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

Skript zur Vorlesung:

Einführung in die Informatik: Systeme und Anwendungen

Sommersemester 2008

Kapitel 3: Datenbanksysteme

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

http://www.dbs.ifi.lmu.de/Lehre/InfoNF

Überblick

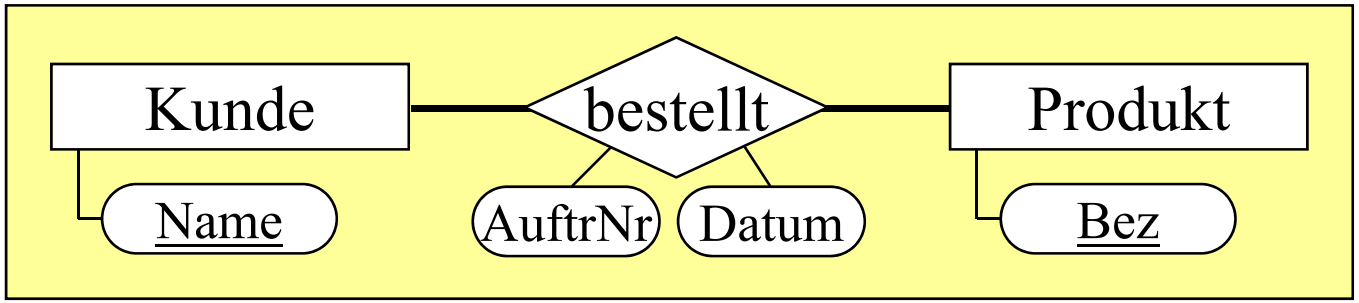
Relationaler Datenbank-Entwurf

- 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

- 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 3.3:

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 3.3:

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

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:

  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

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:
    $\{A,B,...\} \rightarrow R$
- 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

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***

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***

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

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:

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

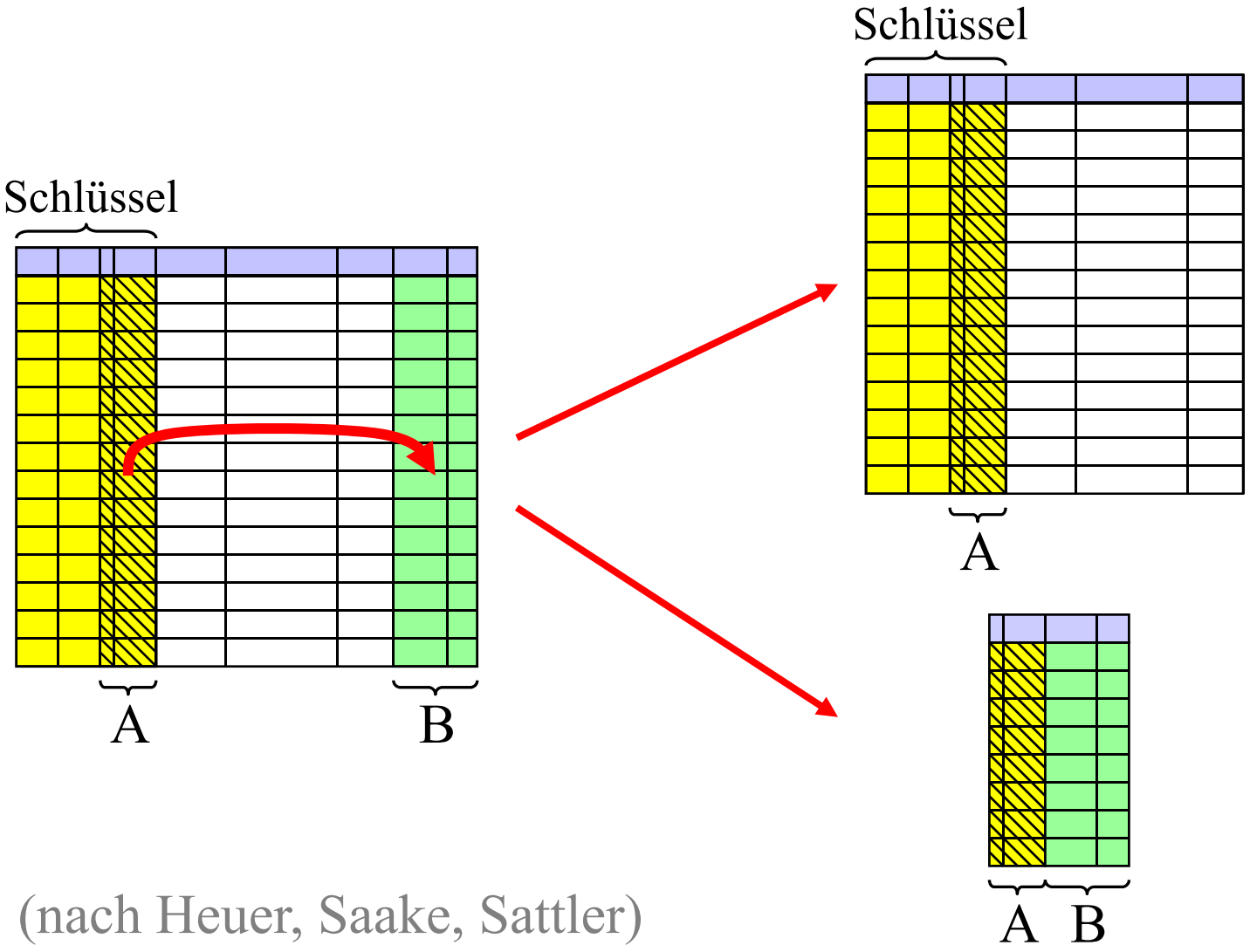
- Zur Transformation in 2. Normalform spaltet man das Relationenschema auf:
  - Attribute, die voll funktional abhängig vom Schlüssel sind, bleiben in der Ursprungsrelation R
  - Für alle Abhängigkeiten $A_i \rightarrow B_i$ von einem Teil eines Schlüssels ($A_i \subset S$) geht man folgendermaßen vor:
    - 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

einzige partielle Abhängigkeit

- 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

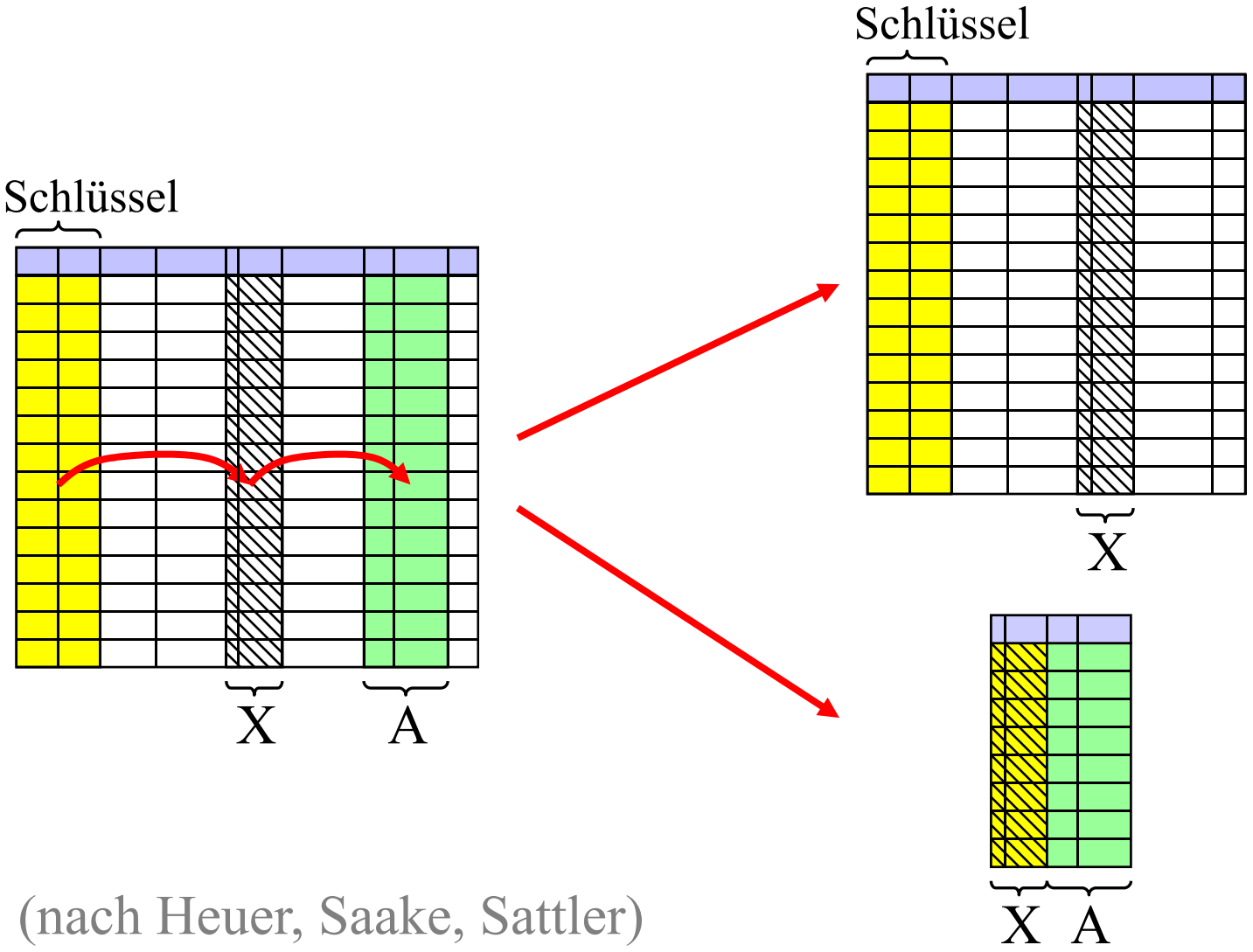
- Abhängigkeit von Nicht-Schlüssel-Attribut bezeichnet man häufig auch als ***transitive Abhängigkeit*** vom Primärschlüssel
  - weil Abhängigkeit *über* ein drittes Attribut besteht:

- 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)

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