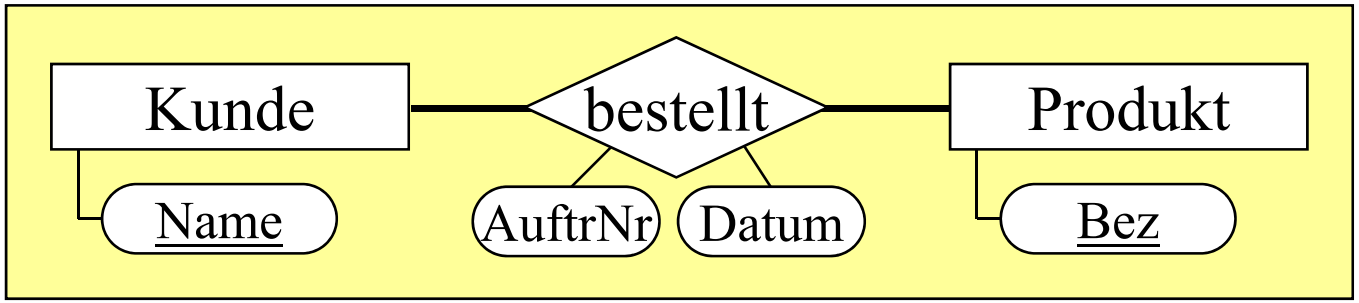
http://www.dbs.informatik.uni-muenchen.de/Lehre/DBS

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

- Nicht immer liefert das E/R-Modell ein redundanzfreies Datenbankschema:



Schema:

Kunde	(Name, ...)
Produkt	(Bez, ...)
bestellt	(Name, Bez, AuftrNr, Datum)

Redundanz: Kundenauftrag für mehrere Produkte

Motivation Normalisierung

- Tabelleninhalt Bestellt:

Name	Bez	AuftrNr	Datum
Huber	Schraube	01	01.01.02
Huber	Nagel	01	01.01.02
Huber	Schraube	02	01.02.02
Meier	Schraube	03	05.01.02

- Hier gibt es offensichtlich einige Redundanzen:
  - →zwei verschiedene Datums zu einem Auftrag möglich
  - →zwei verschiedene Kunden zu einem Auftrag möglich
- Redundanzen durch funktionale Abhängigkeiten
  - Datum funktional abhängig von AuftrNr
  - Name funktional abhängig von AuftrNr

Weiteres Beispiel

Datenbankschema aus Kapitel 3:

Kunde	(KName, KAdr, Kto)
Auftrag	(KName, Ware, Menge)
Lieferant	(LName, LAdr, Ware, Preis)

Das Schema **Lieferant** hat folgende Nachteile:

- Redundanz
  - für jede Ware wird die Adresse des Lieferanten gespeichert, d.h. die Adresse ist mehrfach vorhanden
- Insert-/Delete-/Update-Anomalien
  - **update**: Adressänderung in 1 Tupel
  - **insert**: Einfügen eines Lieferanten erfordert Ware
  - **delete**: Löschen der letzten Ware löscht die Adresse

Verbesserung

Datenbankschema aus Kapitel 3:

Kunde	(KName, KAdr, Kto)
Auftrag	(KName, Ware, Menge)
LiefAdr	(LName, LAdr)
Angebot	(LName, Ware, Preis)

- Vorteile:
  - keine Redundanz
  - keine Anomalien
- Nachteil:
  - Um zu einer Ware die Adressen der Lieferanten zu finden, ist 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.LName, LAdr, Ware, Preis
    from     LieferantAdr L, Angebot A
    where    L.LName = A.LName
```

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

- Ein Teilmenge S der Attribute von R heißt Superschlüssel, wenn nur die Eindeutigkeit gilt.

Definition: *funktional abhängig*

- Gegeben:
  - Ein Relationenschema R
  - A, B: Zwei *Mengen* von Attributen von R ($A,B \subseteq R$)
- Definition:
  B ist von A funktional abhängig (A $\rightarrow$ B) gdw.
  für alle möglichen Ausprägungen von R gilt:

$$falls \quad \forall r,s \in R \quad mit \quad r.A = s.A \quad gilt: r.B = s.B$$

  Zu jedem Wert in A exist. genau ein Wert von B.

- Beispiel **Lieferant** (LName, LAdr, Ware, Preis):
  - {LName} $\rightarrow$ {LAdr}
  - {LName, Ware} $\rightarrow$ {LAdr}
  - {LName, Ware} $\rightarrow$ {Preis}

üblicherweise schreibt man keine Klammern

Bei mehreren Attributen

- Steht auf der linken Seite mehr als ein Attribut:

$$A_1, A_2, \ldots, A_n \rightarrow B$$

dann gilt: *B* ist von der *Kombination* aus Attributen f.a.:

$$r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n \Rightarrow r.B = s.B$$

- Steht auf der rechten Seite mehr als ein Attribut:

$$A \rightarrow B_1, B_2, \ldots B_n$$

dann ist dies eine abkürzende Schreibweise für:

$$A \rightarrow B_1, A \rightarrow B_2, \ldots, A \rightarrow B_n$$

(wenn die Kombination von *B*-Werten von *A* f.a. ist, dann ist auch jeder einzelne *B*-Wert von *A* f.a. und umgekehrt)

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 Rel. sind von S funktional abhängig:
    $$A, B, ... \rightarrow \mathrm{R}$$
    (das ist Folge der *Eindeutigkeits-Eigenschaft* von S)
- 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)

Vergleich mit *Schlüssel*

- Die funktionale Abhängigkeit ist also eine

  Verallgemeinerung des Schlüssel-Konzepts

- 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

Triviale Funktionale Abhängigkeit

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

Solche Abhängigkeiten bezeichnet man als

triviale funktionale Abhängigkeit

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 $A \rightarrow B$
  - Wenn es keine echte Teilmenge $A' \subset A$ gibt, von der B ebenfalls funktional abhängt,
  - dann heißt $A \rightarrow B$ eine volle funktionale Abhängigkeit
  - andernfalls eine partielle funktionale Abhängigkeit

(Anmerkung: Steht auf der linken Seite nur ein Attribut, so ist die FD immer eine volle FD)

Partielle und *volle* FD

- Beispiele:
  - LName → LAdr — voll funktional abhängig
  - LName, Ware → LAdr — partiell funktional abhängig
  - Ware ? Preis — nicht funktional abhängig
  - LName, Ware → Preis — voll funktional abhängig

Prime Attribute

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

Volle FD und minimaler Schlüssel

- Aus der Eindeutigkeits-Eigenschaft ergibt sich, dass alle Attribute von einem Schlüssel *funktional abhängig* sind.
- Frage: Ergibt sich aus der Minimalität des Schlüssels auch, dass alle Attribute von S *voll funktional abhängig* sind?
- Dies würde nahe liegen, denn Definitionen sind ähnlich: „Es gibt keine echte Teilmenge, so dass …“
  - Eindeutigkeit erhalten bleibt (Minimalität Schlüssel)
  - Funktionale Abhängigkeit erhalten bleibt (volle FD)
- Trotzdem gilt: Einzelne Attribute können partiell von einem Schlüssel abhängig sein:
  - LName, Ware → LAdr: partiell funktional abhängig
- Das „Tupel als Ganzes“ ist aber vom Schlüssel *voll f.a.*

Herleitung funktionaler Abhängigkeit

Armstrong Axiome

- Reflexivität: Falls β eine Teilmenge von α ist ($\beta \subseteq \alpha$) dann gilt immer $\alpha \rightarrow \beta$. Inbesondere gilt also immer $\alpha \rightarrow \alpha$.
- Verstärkung: Falls $\alpha \rightarrow \beta$ gilt, dann gilt auch $\alpha\gamma \rightarrow \beta\gamma$. Hierbei steht $\alpha\gamma$ für $\alpha \cup \gamma$.
- Transitivität: Falls $\alpha \rightarrow \beta$ und $\beta \rightarrow \gamma$ gilt, dann gilt auch $\alpha \rightarrow \gamma$.

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.
- alle FDs ableitbar, die durch F impliziert sind.

Hülle einer Attributmenge

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

```
AttrHülle(F,α)
   Erg := α
   while( Änderungen an Erg) do
     foreach FD β→γ  in F do
         if β ⊆ Erg then Erg:=Erg ∪ γ
   Ausgabe α + = Erg
```

Verlustlose Zerlegung

- Eine Zerlegung von R in $R_1,\ldots, R_n$ ist *verlustlos*, falls sich jede mögliche Ausprägung r von R durch den natürlichen Join der Ausprägungen $r_1,\ldots,r_n$ rekonstruieren läßt:

$$r = r_1 \bowtie \ldots \bowtie r_n$$

- Beispiel für eine nicht-verlustlose Zerlegung:

  In der Relation *Einkauf* wird beschrieben, welche Waren ein Kunde (exklusiv) bei welchem Anbieter bezieht (d.h. es gelte *Kunde, Ware* $\rightarrow$ *Anbieter*):

Einkauf

Anbieter	Ware	Kunde
Meier	Eier	Schmidt
Meier	Milch	Huber
Bauer	Milch	Schmidt

Verlustlose Zerlegung

- Eine mögliche Zerlegung in die Relationen *Lieferant* und *Bedarf* ergibt:

Lieferant

Anbieter	**Kunde**
Meier	Schmidt
Meier	Huber
Bauer	Schmidt

Bedarf

Ware	**Kunde**
Eier	Schmidt
Milch	Huber
Milch	Schmidt

- Diese Zerlegung ist **nicht** verlustlos, da die Rekonstruktion von Einkauf als natürlicher Join von Lieferant und Bedarf misslingt, d.h.

$$Lieferant \bowtie Bedarf \neq Einkauf$$

Verlustlose Zerlegung

- Im konkreten Beispiel erhält man zusätzliche (unerwünschte) Tupel:

Lieferant ⋈ Bedarf

Anbieter	Ware	Kunde
Meier	Eier	Schmidt
Meier	*Milch*	*Schmidt*
Meier	Milch	Huber
Bauer	Milch	Schmidt
Bauer	*Eier*	*Schmidt*

Verlustlose Zerlegung

- Hinreichendes Kriterium für Verlustlosigkeit:

  Eine (binäre) Zerlegung von R mit den funktionalen Abhängigkeiten F in R_1 und R_2 ist verlustlos, wenn mindestens eine der folgenden funktionalen Abhängigkeiten auf der Basis von F herleitbar ist:

$$R_1 \cap R_2 \rightarrow R_1$$
$$R_1 \cap R_2 \rightarrow R_2$$

- Im Beispiel gilt nur die nicht-triviale Abhängigkeit

$$\text{Kunde, Ware} \rightarrow \text{Anbieter}$$

  nicht aber eine der beiden Abhängigkeiten, welche die Verlustlosigkeit garantieren würden:

$$\text{Kunde} \rightarrow \text{Anbieter}$$
$$\text{Kunde} \rightarrow \text{Ware}$$

Abhängigkeitserhaltende Zerlegung

- Eine Zerlegung von R in $R_1,...,R_n$ ist *abhängigkeitserhaltend*, wenn die Überprüfung aller funktionalen Abhängigkeiten F auf R lokal auf den R_i erfolgen kann, ohne dass Joins berechnet werden müssen.
- Es gibt dann keine übergreifenden Abhängigkeiten F' über die lokalen F_i hinaus und für die Menge der funktionalen Abhängigkeiten F auf R gilt:

$$F = F_1 \cup ... \cup F_n$$

Abhängigkeitserhaltende Zerlegung

- Beispiel: **Bank** (Filiale, Kunde, Betreuer)

  Funktionale Abhängigkeiten:

  Betreuer → Filiale

  Kunde, Filiale → Betreuer

- Mögliche Zerlegung :

  Personal (Filiale, Betreuer)

  Kunde (Kunde, Betreuer)

- Diese Zerlegung ist ...
  - *verlustlos* (d.h. *Personal* ⋈ *Kunden* = *Bank*), da Betreuer → Betreuer, Filiale gilt.
  - *nicht abhängigkeitserhaltend*, da Kunde, Filiale → Betreuer verlorengegangen ist.

Normalisierung

- In einem Relationenschema sollen 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

1. Normalform

- Keine Einschränkung bezüglich der FDs
- Ein Relationenschema ist in erster Normalform, wenn alle Attributwerte *atomar* sind
- In relationalen Datenbankem 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:

Lieferant (LName, LAdr, Ware, Preis)

Bäcker	Ibk	Brot	3,00
Bäcker	Ibk	Semmel	0,30
Bäcker	Ibk	Breze	0,40
Metzger	Hall	Filet	5,00
Metzger	Hall	Wurst	4,00

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 Schlüssel(-Kandidat) zusammengesetzt ist

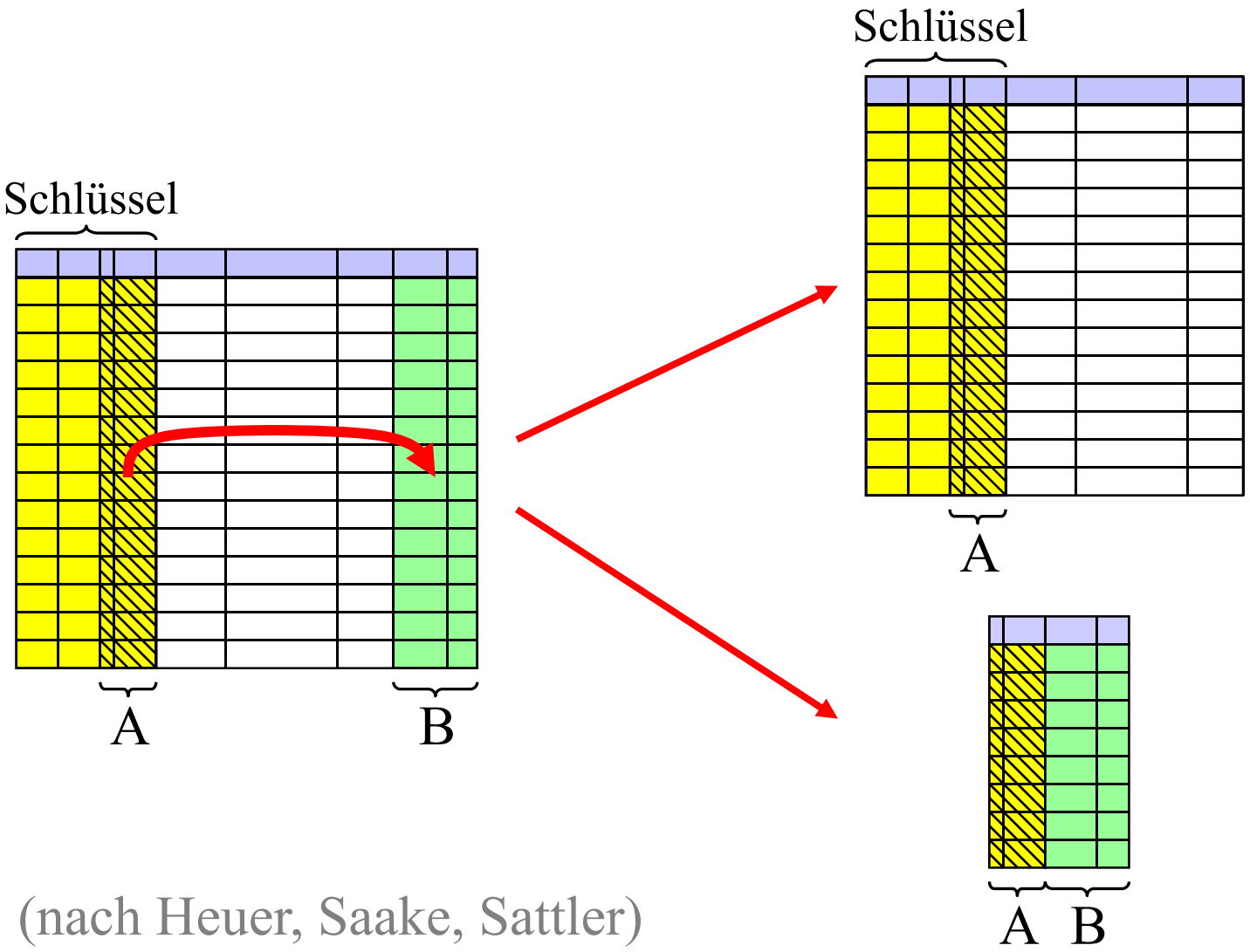
2. Normalform

- 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 $A_i \rightarrow B_i$ von einem Teil eines Schlüssels ($A_i \subset S$) geht man folgendermaßen vor:
    - Lösche die Attribute B_i aus R
    - Gruppiere die Abhängigkeiten nach gleichen linken Seiten A_i
    - Für jede Gruppe führe eine neue Relation ein mit allen enthaltenen Attributen aus A_i und B_i
    - A_i wird Schlüssel in der neuen Relation

2. Normalform

- Beispiel:
  Lieferant (LName, LAdr, Ware, Preis)
- Vorgehen:
  - LAdr wird aus Lieferant gelöscht
  - Gruppierung:
    Nur eine Gruppe mit LName auf der linken Seite
    - es könnten aber noch weitere Attribute von LName abhängig sein (selbe Gruppe)
    - es könnten Attribute von Ware abh. (2. Gruppe)
  - Erzeugen einer Relation mit LName und LAdr
- Ergebnis: **Lieferant** (LName, Ware, Preis)
  LieferAdr (LName, LAdr)

Grafische Darstellung



(nach Heuer, Saake, Sattler)

3. Normalform

- Motivation:
  Man möchte zusätzlich verhindern, dass Attribute von nicht-primen Attributen funktional abhängen.
- Beispiel:
  Bestellung (AuftrNr, Datum, KName, KAdresse)

001	24.04.02	Meier	Innsbruck
002	25.04.02	Meier	Innsbruck
003	25.04.02	Huber	Hall
004	25.04.02	Huber	Hall
005	26.04.02	Huber	Hall

- Redundanz: Kundenadresse mehrfach gespeichert
- Anomalien?

3. Normalform

- Abhängigkeit von Nicht-Schlüssel-Attribut bezeichnet man häufig auch als *transitive Abhängigkeit* vom Primärschlüssel
  - weil Abhängigkeit *über* ein drittes Attribut besteht:

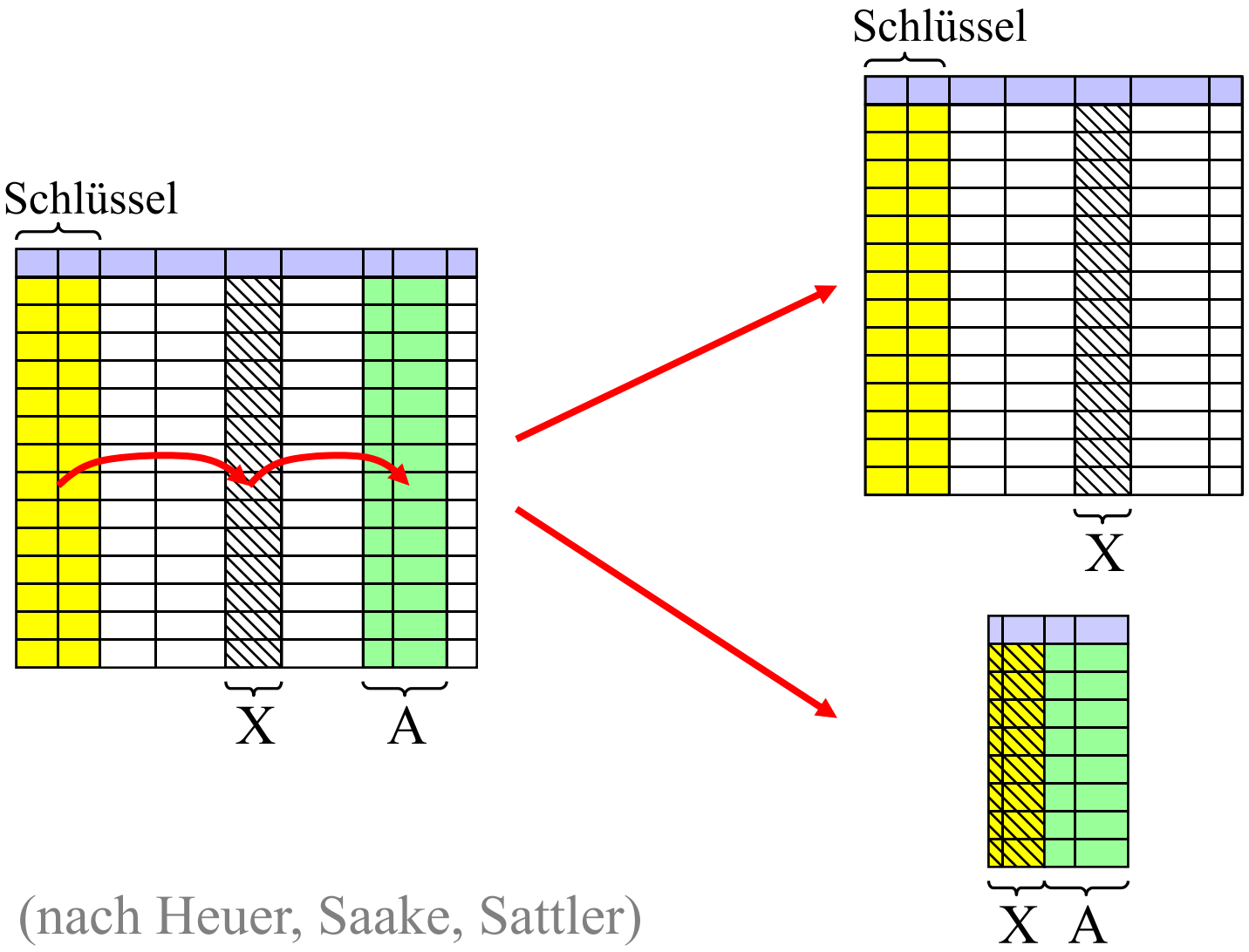
    $\underline{\text{AuftrNr}} \rightarrow \text{KName} \rightarrow \text{KAdr}$

- Definition:
  Ein Relationenschema ist in 3. Normalform, wenn für jede nichttriviale Abhängigkeit $X \rightarrow A$ gilt:
  - X enthält einen Schlüsselkandidaten
  - oder A ist prim
- Beobachtung: 2. Normalform ist mit impliziert

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 $A_i \rightarrow B_i$ von einem Teil eines Schlüssels ($A_i \subset S$) oder von Nicht-Schlüssel-Attribut:
    - Lösche die Attribute B_i aus R
    - Gruppiere die Abhängigkeiten nach gleichen linken Seiten A_i
    - Für jede Gruppe führe eine neue Relation ein mit allen enthaltenen Attributen aus A_i und B_i
    - A_i wird Schlüssel in der neuen Relation

Grafische Darstellung



(nach Heuer, Saake, Sattler)

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. Erzeugung eines neuen Relationenschemas aus F_c
3. Rekonstruktion eines Schlüsselkandidaten
4. Elimination überflüssiger Relationen

Synthesealgorithmus für 3NF

Bestimmung der kanonischen Überdeckung der Menge der funktionalen Abhängigkeiten:

$AB \rightarrow C$
$A \rightarrow C$
$\Downarrow$
$A \rightarrow C$

1. **Linksreduktion** der FDs $A \rightarrow B$, um *partielle* Abhängigkeiten zu entfernen:
   Für jedes $\alpha \in A$, ersetze die Abhängigkeit $A \rightarrow B$ durch $(A-\alpha) \rightarrow B$, falls α auf der linken Seite überflüssig ist, d.h. falls *B* schon durch $(A-\alpha)$ determiniert ist.

$A \rightarrow B \rightarrow C$
$A \rightarrow C$
$\Downarrow$
$A \rightarrow B \rightarrow C$

2. **Rechtsreduktion** der (verbliebenen) FDs $A \rightarrow B$ zur Entfernung *transitiver* Abhängigkeiten:
   Für jedes $\beta \in B$, ersetze die Abhängigkeit $A \rightarrow B$ durch $A \rightarrow (B-\beta)$, falls β auf der rechten Seite überflüssig ist, d.h. falls $A \rightarrow \beta$ eine transitive Abhängigkeit ist.
3. **Entfernung** von rechts-leeren funktionalen Abhängigkeiten $A \rightarrow \varnothing$, die bei der Rechtsreduktion möglicherweise entstanden sind.
4. **Zusammenfassen** von Abhängigkeiten mit gleichen linken Seiten, so daß jede linke Seite nur einmal vorkommt:
   Ersetze die Abhängigkeiten $A \rightarrow B_1, \ldots, A \rightarrow B_m$ durch $A \rightarrow (B_1 \cup \ldots \cup B_m)$.

Synthesealgorithmus für 3NF

2. Erzeugung eines neuen Relationenschemas aus F_c:
   - Erzeuge ein Relationenschema $R_A = (A \cup B)$
   - Ordne dem Schema R_A die FDs $F_A = \{(A' \rightarrow B') \in F_c \mid A' \cup B' \subseteq R_A\}$ zu.
3. Rekonstruktion eines Schlüsselkandidaten:
   Falls eines der in Schritt 2. erzeugten Schemata R_A einen Schlüsselkandidaten von R enthält, sind wir fertig. Ansonsten wähle einen Schlüsselkandidaten $\kappa \in R$ aus und erzeuge das zusätzliche Schema $R_A = \kappa$ mit $F_A = \emptyset$.
4. Elimination überflüssiger Relationen:
   - Eliminiere diejenigen Schemata $R_{A'}$ die in einem anderen Schema $R_{A'}$ enthalten sind: $R_A \subseteq R_{A'}$

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.

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 → HerstellerNr
    HerstellerNr → 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 $X \rightarrow A$ gilt: X enthält einen Schlüsselkandidaten von R

- Die Zerlegung ist teilweise schwieriger.
- Man muß auf die Verlustlosigkeit achten.
- Verlustlose Zerlegung nicht immer möglich.

Mehrwertige Abhängigkeiten

- 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)

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)

Name,	Projekte,	Verwandte)
Huber,	P1,	Heinz,
Huber,	P2,	Hans,
Huber,	P3,	Hubert,
Müller,	P2,	Manfred
Müller,	P3,	NULL

Mehrwertige Abhängigkeiten

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

Mitarbeiter (Name, Projekte, Verwandte)

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)

Geg: $\alpha,\beta,\gamma \subseteq R$, mit $R= \alpha \cup \beta \cup \gamma$

β ist ***mehrwertig abhängig*** von α ($\beta \rightarrow\rightarrow \alpha$), wenn für jede gültige Ausprägung von R gilt: Für jedes Paar aus Tupeln t_1, t_2 mit $t_1.\alpha = t_2.\alpha$, aber (natürlich) $t_1.\beta\gamma <> t_2.\beta\gamma$ existieren 2 weitere Tupel t_3 und t_4 mit folgenden Eigenschaften:

$$t_1.\alpha = t_2.\alpha = t_3.\alpha = t_4.\alpha$$
$$t_3.\beta = t_1.\beta$$
$$t_3.\gamma = t_2.\gamma$$
$$t_4.\beta = t_2.\beta$$
$$t_4.\gamma = t_1.\gamma$$

Jede FD ist auch eine MVD !!!

Beispiel MVD

	R		
	α $\overbrace{A_1 \ldots A_i}$	β $\overbrace{A_{1+1} \ldots A_j}$	γ $\overbrace{A_{j+1} \ldots A_n}$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$
t_4	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$

Weiteres Beispiel

Relation: Modelle

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

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

Farben

ModellNr	Farbe
E36	blau
E36	schwarz
E34	Schwarz

Leistung

ModellNr	Leistung
E36	170 PS
E36	198 PS
E34	170 PS

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

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. $R_1 \cap R_2 \rightarrow\rightarrow R_1$ oder
  2. $R_1 \cap R_2 \rightarrow\rightarrow R_2$

Triviale MVD und 4. Normalform

Eine MVD $\alpha \rightarrow\rightarrow \beta$ bezogen auf $R \supseteq \alpha \cup \beta$ ist *trivial*, wenn jede mögliche Ausprägung *r* von R diese MVD erfüllt. Man kann zeigen, daß $\alpha \rightarrow\rightarrow \beta$ trivial ist, genau dann wenn:

1. $\beta \subseteq \alpha$ oder
2. $\beta = R - \alpha$

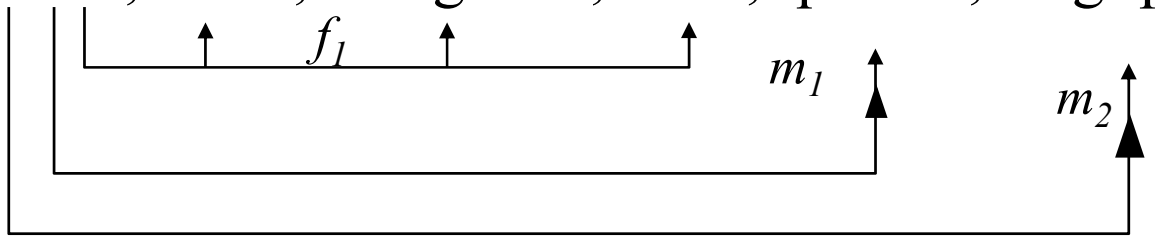
Eine Relation R mit zugeordneter Menge D von funktionalen und mehrwertigen Abhängigkeiten in ***4NF***, wenn für jede MVD $\alpha \rightarrow\rightarrow \beta \in D^+$ eine der folgenden Bedingungen gilt:

1. Die MVD ist trivial oder
2. α 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.