Technische Hochschule Rosenheim

Kapitel 4 – Relationaler Datenbankentwurf

Vorlesung Datenbanken

Prof. Dr. Kai Höfig

Überblick

- 4.1 Redundanzen & Anomalien
- 4.2 Funktionale Abhängigkeiten und Schlüssel
- 4.3 Hülle, Membership, CLOSURE und COVER
- 4.4 Normalformen
- 4.5 Transformationen und Zerlegung

Redundanzen

Redundanzen in Relationen aus mehreren Gründen unerwünscht

- Hauptproblem: Redundanzen führen zu Anomalien
  - Änderungsanomalien (*update anomalies*): **Änderungsoperationen** auf Relationen mit Redundanzen schwer korrekt umsetzbar: wenn eine Information redundant vorkommt, muss eine Änderung diese Information in allen ihren Vorkommen verändern
    → mit normalen relationalen Änderungsoperationen und den in relationalen Systemen vorkommenden lokalen Integritätsbedingungen (Schlüsseln) nur schwer realisierbar
    → treten bei INSERT oder UPDATE auf
  - Löschanomalien (*deletion anomalies*): **Löschoperationen** können zum unbeabsichtigten Verlust weitere Informationen führen
    → treten bei DELETE auf
- Weniger kritisches Problem: Redundante Informationen belegen unnötigen **Speicherplatz**

Beispiel für eine Änderungsanomalie

Änderungsanomalie

- Einfügen inkonsistenter Informationen in die RWEINE-Relation:

```
insert into RWEINE(Name, Farbe, Jahrgang,
                   Weingut, Anbaugebiet, Region)
values             ('Chardonnay', 'Weiß', 2004,
                   'Helena', 'Rheingau', 'Kalifornien')
```

- Weingut Helena war bisher im Napa Valley angesiedelt, nun enthält die Datenbank beide Informationen (Napa Valley und Rheingau)
- Rheingau lag bisher in Hessen, nun liegt es zusätzlich in Kalifornien

RWEINE

Name	Farbe	Jahrgang	Weingut	Anbaugebiet	Region
Pinot Noir	Rot	1999	Helena	Napa Valley	Kalifornien
Riesling Reserve	Weiß	1999	Müller	Rheingau	Hessen
Chardonnay	Weiß	2002	Helena	Rheingau	Kalifornien

Beispiel für eine Löschanomalie

- Löschanomalie
  - Löschen der Information, dass das Weingut `'Chateau La Rose'` einen Wein Namens `'La Rose GrandCru'` herstellt

```
delete from  RWEINE
    where Name = 'La Rose GrandCru'
```

  - Verliert (unbeabsichtigt) die Information, dass `Saint-Emilion` im `Bordeaux` liegt.

RWEINE

Name	Farbe	Jahrgang	Weingut	Anbaugebiet	Region
~~La Rose GrandCru~~	~~Rot~~	~~1998~~	~~Chateau La Rose~~	~~Saint-Emilion~~	~~Bordeaux~~
Creek Shiraz	Rot	2003	Creek	Barossa Valley	South Australia
Zinfandel	Rot	2004	Helena	Napa Valley	Kalifornien
Pinot Noir	Rot	2001	Creek	Barossa Valley	South Australia
Pinot Noir	Rot	1999	Helena	Napa Valley	Kalifornien
Riesling Reserve	Weiß	1999	Müller	Rheingau	Hessen
Chardonnay	Weiß	2002	Bighorn	Napa Valley	Kalifornien

Vermeiden von Anomalien

Vermeiden von Anomalien in Datenbanken

Schritt 1: Erkennen von Redundanzen

Schritt 2: Beseitigen der Redundanzen

Schritt 1: Erkennen von Redundanzen (1)

Redundanzen folgen aus der Anwendung – aus dem Wissen über die Daten

- Beispiele für „Wissen über die Daten“
  - Jedes `Weingut` liegt in genau einem `Anbaugebiet`, d.h. es darf nicht zwei Tupel mit gleichen `Weingut` aber unterschiedlichem `Anbaugebiet` geben.
  - Jedes `Anbaugebiet` liegt in genauer einer `Region`, d.h. wenn bei zwei Tupeln das `Anbaugebiet` gleich ist, muss auch die `Region` gleich sein.
- Allgemein:

  Wenn die Werte für das Attribut A bei zwei Tupeln gleich sind,
  dann müssen auch die Werte für Attribut B gleich sein.

RWEINE

Name	Farbe	Jahrgang	Weingut	Anbaugebiet	Region
La Rose GrandCru	Rot	1998	Chateau La Rose	Saint-Emilion	Bordeaux
Creek Shiraz	Rot	2003	Creek	Barossa Valley	South Australia
Zinfandel	Rot	2004	[illegible]	[illegible]	[illegible]nien
Pinot Noir	Rot	2001	Creek	Barossa Valley	South Australia
Pinot Noir	Rot	1999	Helena	Napa Valley	Kalifornien
Riesling Reserve	Weiß	1999	Müller	[illegible]	[illegible]
Chardonnay	Weiß	2002	Bighorn	Napa Valley	Kalifornien

Weingut bestimmt Anbaugebiet

Anbaugebiet bestimmt Region

Schritt 1: Erkennen von Redundanzen (2)

Redundanzen folgen aus der Anwendung – aus dem Wissen über die Daten

- Komplexeres Beispiel: Wenn `Name`, `Jahrgang` und `Weingut` gleich sind, müssen auch `Anbaugebiet` und `Farbe` gleich sein.
- Allgemein:
  Wenn die Werte für die Attribute $A_1, A_2, \ldots, A_n$ gleich sind,
  dann müssen auch die Werte für die Attribute $B_1, B_2, \ldots, B_m$ gleich sein

RWEINE

Name	**Farbe**	**Jahrgang**	**Weingut**	**Anbaugebiet**	**Region**
La Rose GrandCru	Rot	1998	Chateau La Rose	Saint-Emilion	Bordeaux
Creek Shiraz	Rot	2003	Creek	Barossa Valley	South Australia
Zinfandel	Rot	2004	Helena	Napa Valley	Kalifornien
Pinot Noir	Rot	2001	Creek	Barossa Valley	South Australia
Pinot Noir	Rot	1999	Helena	Napa Valley	Kalifornien
Riesling Reserve	Weiß	1999	Müller	Rheingau	Hessen
Chardonnay	Weiß	2002	Bighorn	Napa Valley	Kalifornien

Schritt 1: Erkennen von Redundanzen (3)

Funktionale Abhängigkeiten

- Wenn die Werte für die Attribute $A = A_1, A_2, \ldots, A_n$ gleich sind, dann müssen auch die Werte für die Attribute $B = B_1, B_2, \ldots, B_m$ gleich sein.
- Wir schreiben $A \rightarrow B$ und meinen *A bestimmt eindeutig B* oder *B ist funktional abhängig von A*.
- Funktionale Abhängigkeiten (FD=*functional dependencies*) lassen sich nur aus dem Kontext der jeweiligen Datenbankanwendung ermitteln.
- Der **Anwender** kennt die FDs und Aufgabe des **Datenbankdesigners** ist es alle oder zumindest eine repräsentative Menge (später) aller FDs zu ermitteln.
- Dies ist ein nicht automatisierbarer Prozess, denn eine FD $A \rightarrow B$ darf nicht nur für die aktuelle Menge von Tupeln gelten, sondern auch für alle möglichen zukünftigen Tupel.

  - Beispiel Weindatenbank: Im Gespräch zwischen einem Weinkenner (Anwender, Fachexperte) und dem Datenbankdesigner erwähnt der Weinkenner, dass es keine Anbaugebiete gibt, die über mehrere Regionen verteilt liegen. Der hellhörige Datenbankdesigner notiert hier sofort die FD Anbaugebiet → Region.

Schritt 2: Beseitigen der Redundanzen (1)

- Beispiel für eine funktionale Abhängigkeit von vorher:
  Jedes `Weingut` liegt in genau einem `Anbaugebiet`, d.h. es darf nicht zwei Tupel mit gleichen `Weingut` aber unterschiedlichem `Anbaugebiet` geben.

RWEINE

Name	Farbe	Jahrgang	Weingut	Anbaugebiet	Region
La Rose GrandCru	Rot	1998	Chateau La Rose	Saint-Emilion	Bordeaux
Creek Shiraz	Rot	2003	Creek	Barossa Valley	South Australia
Zinfandel	Rot	2004	Helena	Napa Valley	Kalifornien
Pinot Noir	Rot	2001	Creek	Barossa Valley	South Australia
Pinot Noir	Rot	1999	Helena	Napa Valley	Kalifornien
Riesling Reserve	Weiß	1999	Müller	Rheingau	Hessen
Chardonnay	Weiß	2002	Bighorn	Napa Valley	Kalifornien

ERZEUGER

Weingut	Anbaugebiet	Region
Creek	Barossa Valley	South Australia
Helena	Napa Valley	Kalifornien
Chateau La Rose	Saint-Emilion	Bordeaux
Chateau La Pointe	Pomerol	Bordeaux
Müller	Rheingau	Hessen
Bighorn	Napa Valley	Kalifornien

WEINE

Name	Farbe	Jahrgang	Weingut
La Rose GrandCru	Rot	1998	Chateau La Rose
Creek Shiraz	Rot	2003	Creek
Zinfandel	Rot	2004	Helena
Pinot Noir	Rot	2001	Creek
Pinot Noir	Rot	1999	Helena
Riesling Reserve	Weiß	1999	Müller
Chardonnay	Weiß	2002	Bighorn

→ Dies nennt man **Normalisierung**

Schritt 2: Beseitigen der Redundanzen (2)

- Beispiel für eine funktionale Abhängigkeit von vorher: Jedes `Weingut` liegt in genau einem `Anbaugebiet`, d.h. es darf nicht zwei Tupel mit gleichen `Weingut` aber unterschiedlichem `Anbaugebiet` geben.
- Wie stellt man so etwas in einer relationalen Datenbank sicher?
- Erste Idee: `Weingut` muss ein Schlüssel der Tabelle sein!
  - Dann sorgt das DBMS dafür, dass es keine zwei Tupel mit gleichem `Weingut` gibt
- Aber: `Weingut` kann nicht Schlüssel von `RWEINE` sein, dann können wir ja für jedes `Weingut` nur noch einen Wein in `RWEINE` speichern.
- Lösung: Aufteilen von `RWEINE` in zwei Relationen – `WEINE` und `ERZEUGER`.
- Soweit gut, aber wir haben ja noch mehr funktionale Abhängigkeiten (z.B. zu jedem `Anbaugebiet` gibt es genau eine `Region`), wie müssen das also für alle funktionalen Abhängigkeiten machen (sprich: Relationen weiter aufteilen)

➔ Dies nennt man **Normalisierung**

Zusammenfassung

- Es gibt viele Möglichkeiten, ein Schema für eine relationale DB zu erstellen
  - z.B. aus einem ER-Diagramm oder direkt aus der Anforderungsanalyse
- Häufig ist das erste Schema jedoch verbesserungswürdig, weil es Redundanzen enthält.
- Redundanzen führen zu Anomalien (und Speicherplatzverschwendung).
- Funktionale Abhängigkeiten beschreiben Anwendungswissen über die Daten. Und erlauben es uns, Redundanzen zu erkennen und zu beseitigen.
- Normalformentheorie: Mittels funktionaler Abhängigkeiten formulierte Bedingungen, die ein Schema erfüllen muss, damit es keine Redundanzen enthält und somit keine Anomalien verursacht.
- Normalisierung: Zerlegung der Relationen mit Hilfe funktionaler Abhängigkeiten in zwei oder mehr neue Relationen, die keine entsprechenden Redundanzen mehr enthalten und somit keine Anomalien verursachen.

Identifizierende Attributmenge

- Eine **Identifizierende Attributmenge** ist eine Menge von Attributen $\{B_1, \ldots, B_k\} \subseteq R$ so, dass sich zwei unterschiedliche Tupel über R immer in mind. Einem Attributwert $B \in \{B_1, \ldots, B_k\}$ unterscheiden. Formal:
  - $\forall t_1, t_2 \in r \,.\, [\; t_1 \neq t_2 \;\Rightarrow\; \exists B \in \{B_1, \ldots, B_k\} : t_1(B) \neq t_2(B) \;]$

  Oder anders formuliert: betrachte man die Tupel nur über B, gibt es keine identischen Zeilen.

  Beispiel: {`Name, Jahrgang, Weingut`}, **{`Name, Farbe, Jahrgang, Weingut`}**

  Aber nicht: **{`Anbaugebiet,Region`}**

RWEINE

Name	Farbe	Jahrgang	Weingut	Anbaugebiet	Region
La Rose GrandCru	Rot	1998	Chateau La Rose	Saint-Emilion	Bordeaux
Creek Shiraz	Rot	2003	Creek	Barossa Valley	South Australia
Zinfandel	Rot	2004	Helena	Napa Valley	Kalifornien
Pinot Noir	Rot	2001	Creek	Barossa Valley	South Australia
Pinot Noir	Rot	1999	Helena	Napa Valley	Kalifornien
Riesling Reserve	Weiß	1999	Müller	Rheingau	Hessen
Chardonnay	Weiß	2002	Bighorn	Napa Valley	Kalifornien

Schlüssel

- Eine **minimale** identifizierende Attributmenge nennt man einen **Schlüssel**.
  - Primattribut: Attribut eines Schlüssels
  - Primärschlüssel: beim Entwurf ausgezeichneter Schlüssel
  - Identifizierende Attributmengen werden auch Oberschlüssel oder Superkey genannt (da sie Obermenge eines Schlüssels sind)

  - Beispiel: {`Name, Jahrgang, Weingut`} ist (der einzige) Schlüssel für `WEINE`

RWEINE

Name	**Jahrgang**	**Weingut**	**Anbaugebiet**	**Region**
La Rose GrandCru	1998	Chateau La Rose	Saint-Emilion	Bordeaux
Creek Shiraz	2003	Creek	Barossa Valley	South Australia
Zinfandel	2004	Helena	Napa Valley	Kalifornien
Pinot Noir	2001	Creek	Barossa Valley	South Australia
Pinot Noir	1999	Helena	Napa Valley	Kalifornien
Riesling Reserve	1999	Müller	Rheingau	Hessen
Chardonnay	2002	Bighorn	Napa Valley	Kalifornien

Fremdschlüssel

- Wenn jeder vorkommende Wert für ein Attribut *X* in einer Relation R_1 auch für ein Attribut *Y* in einer Relation R_2 vorkommt, nennt man *X* einen **Fremdschlüssel** in R_1 für *Y* in R_2. Formal:

- Sei $X \subseteq R_1$, $Y \subseteq R_2$. Die Fremdschlüsselbedingung $X(R_1) \rightarrow Y(R_2)$ ist erfüllt, wenn gilt :

$$\{t(X) \mid t \in r_1\} \subseteq \{t(Y) \mid t \in r_2\}$$

- Beispiel:
  $X = Y = \{\texttt{Weingut}\}$ ist Fremdschlüssel in `WEINE` bzgl. `ERZEUGER`
  weil die Fremdschlüsselbedingung erfüllt ist:

  $\texttt{Weingut(WEINE)} \rightarrow \texttt{Weingut(ERZEUGER)}$, d.h.

  $$\{t(\{\texttt{Weingut}\}) \mid t \in r_1(\texttt{WEINE})\} \subseteq \{t(\{\texttt{Weingut}\}) \mid t \in r_2(\texttt{ERZEUGER})\}$$

  in Worten: jeder vorkommende Wert für `Weingut` in `WEINE` muss auch als Wert für `Weingut` in `ERZEUGER` vorkommen.

Funktionale Abhängigkeiten - formal

- Funktionale Abhängigkeit (FD, von functional dependency) zwischen Attributmengen $A_1, A_2, \ldots, A_n$ und $B_1, B_2, \ldots, B_m$ eines Relationenschemas R: Wenn in jeder Relation über R die Werte der $A_1, A_2, \ldots, A_n$ -Attribute die Werte der $B_1, B_2, \ldots, B_m$ -Attribute festlegen. Es gilt außerdem $A \rightarrow A$.

- Unterscheiden sich zwei Tupel in den $A_1, A_2, \ldots, A_n$ -Attributen nicht, so haben sie auch gleiche Werte für alle $B_1, B_2, \ldots, B_m$ -Attribute

- Notation für funktionale Abhängigkeit: $\{A_1, A_2, \ldots, A_n\} \rightarrow \{B_1, B_2, \ldots, B_m\}$ (Mengenklammer bei den Attributmengen werden häufig weggelassen)
  - Beispiele für `RWEINE`:
    1) `{Name, Jahrgang, Weingut}` $\rightarrow$ `{Anbaugebiet, Farbe}`
    2) `Anbaugebiet` $\rightarrow$ `Region`
    3) aber nicht: `Weingut` $\rightarrow$ `Name`

- **Achtung: FDs sagen etwas über *alle möglichen* Datenbankinhalte aus, nicht nur über den aktuellen Inhalt**

Zusammenhang Funktionale Abhängigkeit und Schlüssel

- Eine Mengen von Attributen $K \subseteq R$ ist genau dann ein **Schlüssel** von R, wenn
  1. $K \to R$
  2. Es gibt keine echte Teilmenge $X \subset K$ mit $X \to R$

  In Worten: die Attribute in K bestimmen eindeutig alle Attribute von R und sind minimal.

- Eine Mengen von Attributen $S \subseteq R$ ist genau dann ein **Oberschlüssel** (**Superkey**) von R, wenn $S \to R$

- Das bedeutet: wenn wir Funktionale Abhängigkeiten finden, wo rechts eine ganze Relation steht, haben wir einen (Ober)Schlüssel gefunden.

Beispiel für Schlüssel

- In der Relation `ERZEUGER` gilt
  `Weingut` → `Weingut, Anbaugebiet, Region`
  Trivialerweise gibt es keine echte Teilmenge von `{Weingut}` die `ERZEUGER` funktional bestimmt. Somit ist `{Weingut}` ein Schlüssel für `ERZEUGER`

  Superkey sind `{Weingut,Region}`, `{Weingut,Anbaugebiet}` und `{Weingut,Anbaugebiet,Region}`

ERZEUGER	**<u>Weingut</u>**	**Anbaugebiet**	**Region**
	Creek	Barossa Valley	South Australia
	Helena	Napa Valley	Kalifornien
	Chateau La Rose	Saint-Emilion	Bordeaux
	Chateau La Pointe	Pomerol	Bordeaux
	Müller	Rheingau	Hessen
	Bighorn	Napa Valley	Kalifornien

Was nützt uns das denn nun?

- Wenn wir *alle* FDs kennen, die eine Relation erfüllt, können wir die Schlüssel der Relation bestimmen!
  - Bisher waren Schlüssel immer „einfach da“, nun haben wir eine Idee, wo diese eigentlich herkommen!
  - Nur leider kennen wir meist nicht *alle* FDs, sondern nur ein paar. Wir brauchen also Regeln, um aus gegebenen FDs weitere (bzw. alle) FDs zu bestimmen („berechnen“).

- FDs und Redundanzen (und damit Anomalien) hängen zusammen
  - → wir werden im folgenden Bedingungen mittels der FDs formulieren, so dass eine Relation, die diese Bedingungen erfüllt, frei von Änderungs- und Löschanomalien ist
  - → Wir werden weiterhin einen Algorithmus kennen lernen, mit dem wir eine gegebene Relation, die diese Bedingungen nicht erfüllt, mittels der FDs in mehrere Relationen aufteilen können, die die Bedingungen erfüllen
  - Auch dazu benötigen wir jedoch Regeln zum Umformen von („rechnen“ mit) FDs

Ableitung von FDs

- Beispiel für die Ableitung von FDs:
  - Relation `R(A,B,C)` mit den funktionale Abhängigkeiten

    $A \rightarrow B$

    $B \rightarrow C$
  - Mögliche Ausprägung:

R

A	B	C
a1	b1	c1
a2	b1	c1
a3	b2	c1
a4	b1	c1

  - offensichtlich gilt damit auch: $A \rightarrow C$ jedoch nicht: $C \rightarrow A$ oder $C \rightarrow B$

- Allgemeine Fragen
  - gegeben eine Menge von FDs F, welche weiteren FDs f lassen sich aus F ableiten?
  - Bzw. noch interessanter: gegeben eine Menge von FDs F, kann eine bestimmte FD f aus F abgeleitet werden (Membership-Problem)?

Transitivität

- Funktionale Abhängigkeiten sind transitiv, das bedeutet für $X, Y, Z \subseteq R$ gilt:

$$\{ X \rightarrow Y,\ Y \rightarrow Z \} \quad \Rightarrow\ X \rightarrow Z$$

- Beweis:
  - Annahme: in $r(R)$ gelten:
    (1) $X \rightarrow Y$ und
    (2) $Y \rightarrow Z$
  - Demzufolge für zwei beliebige Tupel $t_1, t_2 \in r(R)$ mit $t_1(X) = t_2(X)$ muss gelten:
    (3) $t_1(Y) = t_2(Y)$ (wegen (1))
    (4) $t_1(Z) = t_2(Z)$ (wegen (3) und (2))
  - daher gilt: $X \rightarrow Z$

R	A	B	C
	a1	b1	c1
	a2	b1	c1
	a3	b2	c1
	a4	b1	c1

$$A \rightarrow B\ ,\ B \rightarrow C\ \Rightarrow\ A \rightarrow C$$

Reflexivität

- Funktionale Abhängigkeiten sind reflexiv, das bedeutet für $Y \subseteq X \subseteq R$ gilt:

  $$X \supseteq Y \Rightarrow X \rightarrow Y$$

- Beweis:
  - Annahme: $Y \subseteq X \subseteq R$;
    $t_1, t_2 \in r(R)$ mit $t_1(X) = t_2(X)$
  - dann folgt: $t_1(Y) = t_2(Y)$ wegen $X \supseteq Y$, und daraus folgt: $X \rightarrow Y$

R

A	B	C
a1	b1	c1
a2	b1	c1
a3	b2	c1
a4	b1	c1

$$A \subseteq AB \subseteq R \Rightarrow AB \rightarrow A$$

Augmentation

- Für funktionale Abhängigkeiten gilt:

$$\{ X \to Y \} \quad \Rightarrow \; XZ \to YZ \text{ sowie } XZ \to Y$$

- Beweis:
  - Annahme:
    $X \to Y$ gilt in $r(R)$
    $XZ \to YZ$ gilt nicht.
  - Es gibt zwei Tupel $t_1, t_2 \in r(R)$ für die gilt:
    $t_1(X) = t_2(X) \Rightarrow \; t_1(Y) = t_2(Y)$
    $t_1(XZ) = t_2(XZ) \Rightarrow t_1(Z) = t_2(Z)$
    $t_1(XZ) = t_2(XZ) \Rightarrow t_1(YZ) \neq t_2(YZ)$ Wiederspruch.

R

A	B	C
a1	b1	c1
a1	b1	c2
a3	b2	c3
a4	b1	c4

R[A,C]

A	C
a1	c1
a1	c2
a3	c3
a4	c4

R[B,C]

B	C
b1	c1
b1	c2
b2	c3
b1	c4

$$A \to B \; \Rightarrow \; AC \to BC \text{ und } AC \to B$$

Ableitungsregeln

- Forderungen an eine sinnvolle Mengen von Ableitungsregeln
  - gültig (sound): Regeln leiten keine FDs ab, die logisch nicht impliziert sind
  - vollständig (complete): alle implizierten FDs werden abgeleitet
  - unabhängig (independent) (d.h. oder auch minimal): keine Regel kann weggelassen werden

- Allgemein übliche Regelmenge (gültig & vollständig)
  - F1 Reflexivität $X \supseteq Y \quad \Rightarrow \quad X \to Y$
  - F2 Augmentation $\{X \to Y\} \quad \Rightarrow \quad XZ \to YZ$ sowie $XZ \to Y$
  - F3 Transitivität $\{X \to Y, Y \to Z\} \quad \Rightarrow \quad X \to Z$
  - F4 Dekomposition $\{X \to YZ\} \quad \Rightarrow \quad X \to Y$
  - F5 Vereinigung $\{X \to Y, X \to Z\} \quad \Rightarrow \quad X \to YZ$
  - F6 Pseudotransitivität $\{X \to Y, WY \to Z\} \quad \Rightarrow \quad WX \to Z$

- Bemerkungen
  - F1-F3 bekannt als Armstrong-Axiome (sound, complete, independent)
  - FDs wie in F1 werden auch triviale FDs genannt.
  - F4 wird auch Splitting Regel genannt, F5 Combining Regel

Vereinfachung von FD Mengen mittels der Regeln

- Gegeben sei eine Menge F von FDs. Wir formen diese in zwei Schritten um:
  - 1. Schritt: Anwendung der Splitting Regel F4: Ersetze jede FD

    $$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \in F$$

    durch die m FDs

    $$A_1, A_2, \ldots, A_n \rightarrow B_1$$
    $$A_1, A_2, \ldots, A_n \rightarrow B_2$$
    $$\ldots$$
    $$A_1, A_2, \ldots, A_n \rightarrow B_m$$

  - 2. Schritt: Elimination von trivialen FDs nach F1

    Entferne alle FDs $A_1, A_2, \ldots, A_n \rightarrow B \in F$ mit $B \in \{A_1, A_2, \ldots, A_n\}$ aus F

Wir können also von nun an für jede FD Menge F (implizit) annehmen, dass sie diese vereinfachte Form hat und nennen diesen Algorithmus SPLITTING(F)

Beispiel für die Vereinfachung von FDs

Betrachten wir erneut die Relation

- `RWEINE = {Name, Farbe, Jahrgang, Weingut, Anbaugebiet, Region}`
- Gegeben seien folgende FDs
  1) `Name, Jahrgang, Weingut` → `Anbaugebiet, Farbe`
  2) `Weingut` → `Anbaugebiet, Weingut, Region`
  3) `Anbaugebiet` → `Region`
- Vereinfachung Schritt 1 (Splitting):
  1a) `Name, Jahrgang, Weingut` → `Anbaugebiet`
  1b) `Name, Jahrgang, Weingut` → `Farbe`
  2a) `Weingut` → `Anbaugebiet`
  2b) `Weingut` → `Weingut`
  2c) `Weingut` → `Region`
  3) `Anbaugebiet` → `Region`
- Vereinfachung Schritt 2 (Elimination):
  Eliminiere 2b)
- Ergebnis:
  1) `Name, Jahrgang, Weingut` → `Anbaugebiet`
  2) `Name, Jahrgang, Weingut` → `Farbe`
  3) `Weingut` → `Anbaugebiet`
  4) `Weingut` → `Region`
  5) `Anbaugebiet` → `Region`

Hüllenbildung

- Als Hülle F^+ einer Menge funktionaler Abhängigkeiten F bezeichnen wir die Menge aller funktionaler Abhängigkeiten, die aus F abgeleitet werden können.

- Beispiel:
  $\{A \rightarrow B, B \rightarrow C\}^+ = \{$ $A \rightarrow B, B \rightarrow C,$
  $A \rightarrow C,$ F3
  $AB \rightarrow C, AB \rightarrow AC,$ F2
  $A \rightarrow A,$ F1
  …}

Hüllenbildung

- Als Hülle X_F^+ einer Menge von Attributen X bezüglich einer Menge funktionaler Abhängigkeiten F bezeichnen wir die Menge aller Attribute, die laut F von X funktional abhängen.

$$X_F^+ := \{ A \mid X \to A \in F^+ \}$$

  Wegen Regel F1 gilt: $X \subseteq X_F^+$

- Beispiele:

  Für $F=\{A\to B, B\to C\}$ über $R=\{A,B,C,D\}$ ist
  $\{A\}_F^+ = \{A,B,C\}$
  $\{B\}_F^+ = \{B,C\}$
  $\{A,B\}_F^+ = \{A,B,C\}$
  $\{C\}_F^+ = \{C\}$
  $\{D\}_F^+ = \{D\}$

Membership-Problem

- Membership-Problem: Gegeben sein eine Menge funktionaler Abhängigkeiten F. Kann $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$ aus F abgeleitet werden?

- Lösungsversuch 1:

  $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$ kann aus F genau dann abgeleitet werden, wenn $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \in F^+$

  - Problem dabei: F^+ enthält potentiell exponentiell viele FDs, d.h. jeder Algorithmus zur Berechnung von F^+ hat exponentielle Laufzeitkomplexität. Praktisch ist damit dieser Lösungsversuch nicht umsetzbar.

- Lösungsversuch 2:

  $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$ kann aus F genau dann abgeleitet werden, wenn $\{B_1, B_2, \ldots, B_m\}$ eine Teilmenge der Hülle von $\{A_1, A_2, \ldots, A_n\}$ bezüglich F ist: $\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}_F^+$

  - Algorithmus zur Berechnung von $\{A_1, A_2, \ldots, A_n\}_F^+$ in linearer Zeit existiert (CLOSURE-Algorithmus)!

Algorithmus CLOSURE - Idee

- Eingabe: *F*, vereinfachte Menge funktionaler Abhängigkeiten
  $\{A_1, A_2, \ldots, A_n\}$, Attribute

- Ausgabe: $\{A_1, A_2, \ldots, A_n\}_F^+$,Hülle der Attribute bzgl. F

- Algorithmus Idee:
  1) $H = \{A_1, A_2, \ldots, A_n\}$
  2) Für jede FD $B_1, B_2, \ldots, B_m \rightarrow C$ in F:
     Wenn alle $B_1, B_2, \ldots, B_m$ in H sind, aber C nicht in H ist, dann füge C zu H hinzu.
  3) Wiederhole Schritt 2, bis keine weitere Attribute mehr zu H hinzugefügt werden können.
  4) Nun enthält H die Hülle von $\{A_1, A_2, \ldots, A_n\}$ bzgl. F.

F:	A→B	B→C	C→D	H={A}
Schritt 1				H={A,B}
Schritt 2				H={A,B,C}
Schritt 3				H={A,B,C,D}

D ⊆ CLOSURE(F,{A}) ⇒
A→D aus F ableitbar.

Algorithmen CLOSURE und MEMBER

- Ermittlung von X_F^+: die Hülle von X bzgl. F

Algorithmus: **CLOSURE**(F, X):

```
H := X
repeat
    done = true
    forall FDs Y → C ∈ F
        if Y ⊆ H and C ∉ H then
            H = H ∪ {C}, done = false
until(done)
return H
```

- Membership Test

Algorithmus: **MEMBER**(F, X → Y): /* Test auf X → Y ∈ F⁺ */

```
return (Y ⊆ CLOSURE(F, X))
```

Beispiele

- Beispiel 2:
  - Relation
    `RWEINE={Name, Farbe, Jahrgang, Weingut, Anbaugebiet, Region}`
  - Gegebene FDs:
    - `Name, Jahrgang, Weingut` → `Anbaugebiet, Farbe`
    - `Anbaugebiet` → `Region`
  - Frage: Gilt die FD
    - `Name, Jahrgang, Weingut, Farbe` → `Region, Anbaugebiet`

- Algorithmus: Gilt
  `Region, Anbaugebiet` ⊆ CLOSURE(F,{`Name, Jahrgang, Weingut, Farbe`})?

F:	Name, Jahrgang, Weingut → Anbaugebiet, Farbe	Anbaugebiet → Region	H={Name, Jahrgang, Weingut, Farbe}
Schritt 1			H={Name, Jahrgang, Weingut, Farbe, Anbaugebiet}
Schritt 1			H={Name, Jahrgang, Weingut, Farbe, Anbaugebiet, Region}

Zusammenhang Funktionale Abhängigkeit und Schlüssel

- Eine Mengen von Attributen $K \subseteq R$ ist genau dann ein **Schlüssel** von R, wenn
  1. $K \rightarrow R \in F$
  2. Es gibt keine echte Teilmenge $X \subset K$ mit $X \rightarrow R$

  In Worten: die Attribute in K bestimmen eindeutig alle Attribute von R und sind minimal. Ohne (2) sprechen wir von einem **Oberschlüssel** (**Superkey**).

- Mittels MEMBER und CLOSURE können wir jetzt folgendes berechnen:
  1. K ist genau dann ein **Superkey** von R, wenn
     $K_F^+ = R \Leftrightarrow$
     R = CLOSURE(F,K) $\Leftrightarrow$
     MEMBER(F,$K \rightarrow R$)
  2. K ist genau dann ein **Schlüssel** von R, wenn zusätzlich für alle Teilmengen $X \subset K$ zusätzlich gilt
     $X_F^+ \neq R \Leftrightarrow$
     $R \neq$ CLOSURE(F,X) $\Leftrightarrow$
     not(MEMBER(F,$X \rightarrow R$))

Schlüsselbestimmung in der Praxis (1)

- Alle möglichen Attributkombinationen (Kandidaten) generieren, und mittels der MEMBER Tests wie gerade gezeigt, prüfen, ob sie Schlüssel sind
  - Prüfen ist in linearer Zeit möglich → ok
  - Aber es existieren exponentiell viele Attributkombinationen → nicht gangbar

- Einsatz von Heuristiken, um die Anzahl Kandidaten zu reduzieren
  1. FDs F vereinfachen, dadurch werden die rechten Seiten zu einelementigen Mengen und die trivialen und doppelten FDs entfernt.
  2. Alle Attribute $S \subseteq R$, die in keiner FD vorkommen, müssen im Schlüssel sein. Beweis siehe 3.
  3. Alle Attribute $S \subseteq R$, die auf keiner *rechten* Seite einer FD vorkommen, müssen im Schlüssel enthalten sein.
     Beweis: Wenn S auf keiner *(rechten)* Seite der FD vorkommt, dann gilt
     $$\forall K \subseteq \{R/S\}: S \notin CLOSURE(F, K),$$
     dann kann K kein Schlüssel sein, denn es gilt
     $$K_F^+ \neq R.$$
     Wiederspruch, S ist im Schlüssel.
  4.

Schlüsselbestimmung in der Praxis (2)

4. Test, ob die Hülle der gefundenen Attributmenge alle Attribute enthält
   1. Ja → Einziger Schlüssel

      Beweis:

      Annahme: S ist Schlüssel, keins der Elemente von S kommt auf der rechten Seite einer FD vor und es gibt einen Schlüssel $K \neq S$. Dann gilt

      $$\forall\ K \subseteq \{R/s\},\ s \in S:\ s \notin CLOSURE(F, K)$$

      dann kann K kein Schlüssel sein, denn es gilt

      $$K_F^+ \neq R.$$

      Wiederspruch, S ist einziger Schlüssel denn es gilt zusätzlich

      $$K_F^+ = R \text{ für } K = S \cup X \subseteq R,$$

      was K zu einem Superkey machen würde.
   2. Nein → Durchprobieren aller weiterer Kombinationen: erst ein weiteres Attribut hinzufügen, dann zwei, dann drei etc. bis alle Schlüssel gefunden wurden. Dabei natürlich nur die sinnvollen Tests durchführen.

- Algorithmus ist zwar immer noch exponentiell, aber in vielen Fällen durchführbar.

Schlüsselbestimmung Beispiel 1

- Schlüsselbestimmung für die Relation

  `RWEINE={Name,Farbe,Jahrgang,Weingut,Anbaugebiet,Region}`

  - mit den FDs F

    `Name, Jahrgang, Weingut` → `Anbaugebiet, Farbe`
    `Weingut` → `Anbaugebiet, Weingut, Region`
    `Anbaugebiet` → `Region`

1. Vereinfachung (siehe Beispiel zur Vereinfachung) ergibt
   `Name, Jahrgang, Weingut` → `Anbaugebiet`
   `Name, Jahrgang, Weingut` → `Farbe`
   `Weingut` → `Anbaugebiet`
   `Weingut` → `Region`
   `Anbaugebiet` → `Region`
2. Nicht vorkommende Attribute: keine
3. Nicht rechts vorkommende Attribute: `Name, Weingut, Jahrgang`
4. Berechnung der Hülle: $\{\texttt{Name, Weingut, Jahrgang}\}_F^+ =$
   $\{\texttt{Name, Weingut, Jahrgang, Farbe, Anbaugebiet, Region}\} =$ `RWEINE`
   → ist einziger Schlüssel!

➔ Schlüssel von `RWEINE`: {{`Name, Weingut, Jahrgang`}}

Schlüsselbestimmung Beispiel 2

- Schlüsselbestimmung für `R(A,B,C,D)` mit den FDs F

  `A, B, C` $\rightarrow$ `D`

  `D` $\rightarrow$ `A`

  1. Vereinfachung ergibt: keine Änderung an den FDs
  2. Nicht vorkommende Attribute: keine
  3. Nicht rechts vorkommende Attribute: `B, C`
  4. Berechnung der Hülle: $\{B, C\}_F^+ = \{B, C\} \neq R$ → $\{B, C\}$ ist KEIN Schlüssel. Dennoch muss $\{B, C\}$ in jedem Schlüssel enthalten sein. Durchprobieren:
     1. $\{B, C, A\}$: $\{B, C, A\}_F^+ = \{B, C, A, D\} = R$ → ist Schlüssel
     2. $\{B, C, D\}$: $\{B, C, D\}_F^+ = \{B, C, D, A\} = R$ → ist Schlüssel
     3. Falls die beiden ersten beide keine Schlüssel gewesen wären, müssten wir nun $\{B, C, A, D\}$ probieren. So wissen wir ohne weitere Tests, dass dies ein Superkey, aber kein Schlüssel ist.

  ➔ Schlüssel von `R: {{A, B, C}, {B, C, D}}`

Äquivalenz von Mengen von FDs

- Zwei Mengen von FDs F und G heißten äquivalent (geschrieben $F \equiv G$), wenn sie dieselbe Hülle haben: $F \equiv G$:= $F^+ = G^+$
  - D.h. wenn aus Ihnen dieselben FDs abgeleitet werden können
  - z.B. $\{A \to B, B \to C\} \equiv \{A \to B, B \to C, A \to C\}$

- Hülle F^+ zu einer FD Menge F ist eindeutig, aber sehr groß, daher unhandlich.
  - Mit je weniger FDs wir arbeiten müssen, umso effizienter wird der Entwurfsprozess und die daraus entstehende Datenbank
  - Zu einer gegebenen Mengen von FDs suchen wir also eine möglichst kleine, äquivalente Mengen von FDs
  - SPLITTING(F) ≡ F. Das ist schon mal einfacher, es geht aber noch besser.

Äquivalenz von Mengen von FDs, Beispiel

- Meist gibt es viele verschiedene Mengen von FDs, die äquivalent sind.
  - Beispiel: `R=(ABC)`.
    Die FDs F_1, F_2, F_3, F_4 sind alle äquivalent ($F_1 \equiv F_2 \equiv F_3 \equiv F_4$)

$$F_1 = \{ A \to B,\ B \to C,\ A \to C \}$$

$$F_2 = \{ A \to B,\ B \to C \}$$

$$F_3 = \{ A \to B,\ B \to C,\ AB \to C \}$$

$$F_4 = \{ A \to B,\ B \to BC,\ AC \to BC \}$$

weil sie alle dieselbe Hülle aufspannen:

$$F_1^+ = F_2^+ = F_3^+ = F_4^+ = \{ A \to B,\ B \to C,\ A \to C,\ A \to AB,\ A \to BC,\ A \to AC,\ A \to ABC,\ B \to BC,\ AB \to AC,\ AC \to BC,\ AB \to C,\ AB \to BC,\ + \text{triviale FDs}\}$$

Kanonische (minimale) Überdeckung

- Die kanonische (minimale) Überdeckung (canonical cover, minimal cover) F_c zu einer Menge von FDs F erfüllt folgende Bedingungen
  1) $F_c \equiv F$
  2) Die linke Seite jeder FD $f \in F_c$ ist einzigartig.
     Denn es ist ja $\{A \to X, A \to Y\} \equiv \{A \to XY\}$, doppelte linke Seiten lassen sich also leicht entfernen.
  3) Weder linke noch rechte Seite jeder FD $X \to Y \in F_c$ enthalten überflüssige Attribute:
     (a) Keine Linke Seite kann verkürzt werden
     $$\forall A \in X: (F_c - (X \to Y)) \cup ((X\text{-}A) \to Y)) \not\equiv F_c$$
     (b) Keine rechte Seite kann verkürzt werden
     $$\forall B \in Y: (F_c - (X \to Y)) \cup (X \to (Y\text{-}B))) \not\equiv F_c$$

  - Beispiel der vorigen Folie: F_2 ist eine kanonische Überdeckung für F_1, F_3 und F_4.

Algorithmus zur Berechnung der kanonischen Überdeckung

- Eingabe: Menge von FDs F, Ausgabe: kanonische Überdeckung F_c

- Algorithmus: **COVER**(F)
  1) Initialisiere $F_c = F$
  2) Vereinfache F_c = SPLITTING(F_c):

     Ersetze jede FD

     $$A_1, A_2, \ldots, A_n \rightarrow \boldsymbol{B_1, B_2, \ldots, B_m} \in F_c$$

     durch die m FDs

     $$A_1, A_2, \ldots, A_n \rightarrow \boldsymbol{B_1}$$

     $$\ldots$$

     $$A_1, A_2, \ldots, A_n \rightarrow \boldsymbol{B_m}$$

     und eliminiere triviale FDs.

Algorithmus zur Berechnung der kanonischen Überdeckung

3) Minimiere linke Seiten:

für jede FD

$A_1, A_2, \ldots, A_n \rightarrow B \in F_c$ und jedes i = 1, …, n:

falls B auch ohne ein A_i durch F_c erzeugt werden kann

$B \in$ CLOSURE(F_c,$\{A_1, \ldots, A_{i\text{-}1}, A_{i+1}, \ldots, A_n\}$) bzw.

$B \in \{A_1, \ldots, A_{i\text{-}1}, A_{i+1}, \ldots, A_n\}_{F_c}^{+}$

dann entferne A_i aus der FD

$F_c = F_c - \{A_1, A_2, \ldots, A_n \rightarrow B\} \cup \{A_1, \ldots, A_{i\text{-}1}, A_{i+1}, \ldots, A_n \rightarrow B\}$

4) Entferne überflüssige FDs:

für jede FD $X \rightarrow B \in F_c$ falls die FD auch ohne diese FD aus F_c erzeugt werden kann

$B \in \{X\}_{(Fc\ -\ (X \rightarrow B))}^{+}$,

dann entferne diese FD aus F_c

$F_c = F_c - \{X \rightarrow B\}$.

5) Zusammenfassung gleicher linker Seiten (SPLITTING(F_c) rückgängig):

Ersetze alle FDs $X \rightarrow \mathbf{B_1}, X \rightarrow \mathbf{B_2}, \ldots, X \rightarrow \mathbf{B_n} \in F_c$

durch die eine FD $X \rightarrow \mathbf{B_1}, \mathbf{B_2}, \ldots, \mathbf{B_n}$

Beispiel zur Berechnung einer kanonischen Überdeckung

1. $F_4 = F_c = \{ A \to B,\ B \to BC,\ AC \to BC \}$
2. $SPLITTING(F_c) = \{ A \to B,\ B \to C,\ AC \to B \}$
3. Minimierung linker Seiten nur für $AC \to B$ nötig.
   (1) Test ob $B \in \{C\}_{F_c}^{+} = \{C\}$. Nein, also kein Änderung an F_c
   (2) Test ob $B \in \{A\}_{F_c}^{+} = \{A, B, C\}$. Ja, also
   $F_c = \{ A \to B,\ B \to C,\ \mathbf{A} \to \mathbf{B} \}$
4. Entfernen überflüssiger FDs:
   (1) für $A \to B$: Test ob $B \in \{A\}_{\{B \to C\}}^{+} = \{A\}$. Nein, also keine Änderung.
   (1) für $B \to C$: Test ob $C \in \{B\}_{\{A \to B\}}^{+} = \{B\}$. Nein, also keine Änderung.
5. Zusammenfassung linker Seiten:
   Keine gleichen linken Seiten vorhanden.
   6. $F_c = \{ A \to B,\ B \to C \} = F_2$

Dekomposition von Relationen zur Redundanzvermeidung

- Wenn wir „zu viel“ in ein Relationenschema packen, erzeugen wir Redundanzen. Diese führen zu Anomalien (Änderungs- und Löschanomalien)

- Funktionale Abhängigkeiten erlauben uns, unser Wissen über die Anwendungsdomäne zu formalisieren. FDs können wir umformen. Aus FDs können wir die Schlüssel einer Relation berechnen.

→ Dekomposition = Zerlegung von Relationenschemata mit Hilfe der FDs, so dass die resultierenden Relationenschemata <u>keine</u> Redundanzen mehr enthalten.

- Ziel: Vollständige Vermeidung von Redundanzen - jedoch, ohne gleichzeitig
  - semantische Informationen zu verlieren (Abhängigkeitstreue)
  - die Möglichkeit zur Rekonstruktion der Relationen zu verlieren (Verbundtreue)

  (Wir werden jedoch sehen, dass diese Ziele nicht immer vollständig erreichbar ist.)

Normalformen und Normalisierung

- Normalformen
  - Legen Eigenschaften von Relationenschemata fest
  - Verbieten bestimmte Kombinationen von funktionalen Abhängigkeiten in Relationen

- Normalisierung
  - Der Prozess, ein Mengen von Relationenschemata, die nicht in einer gewünschten Normalform sind, in eine neue Menge von Relationenschemata zu überführen, die alle in der gewünschten Normalform sind

Erste Normalform 1NF

- Eine Relation R entspricht der ersten Normalform, wenn
  1. Jedes Attribut hat einen atomaren Wertebereich
  2. R ist frei von Wiederholungsgruppen

- Vorteil bei der Sortierbarkeit und Bearbeitung

Wiederholungsgruppen

R:

Fahrzeug	Ausführungen
BMW 1er	3ZB, 4ZB,..
BMW 3er	R6ZB,R6ZD,..
Tata Estate	1ZG
Tata Sierra	4ZE

Nicht atomar: Marke, Name

R:

Hersteller	Name	Ausführung
BMW	1er	3ZB
BMW	1er	4ZB
BMW	3er	R6ZB
BMW	3er	R6ZD
Tata	Estate	1ZG
Tata	Sierra	4ZE

Zweite Normalform 2NF

- Eine Relation R entspricht der zweiten Normalform, wenn
  1. R entspricht der ersten Normalform
  2. Jedes Attribut, das nicht Teil eines Schlüssels ist, hängt vom ganzen Schlüssel ab, und nicht nur von einer echten Teilmenge.

- Vorteil: Monothematische Tabellen

R:

Marke	Name	Abgasnorm	Zentrale
BMW	1er	Euro5A	München
BMW	3er	Euro4	München
Tata	Estate	Euro2	Mumbai
Tata	Sierra	Euro1	Mumbai

R1:

Marke	Name	Abgasnorm
BMW	1er	Euro5A
BMW	3er	Euro4
Tata	Estate	Euro2
Tata	Sierra	Euro1

R2:

Marke	Zentrale
BMW	München
Tata	Mumbai

Klar: Die Redundanz in der Firmenzentrale ist unvorteilhaft: Speicherverschwendung und es kann zu Änderungs- und Löschanomalien kommen.

Aber: wie überprüfen wir das zweite Kriterium?

Zweite Normalform 2NF

R:

Marke	Name	Abgasnorm	Zentrale
BMW	1er	Euro5A	München
BMW	3er	Euro4	München
Tata	Estate	Euro2	Mumbai
Tata	Sierra	Euro1	Mumbai

F: $\{Marke,Name\} \rightarrow \{Abgasnorm, Zentrale\}$
$Marke \rightarrow Zentrale$
$\{Marke,Name\} \rightarrow Abgasnorm$

1. Schlüssel über Heuristik:
i) Attribute die in keiner FD? Nein (wären sonst im Schlüssel)
ii) Alle Attribute die auf keiner rechten Seite sind, sind im Schlüssel {Marke,Name}

2. Test Schlüssel
i) $\{Marke,Name\}_F^+ = R$
ii) Damit ist {Marke,Name} einziger Schlüssel
iii) Zum Spaß: auch nicht verkürzbar?
$\{Marke\}_F^+ = \{Marke,Zentrale\} \neq R$
$\{Name\}_F^+ = \{Name\} \neq R$, passt!

3. F_C
i) SPLIT(F)= $\{Marke,Name\} \rightarrow Abgasnorm$
$\{Marke,Name\} \rightarrow Zentrale$
$Marke \rightarrow Zentrale$
ii) Verkürze $\{Marke,Name\} \rightarrow Abgasnorm$
$Abgasnorm \notin \{Name\}_F^+$
$Abgasnorm \notin \{Marke\}_F^+$
iii) Verkürze $\{Marke,Name\} \rightarrow Zentrale$
$Zentrale \notin \{Name\}_F^+$
$Zentrale \in \{Marke\}_F^+$
$\Rightarrow F_c = F \setminus \{Marke,Name\} \rightarrow Zentrale$
$\cup \; Marke \rightarrow Zentrale$
iv) $F_C =$ $\{Marke,Name\} \rightarrow Abgasnorm$
$Marke \rightarrow Zentrale$

Verletzt 2NF Bedingung: Jedes Attribut, das nicht Teil eines Schlüssels ist (Zentrale), hängt vom ganzen Schlüssel ab, und nicht nur von einer echten Teilmenge (Marke).

Dritte Normalform 3NF

- Eine Relation R entspricht der dritten Normalform, wenn
  1. R entspricht der zweiten Normalform
  2. Kein nicht Schlüsselattribut hängt transitiv von einem Schlüssel ab. Oder anders: jede linke Seite ist Superkey oder die rechte Seite ist prim.

- Vorteil: Monothematische Tabellen

R:

Marke	Name	Abgasnorm	CO
BMW	1er	Euro5A	1000
BMW	3er	Euro4	1000
Tata	Estate	Euro2	2200
Tata	Sierra	Euro1	2720

F = {Marke,Name}→Abgasnorm
Abgasnorm→CO

CO hängt transitiv von einem Schlüssel ab

R1:

Marke	Name	Abgasnorm
BMW	1er	Euro5A
BMW	3er	Euro4
Tata	Estate	Euro2
Tata	Sierra	Euro1

R2:

Abgasnorm	CO
Euro5A	1000
Euro4	1000
Euro3	2300
Euro2	2200
Euro1	2720

Dritte Normalform 3NF

R:

Marke	Name	Abgasnorm	CO
BMW	1er	Euro5A	1000
BMW	3er	Euro4	1000
Tata	Estate	Euro2	2200
Tata	Sierra	Euro1	2720

F: {Marke,Name}→Abgasnorm
Abgasnorm→CO

1. Schlüssel über Heuristik:
i) Attribute die in keiner FD? Nein (wären sonst im Schlüssel)
ii) Alle Attribute die auf keiner rechten Seite sind, sind im Schlüssel {Marke,Name}

2. Test Schlüssel
i) $\{Marke,Name\}_F^+ = R$
ii) Damit ist {Marke,Name} einziger Schlüssel
iii) Zum Spaß: auch nicht verkürzbar?
$\{Marke\}_F^+ = \{Marke,Zentrale\} \neq R$
$\{Name\}_F^+ = \{Name\} \neq R$, passt!

3. F_C
SPLIT(F)= {Marke,Name}→Abgasnorm
Abgasnorm→CO

i) Verkürze {Marke,Name}→Abgasnorm
$Abgasnorm \notin \{Name\}_F^+$
$Abgasnorm \notin \{Marke\}_F^+$
ii) F_C= {Marke,Name}→Abgasnorm
Abgasnorm→CO

Abgasnorm ist kein Superkey und
CO ist auch nicht prim → Verletzt 3NF

Alternative Bedingung zu 3NF

- Dritte Normalform (3NF)
  - Relation R mit Menge von vereinfachten FDs F ist in 3NF genau dann, wenn die linke Seite jeder FD ein Superkey von R ist oder die rechte Seite prim ist.
  - Formal: Relation R mit vereinfachten Menge von FDs F ist in 3NF genau dann, wenn:

    $\forall X \rightarrow A \in F^{+}$ gilt: $X \rightarrow$ (A trivial $\vee$) X ist Superkey von $R \vee A$ ist prim
  - Ein Attribut heißt prim, wenn es Teil eines Schlüssels ist.
  - Jedes Relationenschema in BCNF ist automatisch auch in 3NF

Wie bestimmt man praktisch, ob eine Relation in 3NF ist?

- Gegeben: eine Relation R mit einer FD Menge F

- Berechne die kanonische Überdeckung F_c von F.
  R ist in 3NF genau dann, wenn die linken Seiten aller FDs in F_c Superkeys sind oder die rechte Seite prim ist.

- Problem dabei: um zu testen, ob ein Attribut prim ist (d.h. Teil eines Schlüssels ist), müssen wir alle Schlüssel von R kennen. Diese zu berechnen benötigt jedoch, wie gesehen, im worst-case exponentielle Zeit!
  - In der Praxis jedoch meist möglich.

Dekomposition: Normalisierung von Relationen in 3NF

1) Gegeben: Menge von Relationenschema Z + vereinfachte Menge von FDs
   - Im einfachsten Fall nur einer Relation R ist $Z = \{ R \}$.

2) Solange es noch ein Relationenschema $S \in Z$ gibt, das nicht in 3NF ist:
   - Finde eine für S geltende nicht-triviale FD $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$, die die 3NF verletzt (d.h. $\{A_1, \ldots, A_n\}$ ist kein Superkey von S oder mind. ein B_i ist nicht prim)
   - Berechne $S_1 = \{A_1, \ldots, A_n\}^+$ und $S_2 = S - S_1 \cup \{A_1, \ldots, A_n\}$
   - Entferne S aus Z und füge S_1 und S_2 in Z ein, d.h. $Z = Z - \{S\} \cup \{S_1, S_2\}$
   - Ordne die FDs den (neu entstandenen) Relationen zu

3) Z ist nun eine 3NF konforme Zerlegung der ursprünglichen Schemas

- Bemerkungen
  - S_1 ist jeweils die maximale, von $\{A_1, \ldots, A_n\}$ funktional bestimmte Attributmenge (die natürlich sowohl $\{A_1, \ldots, A_n\}$ als auch $\{B_1, \ldots, B_m\}$ enthält)
  - S_2 ist jeweils die Menge aller übrigen Attribute vereinigt mit $\{A_1, \ldots, A_n\}$

3NF Zerlegung

SPLIT(F)= {Marke,Name}→Abgasnorm
Abgasnorm→CO

Abgasnorm ist kein Superkey und CO ist auch nicht prim → Verletzt 3NF

$S_1 = \{A_1, \ldots, A_n\}^+$ und $S_2 = S - S_1 \cup \{A_1, \ldots, A_n\}$

$S_1 = \{Abgasnorm\}^+$ und $S_2 = S - \{Abgasnorm, CO\} \cup \{Abgasnorm\}$

R:

Marke	Name	Abgasnorm	CO
BMW	1er	Euro5A	1000
BMW	3er	Euro4	1000
Tata	Estate	Euro2	2200
Tata	Sierra	Euro1	2720

R1:

Marke	Name	Abgasnorm
BMW	1er	Euro5A
BMW	3er	Euro4
Tata	Estate	Euro2
Tata	Sierra	Euro1

R2:

Abgasnorm	CO
Euro5A	1000
Euro4	1000
Euro3	2300
Euro2	2200
Euro1	2720

Boyce-Codd-Normalform (BCNF)

- Eine Relation R entspricht der Boyce-Codd Normalform, wenn
  1. R entspricht der dritten Normalform
  2. Jede Attributmenge von der andere Attribute funktional abhängen ist ein Superschlüssel.

- Vorteil: Monothematische Tabellen

$F = \{Marke, Name\} \rightarrow Abgasnorm$
$\{Marke, Name, Abgasnorm\} \rightarrow Skandal$
$\{Abgasnorm, Skandal\} \rightarrow Schwere$

R:

Marke	Name	Abgasnorm	Skandal	Schwere
ACME	313er	Euro5A	nein	keine
ACME	Benny	Euro4	ja	gering
Umbrella	Van	Euro2	nein	keine
Umbrella	Truck	Euro1	ja	hoch

$\{Marke,Name\}_F^+ = R$ ist Schlüssel.

R1:

Marke	Name	Abgasnorm	Skandal
ACME	313er	Euro5A	nein
ACME	Benny	Euro4	ja
Umbrella	Van	Euro2	nein
Umbrella	Truck	Euro1	ja

Aber: Schwere hängt ab von

R2:

Abgasnorm	Skandal	Schwere

{Abgasnorm,Skandal} und das ist kein (Super)Schlüssel.

Wie bestimmt man praktisch, ob eine Relation in BCNF ist?

- Gegeben: eine Relation R mit einer FD Menge F

- Berechne die kanonische Überdeckung F_c von F.
  R ist in BCNF genau dann, wenn die linken Seiten aller FDs in F_c Superkeys sind.

- Häufig ist eine einfachere Alternative, um zu zeigen, dass R nicht in BCNF ist, ein Gegenbeispiel anzugeben!
  - Beispiel: Relation `RWEINE` mit den FDs
    - `Name, Jahrgang, Weingut → Anbaugebiet, Farbe`
      `Weingut → Anbaugebiet, Weingut, Region`
      `Anbaugebiet → Region`
    - Schlüssel von `RWEINE` sind (wie bereits bestimmt): {{`Name,Weingut,Jahrgang`}}
    - `RWEINE` ist nicht in BCNF, da z.B. die die linke Seite der FD `Anbaugebiet→Region` (also {`Anbaugebiet`}) kein Superkey ist.

- Alle Relationen mit nur 2 Attributen sind in BCNF.
  - Beweis: siehe CompleteBook

Beispiel BCNF Bestimmung

$F = \{Marke, Name\} \rightarrow Abgasnorm$
$\quad \{Marke, Name, Abgasnorm\} \rightarrow Skandal$
$\quad \{Abgasnorm, Skandal\} \rightarrow Schwere$

$F_c = SPLIT(F) = F$

$Abgasnorm \notin CLOSURE(F, Marke) = \{Marke\}$
$Abgasnorm \notin CLOSURE(F, Name) = \{Name\}$
$Skandal \in CLOSURE(F, \{Marke, Name\}) = \{Marke, Name, Abgasnorm, Skandal, Schwere\}$
$\quad \Rightarrow F_c = F \setminus \{Marke, Name, Abgasnorm\} \rightarrow Skandal$
$\quad \quad \cup \{Marke, Name\} \rightarrow Skandal$
$Skandal \notin CLOSURE(F, \{Marke, Abgasnorm\}) = \{Marke, Abgasnorm\}$
$Skandal \notin CLOSURE(F, \{Name, Abgasnorm\}) = \{Name, Abgasnorm\}$
$Schwere \notin CLOSURE(F, Abgasnorm) = \{Abgasnorm\}$
$Schwere \notin CLOSURE(F, Skandal) = \{Skandal\}$

$F_c = \{Marke, Name\} \rightarrow \{Abgasnorm, Skandal\}$
$\quad \{Abgasnorm, Skandal\} \rightarrow Schwere$

Berechne die kanonische Überdeckung Fc von F.
R ist in BCNF genau dann, wenn die linken Seiten aller FDs in Fc Superkeys sind.

Dekomposition: Normalisierung von Relationen in BCNF (1)

1) Gegeben: Menge von Relationenschema Z und eine vereinfachte Menge von FDs

2) Solange es noch ein Relationenschema $R \in Z$ gibt, das nicht in BCNF ist:

- Finde eine für R geltende nicht-triviale FD $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$, die die BCNF verletzt (d.h. $\{A_1, \ldots, A_n\}$ ist kein Superkey von S)
- Berechne $S_1 = \{A_1, \ldots, A_n\}^+$ und $S_2 = (R - S_1) \cup \{A_1, \ldots, A_n\}$
- Entferne R aus Z und füge S_1 und S_2 in Z ein, d.h. $Z = Z - \{S\} \cup \{S_1, S_2\}$
- Ordne die FDs den (neu entstandenen) Relationen zu

F_c = {Marke, Name}→Abgasnorm
{Marke, Name}→Skandal
⚡{Abgasnorm, Skandal}→Schwere

S_1 = $\{\text{Abgasnorm,Skandal}\}_F^+$
= {Abgasnorm,Skandal,Schwere}

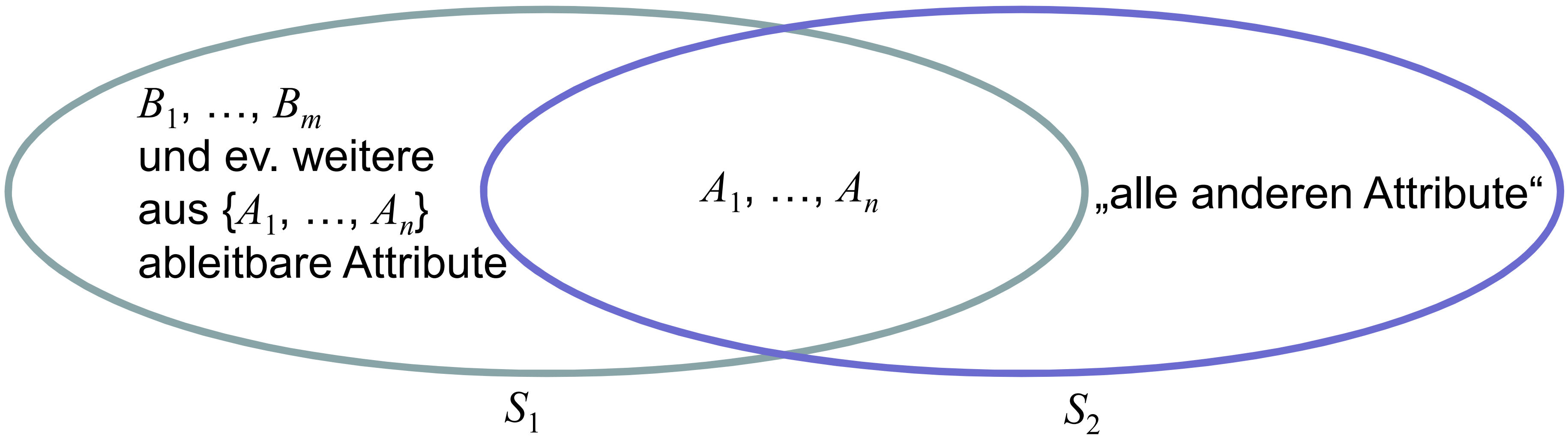
S_2 = R-S_1 $\cup$ {Abgasnorm,Skandal}
={Marke,Name,Abgasnorm,Skandal}

3) Z ist nun eine BCNF konforme Zerlegung der ursprünglichen Schemas

Dekomposition: Normalisierung von Relationen in BCNF (2)

- Bei der Zerlegung können wir drei Attributmengen unterscheiden:
  1) $\{A_1, \ldots, A_n\}$: die linke Seite der die BCNF verletzenden FD
  2) $\{A_1, \ldots, A_n\}^+$: die Hülle der linken Seite der die BCNF verletzenden FD. Diese enthält natürlich die komplette rechte Seite $\{B_1, \ldots, B_m\}$ dieser FD, und auch die komplette linke Seite (beide sind trivialerweise mittels der FD aus $\{A_1, \ldots, A_n\}$ ableitbar), sowie potentiell weitere aus $\{A_1, \ldots, A_n\}$ mittels anderer FDs ableitbare Attribute
  3) Alle anderen Attribute.

- Visuelle Darstellung der Zerlegung von S



Beispiel für die Dekomposition in BCNF

- Betrachte
  - Relation ERZEUGER = {Weingut, Anbaugebiet, Region} mit den FDs
    - FD1: Weingut → Anbaugebiet
    - FD2: Anbaugebiet → Region
  - Bestimmung der Schlüssel von R ergibt {{Weingut}}

- Dekompositionsalgorithmus
  1) Z = {ERZEUGER}
  2) - ERZEUGER ist nicht in BCNF wegen der FD Anbaugebiet → Region, weil {Anbaugebiet} kein Superschlüssel ist.
     - ERZEUGER1 = $\{$Anbaugebiet$\}^+$ = {Anbaugebiet,Region}
     - ERZEUGER2 = ERZEUGER – ERZEUGER1 ∪ {Anbaugebiet} =
       = {Weingut,Anbaugebiet,Region} – {Anbaugebiet,Region} ∪ {Anbaugebiet}
       = {Weingut, Anbaugebiet}
     - FD1 wird ERZEUGER2 zugeordnet (nur dort sind alle Attribute aus FD1 vorhanden) und FD2 wird ERZEUGER1 zugeordnet (nur dort sind alle Attribute aus FD2 vorhanden)
     - Z = {ERZEUGER1, ERZEUGER2}, alle sind in BCNF
  3) Somit ist {ERZEUGER1, ERZEUGER2} eine BCNF konforme Zerlegung

Forderungen an eine Zerlegung / Transformation

- Für Zerlegung (Transformation) eines Relationenschemas wünschen wir uns generell drei wichtige Eigenschaften:

1) Eliminierung der Anomalien

- In den resultierenden Relationenschemas sollen keine Anomalien mehr auftreten

2) Verbundtreue

- Genau die Tupel (Anwendungsdaten) der ursprünglichen Relation sollen sich aus den Tupeln der zerlegten Relationenschemas wieder herleiten lassen

3) Abhängigkeitstreue

- Die funktionalen Abhängigkeiten, die aus den Schlüsseln der zerlegten Relationen abgeleitet werden können, sollten äquivalent zu den ursprünglichen FDs sein

Verbundtreue

- Zur Erfüllung des Kriteriums der Normalformen werden Relationenschemata in kleinere Relationenschemata zerlegt

- Für Beschränkung auf „sinnvolle“ Zerlegungen gilt Forderung, dass die Originalrelation wieder aus den zerlegten Relationen mit dem natürlichen Verbund zurückgewonnen werden kann
  - Das muss für jede Datenbank gelten, die die FDs erfüllt!

➔ Verbundtreue

Verbundtreue – Beispiel 1

- Gegeben

  R = ABC mit

  $F = \{A \rightarrow B,\ C \rightarrow B\}$

- Dekomposition in

  $R_1 = AB$ und $R_2 = BC$

- Ist ***nicht*** verbundtreu:

A	B	C
1	2	3
4	2	5

A	B
1	2
4	2

B	C
2	3
2	5

A	B	C
1	2	3
4	2	5
1	2	5
4	2	3

Verbundtreue – Beispiel 2

- Gegeben

  R = ABC mit
  $F = \{A \rightarrow B, \; B \rightarrow C\}$

- Dekomposition in

  R_1 = AB und R_2 = BC

- Ist verbundtreu:

A	B	C
1	2	3
4	2	3

A	B
1	2
4	2

B	C
2	3

A	B	C
1	2	3
4	2	3

Verbundtreue - Formal

- Formal: Die Dekomposition einer Attributmenge

$$X \; in \; X_1, \ldots, X_p \text{ mit } X = \bigcup\nolimits^{p}_{i=1} X_i$$

heißt verbundtreu ($\pi \bowtie$-treu, lossless) bezüglich einer Menge von FDs F genau dann, wenn für alle Relationen $r(X)$, die die FDs F erfüllen gilt:

$$\pi_{X_1}(r) \bowtie \ldots \bowtie \pi_{X_p}(r) = r$$

  - Bemerkung: Verlieren kann man durch die Projektions-Verbund-Abfolge keine Tupel, aber man könnte zusätzliche Tupel erhalten. Warum dann „lossless“? Weil man die Information, welche Tupel ursprünglichen da waren, verloren hat!
  - Dekomposition ist “lossy” wenn R1 ⋈ R2 ⊃ R (Beispiel 1)
  - Dekomposition ist “lossless” wenn R1 ⋈ R2 = R (Beispiel 2)

Verbundtreue - Formal

- Einfaches Kriterium für Verbundtreue bei Dekomposition in zwei Relationenschemata: Dekomposition von

$$X \text{ in } X_1 \text{ und } X_2$$

ist verbundtreu bzgl. F, wenn die Schnittmenge der Attribute für mindestens eine der beiden entstehenden Relationen ein Superkey ist:

$$(X_1 \cap X_2) \rightarrow X_1 \in F^+ \text{ oder}$$
$$(X_1 \cap X_2) \rightarrow X_2 \in F^+$$

F_c = {Marke, Name}→Abgasnorm
{Marke, Name}→Skandal
⚡{Abgasnorm, Skandal}→Schwere

S_1 = $\{\text{Abgasnorm,Skandal}\}_F^+$
= {Abgasnorm,Skandal,Schwere}

S_2 = R-S_1 ∪ {Abgasnorm,Skandal}
={Marke,Name,Abgasnorm,Skandal}

(S_1 ∩ S_2) = {Abgasnorm,Skandal}
$\{\text{Abgasnorm,Skandal}\}_F^+ = S_1$
Superkey, immer so gewählt

BCNF Zerlegung immer Verbundtreu!

Abhängigkeitstreue

- Ausgangspunkt war ein Relationenschema R mit einer Menge von FDs F
- Nach der Zerlegung (Transformation) gibt es R nicht mehr, statt dessen eine Mengen von Relationenschemata $\{R_1, \ldots, R_n\}$, wobei jedes R_i einige, aber nicht unbedingt alle, der Attribute von R enthält.
- Daher kann es in F FDs geben, die in keiner R_i komplett überprüft werden können, da sie Attribute enthalten, die in keiner R_i zusammen vorkommen
  - Man könnte natürlich R durch Joins aus den R_i berechnen, und dann die FDs überprüfen, aber das ist viel zu aufwändig
- Wir definieren die Schlüsselabhängigkeiten F_{R_i} auf R_i als die Menge aller FDs aus F^+, die nur Attribute aus R_i enthalten und deren linke Seite ein Schlüssel für R_i ist.
- Eine Zerlegung ist abhängigkeitstreu genau dann, wenn
  $F \equiv F_{R_1} \cup \ldots \cup F_{R_n}$ (bzw. $F^+ = (F_{R_1} \cup \ldots \cup F_{R_n})^+$)

Abhängigkeitstreue – Beispiel

- Zerlegung der Relation
  - `ERZEUGER = {Weingut, Anbaugebiet, Region}` mit
    - `Weingut → Anbaugebiet`
    - `Anbaugebiet → Region`

A	B	C
1	2	3
4	2	3

- in die Relationen
  - `E1` = {`Anbaugebiet, Region`} mit Schlüsseln {{`Anbaugebiet`}} und
  - `E2` = {`Weingut, Anbaugebiet`} mit Schlüsseln {{`Weingut`}}

A	B
1	2
4	2

B	C
2	3

- Damit ist
  - F_{E1}= {`Anbaugebiet → Region`} $\cup$ triviale FDs
  - F_{E2}= {`Weingut → Anbaugebiet`} $\cup$ triviale FDs
- Offensichtlich gilt $F^{+} = (F_{E1} \cup F_{E2})^{+}$ da
  $F = F_{E1} \cup F_{E2}$
- Diese BCNF-Dekomposition ist somit abhängigkeitstreu.

Nicht abhängigkeitstreue Dekomposition in BCNF – Beispiel (1)

Relation `PLZ-VERZEICHNIS` mit Attributen `Straße, Ort, Bundesland, PLZ`

- Orte werden durch Namen und Bundesland eindeutig identifiziert
- Innerhalb einer Straße ändert sich die PLZ nicht
- PLZ Gebiete gehen nicht über Ortsgrenzen und Orte nicht über Bundeslandgrenzen hinweg
- Damit ergeben sich die FDs F:
  - `PLZ → Ort, Bundesland`
  - `Straße, Ort, Bundesland → PLZ`
- Daraus ergeben sich als Schlüssel für die Relation `PLZ-Verzeichnis`:
  - {{`Straße, Ort, Bundesland`}, {`Straße, PLZ`}}

Ist `PLZ-VERZEICHNIS` in BCNF?

- Alle linken Seiten von F_C sind Superkeys?
  - F_C=SPLIT(F)={PLZ→Ort, PLZ→Bundesland, {Straße, Ort, Bundesland}→PLZ}
  - PLZ $\notin$ $\{\text{Straße,Ort}\}_F^+$, PLZ $\notin$ $\{\text{Straße,Bundesland}\}_F^+$, PLZ $\notin$ $\{\text{Bundesland,Ort}\}_F^+$,
  - F_C=F.
- Nein, PLZ ist kein Superkey. Daher verletzt `PLZ → Ort, Bundesland` BCNF

Quelle: Kemper/Eickler: Datenbanksysteme

Nicht abhängigkeitstreue Dekomposition in BCNF – Beispiel (1)

- Anwendung des Dekompositionsalgorithmus für FD
  `PLZ → Ort, Bundesland`
  - $S1 = \{PLZ\}_F^{+} = \{PLZ, Ort, Bundesland\}$,
  - $S2 = R - S1 \cup PLZ = \{Straße, PLZ\}$
  - Es ergeben sich die Relationenschemas
    ORTE = {PLZ, Ort, Bundesland} und
    STRASSEN = {PLZ, Straße}

- Anomalien sind beseitigt, neues Schema ist verbundtreu (nach dem einfachen Kriterium: Schnittmenge ist {PLZ}, und das ist Schlüssel für ORTE) ABER die FD

  Straße, Ort, Bundesland → PLZ

  ist keiner Relation mehr zugeordnet. Nicht Abhängigkeitstreu!

Quelle: Kemper/Eickler: Datenbanksysteme

Nicht abhängigkeitstreue Dekomposition in BCNF – Beispiel (2)

Problem der verlorenen FD Straße, Ort, Bundesland $\rightarrow$ PLZ:

PLZ-VERZEICHNIS

Ort	Bundesland	Straße	PLZ
Frankfurt	Hessen	Göthestraße	60313
Frankfurt	Hessen	Galgenstraße	60437
Frankfurt	Brandenburg	Göthestraße	15234

$\pi_{PLZ,Straße}$ $\pi_{Ort,Bundesland,PLZ}$

STRASSEN

PLZ	Straße
60313	Göthestraße
60437	Galgenstraße
15234	Göthestraße
einfügen von: **15235**	**Göthestraße**

ORTE

Ort	Bundesland	PLZ
Frankfurt	Hessen	60313
Frankfurt	Hessen	60437
Frankfurt	Brandenburg	15234
einfügen von: **Frankfurt**	**Brandenburg**	**15235**

ORTE $\bowtie$ STRASSEN

Ort	Bundesland	Straße	PLZ
Frankfurt	Hessen	Göthestraße	60313
Frankfurt	Hessen	Galgenstraße	60437
Frankfurt	Brandenburg	Göthestraße	15234
Frankfurt	Brandenburg	Göthestraße	15235

zur FD!

Quelle: Kemper/Eickler: Datenbanksysteme

Transformationsforderungen und Dekomposition in BCNF

- Transformationsforderungen und die BCNF-Dekomposition
  1) Eliminierung der Anomalien
     - Erfüllt: keine Redundanz-erzeugenden FDs mehr in den zerlegten Relationen.
  2) Verbundtreue
     - Erfüllt: Zerlegung in S1 und S2 ist genau so gewählt, dass {A1, …, An} die Schnittmenge der beiden und gleichzeitig ein Schlüssel von S1 ist
       ➔ Einfache Bedingung für die Verbundtreue ist erfüllt.
  3) Abhängigkeitstreue
     - <u>Nicht immer erfüllt</u>: Gegenbeispiel gerade gesehen.

- Praktische Erfahrung zeigt jedoch: BCNF Dekomposition ist fast immer abhängigkeitstreu!
  - Wenn jedoch nicht: erzeugtes DB Schema i.allg. nicht akzeptabel, alle FDs müssen vom erzeugten DB Schema garantiert werden.
  - Frage: Gibt es eine andere Normalform, die alle drei Forderungen erfüllt?
  - Antwort: <u>Nein</u>, aber es gibt die 3NF, diese erfüllt 2) und 3), aber 1) dafür nicht.

Abhängigkeitstreue in der Praxis

Wie bestimmt man in der Praxis, ob eine Zerlegung abhängigkeitstreu ist?

- Gegeben
  - eine Relation R mit einer FD Menge F.
  - eine Zerlegung von R in n Relationen $R_1, \ldots, R_n$ jeweils mit Schlüsseln $\{K_{i1,} \ldots, K_{i,m\_i}\}$.
- Sie $G = \{K_{ij} \rightarrow R_i\}$ die Menge aller aus den Schlüsseln erzeugten FDs.
- Die Zerlegung ist genau dann abhängigkeitstreu, wenn man jede FD $A_1, \ldots, A_n \rightarrow B \in F$ aus den FDs G ableiten kann.
  - Oder konkret: wenn die Hülle $\{A_1, \ldots, A_n\}_G^+$ das Attribut B enthält (Member-Test)

Die Normalformen und ihre Bedingungen

1NF	Jedes Attribut hat einen atomaren Wertebereich R ist frei von Wiederholungsgruppen
2NF	Jedes Attribut, das nicht Teil eines Schlüssels ist, hängt vom ganzen Schlüssel ab, und nicht nur von einer echten Teilmenge.
3NF	Kein nicht Schlüsselattribut hängt transitiv von einem Schlüssel ab.
BCNF	Jede Attributmenge von der andere Attribute funktional abhängen ist ein Superschlüssel.

Every non-key attribute must provide a fact about the key (1NF), the whole key (2NF), and nothing but the key (3NF), so help me Codd (and Boyce for all (BCNF)).

Vergleich zwischen 3NF und BCNF Dekomposition

- Der bisher kennen gelernte 3NF Dekompositionsalgorithmus läuft in exponentieller Zeitkomplexität, da hierzu die Menge aller Schlüssel bestimmt werden muss → praktisch schlecht einsetzbar.

- Der daraus abgewandelte Version zur Dekomposition in BCNF läuft in polynomieller Zeit, ist aber nicht immer abhängigkeitstreu.

- Daher jetzt bessere Version: Syntheseverfahren.

Syntheseverfahren - Idee

- Prinzip:
  Synthese formt Original-FD-Menge F in resultierende Menge von Schlüsselabhängigkeiten G so um, dass $F \equiv G$ gilt
  - Abhängigkeitstreue ist im Verfahren verankert
  - Verbundtreue ist ebenfalls im Verfahren verankert
  - 3NF wird immer erreicht (häufig sogar BCNF – muss aber jeweils geprüft werden)

- Zeitkomplexität: quadratisch

Syntheseverfahren - Algorithmus

- Syntheseverfahren: Ablauf
  - Geg.: Relationschema R mit FDs F
  - Ges.: verlustfreie & abhängigkeitstreue Zerlegung in $R_1, \ldots, R_n$, wobei alle R_i in 3NF sind

- Algorithmus: **`SYNTHESIZE`**(R, F):
  1) Berechne die kanonische Überdeckung F_c von F
  2) Für jede linke Seite einer $A_1, \ldots, A_n$ einer FD in F_c, erzeuge eine Relation mit den Attributen $\{A_1, \ldots, A_n\} \cup \{ B \mid A_1, \ldots, A_n \rightarrow B \in F_c\}$
  3) Falls keine der erzeugten Relationen ein Superkey von R ist, erzeuge eine weitere Relation die aus den Attributen eines Schlüssels von R besteht.
  4) Entferne jede Relation, die vollständig in einer anderen enthalten ist.

Beispiel Syntheseverfahren (1)

Relation `PLZ-VERZEICHNIS` mit Attributen `Straße, Ort, Bundesland, PLZ`

- $F_C = F$:
  - `PLZ` $\rightarrow$ `Ort, Bundesland`
  - `Straße, Ort, Bundesland` $\rightarrow$ `PLZ`

Ist `PLZ-VERZEICHNIS` in BCNF?

- Alle linken Seiten von F_C sind Superkeys?
  - F_C=SPLIT(F)={PLZ→Ort, PLZ→Bundesland, {Straße, Ort, Bundesland}→PLZ}
  - PLZ $\notin$ $\{Straße,Ort\}_F^+$, PLZ $\notin$ $\{Straße,Bundesland\}_F^+$, PLZ $\notin$ $\{Bundesland,Ort\}_F^+$,
  - F_C=F.
- Nein, PLZ ist kein Superkey. Daher verletzt `PLZ` $\rightarrow$ `Ort, Bundesland` BCNF

Syntheseverfahren ergibt Zerlegung:

- R1={PLZ,Ort,Bundesland}
- R2=R={Straße, Ort, Bundesland, PLZ} (Sinnlos, da keine Zerlegung)

Beispiel Syntheseverfahren (2)

R:

Marke	Name	Abgasnorm	Zentrale
BMW	1er	Euro5A	München
BMW	3er	Euro4	München
Tata	Estate	Euro2	Mumbai
Tata	Sierra	Euro1	Mumbai

R war nicht 3NF

i) F_C= {Marke,Name}→Abgasnorm
Marke→Zentrale

ii) R1:

Marke	Name	Abgasnorm
BMW	1er	Euro5A
BMW	3er	Euro4
Tata	Estate	Euro2
Tata	Sierra	Euro1

R2:

Marke	Zentrale
BMW	München
BMW	München
Tata	Mumbai
Tata	Mumbai

iii) {Marke, Name, Abgasnorm} ist Superkey von R

iv) Es ist keine Relation vollständig in einer anderen enthalten. → Fertig.

Beispiel Syntheseverfahren (3)

- Beispiel
  - Relation R mit Attributen ABCE
  - FD-Menge F = {A $\rightarrow$ B, AB $\rightarrow$ C, A $\rightarrow$ C, B $\rightarrow$ A, C $\rightarrow$ E }

1) Kanonische Überdeckung: F_c = {A $\rightarrow$ B, B $\rightarrow$ C, B $\rightarrow$ A, C $\rightarrow$ E }

2) Erzeugte Relationen: {AB, BCA, CE}

3) {AB} ist Superkey für ABCE (da $\{AB\}^+ = \{ABCE\}$) somit keine weitere Relation nötig

4) Elimination der ersten erzeugten Relation, da komplett in zweiter enthalten.

➔ Ergebnis der Synthese in 3NF: {ABC, CE}
(Bem: Test auf BCNF mittels CLOSURE Algorithmus: Schema ist auch in BCNF)

Syntheseverfahren - Erreichung der Verbundtreue

- Bisher: Erreichen der Verbundtreue durch Schritt 3 im Algorithmus
  - Test, ob eine Relation ein Superkey ist: polynomial
  - Wenn jedoch nicht: Schlüssel bestimmen ist exponentiell!
  - Daher ist das Syntheseverfahren in dieser Form exponentiell!

- Verbesserung: Erreichen der Verbundtreue durch einfachen „Trick":
  - Anstatt von Schritt 3
  - Erweitern der Original-FD-Menge F um $R \rightarrow \delta$ wobei δ ein Dummy-Attribut ist
  - δ wird nach Synthese aus allen Relationen und FDs entfernt

- Beispiel: `R=ABCE`, F={`A` $\rightarrow$ `B`, `C` $\rightarrow$ `E` }
  - Syntheseergebnis {`AB`, `CE`} ist nicht verbundtreu, da weder `AB` noch `CE` Superschlüssel sind (einziger Schlüssel ist `AC`)
  - Dummy-FD `ABCE` $\rightarrow \delta$ in F wird zu `AC` $\rightarrow \delta$ in F_c
  - Und liefert drittes Relationenschema {`AC`}, damit verbundtreue Synthese in 3NF mittels {`AB`, `CE`, `AC`}

Zusammenfassung

- Funktionale Abhängigkeiten
- Schlüssel
- Heuristik zur Schlüsselbestimmung
- Member-Test durch CLOSURE
- Gleichheit von FDs durch COVER
- 1NF, 2NF, 3NF, BCNF zur Vermeidung von Anomalien
- Dekomposition
- Syntheseverfahren