Teil 11: Einführung in relationale Normalformen

Literatur:

- Elmasri/Navathe: Fundamentals of Database Systems, 3.Aufl., Kap. 14, “Functional Dependencies and Normalization for Relational Databases” Kap. 15, “Relational Database Design Algorithms and Further Dependencies”
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3.Aufl., Kap. 7, “Relational Database Design”
- Ramakrishnan/Gehrke: Database Management Systems, 2.Aufl., Mc-Graw Hill, 2000. Kap. 15, “Schema Refinement and Normal Forms”
- Simsion/Witt: Data Modeling Essentials, 2.Auflage. Coriolis, 2001. Kap. 2: “Basic Normalization”, Kap. 8: “Advanced Normalization”.
- Kemper/Eickler: Datenbanksysteme, Oldenbourg, 1997. Kap. 6, “Relationale Entwurfstheorie”
- Rauh/Stickel: Konzeptuelle Datenmodellierung. Teubner, 1997.
- Kent: A Simple Guide to Five Normal Forms in Relational Database Theory. Communications of the ACM 26(2), 120–125, 1983.
- Thalheim: Dependencies in Relational Databases. Teubner, 1991.
- Lipeck: Skript zur Vorlesung Datenbanksysteme, Univ. Hannover, 1996.

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Schlechte relationale Datenbank-Entwürfe (die Redundanzen enthalten) erkennen.

  Eigentlich sollten Sie Redundanzen / Normalisierungsprobleme schon im konzeptionellen Entwurf im ER-Modell erkennen.
- Funktionale Abhängigkeiten bestimmen.
- Boyce-Codd-Normalform definieren und erklären.
- Eine gegebene Tabelle für gegebene funktionale Abhängigkeiten auf BCNF untersuchen.
- BCNF durch Aufspalten der Relation herstellen.

Inhalt

Einführung (1)

- Die Theorie des relationalen DB-Entwurfs basiert auf einer Klasse von Integritätsbedingungen, die “Funktionale Abhängigkeiten” (FAen) heißen. Sie sind Verallgemeinerungen von Schlüsseln.
- Diese Theorie definiert, wann eine Relation für eine gegebene Menge von FAen in einer bestimmten Normalform ist (z.B. 3. Normalform, 3NF).
- Meist ist es schlecht, wenn ein Schema Relationen enthält, die eine Normalform verletzen.

  Es gibt aber Ausnahmen und Kompromisse.

Einführung (2)

- Wenn eine Normalform verletzt ist, werden Daten redundant gespeichert, und Informationen über verschiedene Konzepte vermischt. Z.B.:

VORLESUNGEN			
VNR	TITEL	DNAME	TELEFON
22268	Datenbanken I	Brass	24740
42232	Grundlagen des WWW	Brass	24740
31822	Rechnerarchitektur	Molitor	24710

- Die Telefonnummer eines Dozenten wird einmal für jede von ihr/ihm gehaltene Vorlesung gespeichert (die Nummer von "Brass" hier also zweimal).

Einführung (3)

- Natürlich ist es kein Problem, wenn eine Spalte einen Wert mehrfach enthält (z.B. J/N-Spalte).
- Aber im Beispiel gilt: Haben zwei Zeilen den gleichen Wert in der Spalte DNAME, so müssen sie auch den gleichen Wert in der Spalte TELEFON haben.
- Dies entspricht der funktionalen Abhängigkeit: DNAME $\longrightarrow$ TELEFON.
- Aufgrund dieser Bedingung ist einer der beiden Einträge in der Spalte TELEFON für Brass redundant.

Einführung (4)

- Tabelleneinträge sind redundant, wenn sie aus anderen Tabelleneinträgen und zusätzlicher Information (wie den FAen) hergeleitet werden können.

  Z.B. Wenn eine Angestelltentabelle die Geburtsdaten enthält, dann wäre die zusätzliche Spalte ALTER redundant: Das Alter kann aus dem Geburtsdatum (bei Kenntnis des heutigen Datums) berechnet werden.

- Formaler: In einem DB-Zustand $\mathcal{I}_1$ ist ein Tabelleneintrag e redundant, wenn es keinen DB-Zustand $\mathcal{I}_2$ gibt, der ebenfalls alle IBen erfüllt, und sich nur im Wert für e von $\mathcal{I}_1$ unterscheidet.

  Im Beispiel kann man nicht nur eine der beiden Telefonnummern von Brass ändern: Dann wäre "DNAME $\longrightarrow$ TELEFON" nicht mehr erfüllt.

Einführung (5)

- Redundante Information im konzeptionellen Schema ist schlecht:
  - ◇ Speicherplatz wird verschwendet.

    Gilt nur bedingt: Durch Normalisierung auch mehr Speicherplatz.
  - ◇ Doppelter Aufwand für Dateneingabe.
  - ◇ Wenn die Information aktualisiert wird, müssen auch alle redundanten Kopien aktualisiert werden. Vergißt man eine, so werden die Kopien inkonsistent. (Update Anomalie).

    Merkwürdig: Um ein elementares Fakt zu ändern (die Telefonnummer von Brass), muß man mehrere Tabelleneinträge ändern.

Einführung (6)

- Manchmal ist es für leichtere Anfrageformulierung bequemer, redundante Information zu haben.
  Z.B. ein vorberechneter Verbund.
- Aber in relationalen Datenbanken kann man virtuelle Tabellen (Sichten) definieren.
  Durch Anfrage definiert, kann selbst wie Tabelle verwendet werden.
- Da der Inhalt einer Sicht nicht explizit gespeichert und nicht direkt aktualisiert wird, ist redundante Information für Sichten kein Problem.
  Die gesamte Sicht ist redundant, da sie aus gespeicherten Tabellen berechnet wird.

Einführung (7)

- Manchmal wird redundante Information auch für effiziente Anfrageauswertung benötigt.
- Es gibt einen Zielkonflikt ("Tradeoff"): Speichern von redundanten Informationen ist schlecht, aber langsame Anfrageauswertung ist auch schlecht.
- Das Zufügen redundanter Information sollte nur beim physischen Entwurf diskutiert werden und es muss gute Gründe dafür geben.
- Vermeiden Sie das Speichern redundanter Daten!

Einführung (8)

- In dem Beispiel werden Informationen über die beiden Konzepte "Vorlesung" und "Dozent" in einer Tabelle vermischt. Das ist schlecht:
  - ◇ Die Telefonnummer eines neuen Dozenten kann in der Tabelle nur zusammen mit einer Vorlesung gespeichert werden (Einfügeanomalie).
    Nullwerte helfen hier nicht: `VNR` ist Primärschlüssel (`NOT NULL`).
  - ◇ Die Löschung der letzten Vorlesung eines Dozenten führt zum Verlust seiner Telefonnummer, (Löschanomalie).
    Die Löschung der vorletzten Vorlesung verhält sich ganz anders.

Einführung (9)

- Wenn ein guter ER-Entwurf ins relationale Modell übersetzt wird, so sollten alle Normalformen erfüllt sein (i.w. Teil der Definition von "guter Entwurf").
- Normalformen sind dann nur eine Möglichkeit zur Kontrolle des Schemas.
  Wenn man ein Problem mit der Normalisierung feststellt, sollte man aber nicht nur das relationale Schema verbessern, sondern auch rückwirkend das ER-Schema korrigieren.
- Die Theorie relationaler Normalformen erlaubt es aber auch, aus Attributen und FAen Tabellen (also ein relationales Schema) zu konstruieren.

Einführung (10)

- Da Normalformen etwas Formales und Objektives sind, eignen sie sich gut als Argument bei eventuellen Diskussionen in einem Entwurfs-Team.

  Verletzung einer Normalform muß zumindest gut begründet werden.

- Heutzutage wird die 3. Normalform (3NF) als Teil der Datenbank-Allgemeinbildung angesehen.
- Boyce-Codd Normalform (BCNF) ist etwas strenger, aber leichter zu definieren und intuitiver.

  In dieser Vorlesung steht BCNF im Vordergrund.

Einführung (11)

- Intuitiv bedeutet BCNF, daß alle FAen schon durch Schlüssel erzwungen werden (nach dem Test auf Normalisierung kann man die FAen vergessen).

  D.h. eine Tabelle ist in BCNF gdw. alle FAen auf dieser Tabelle von Schlüsseln logisch impliziert werden.

- Wichtig, weil man beim "`CREATE TABLE`" in SQL nur Schlüssel deklarieren kann, keine allgemeinen FAen.
- Allgemein ist Ziel der Normalisierung (neben der Vermeidung von Redundanzen / Anomalien) auch die Eliminierung schwer zu überwachender IBen.

Inhalt

Funkt. Abhängigkeiten (1)

- Eine funktionale Abhängigkeit (FA) legt fest, daß ein Attribut (oder Attributkombination) eindeutig ein anderes Attribut/Attributkombination bestimmt.
- FAen schreibt man in der Form

$$A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m.$$

  Dabei sind die A_i und B_j Attribute einer Relation R.

  In der Normalformtheorie konzentriert man sich immer auf eine einzige Relation. Alle Attribute stammen aus dieser Relation. Deswegen wird die Relation in der Notation von funktionalen Abhängigkeiten nicht explizit angegeben.

Funkt. Abhängigkeiten (2)

- Die FA $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ auf der Relation R ist nur eine Abkürzung für die Formel
  $$\forall\, R\, X,\ R\, Y\colon\ \ X.A_1 = Y.A_1 \wedge \cdots \wedge X.A_n = Y.A_n \rightarrow X.B_1 = Y.B_1 \wedge \cdots \wedge X.B_m = Y.B_m$$
- Die FA bedeutet also: Wenn zwei Zeilen die gleichen Werte in $A_1, \ldots, A_n$ haben, dann müssen sie auch in $B_1, \ldots, B_m$ übereinstimmen.
- Deswegen können linke und rechte Seite der FA als Mengen von Attributen aufgefasst werden: Reihenfolge und Vielfachheit sind nicht relevant.

Funkt. Abhängigkeiten (3)

- Die FA $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ ist äquivalent zu den m FAen:
  $$\begin{array}{ccc} A_1, \ldots, A_n & \longrightarrow & B_1 \\ \vdots & \vdots & \vdots \\ A_1, \ldots, A_n & \longrightarrow & B_m. \end{array}$$
- Man beachte, daß "$\longrightarrow$" nicht der Implikationspfeil ist. Man liest $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ korrekt als "$A_1, \ldots, A_n$ bestimmen [funktional | eindeutig] $B_1, \ldots, B_m$".

  Und keineswegs "$A_1, \ldots, A_n$ implizieren $B_1, \ldots, B_m$". Die logische Implikation ist nur zwischen Formeln definiert, nicht zwischen Attributen.

Funkt. Abhängigkeiten (4)

- Wie oben bemerkt, gilt in diesem Beispiel die funktionale Abhängigkeit "`DNAME` $\longrightarrow$ `TELEFON`":

VORLESUNGEN			
VNR	TITEL	DNAME	TELEFON
22268	Datenbanken I	Brass	24740
42232	Grundlagen des WWW	Brass	24740
31822	Rechnerarchitektur	Molitor	24710

- Haben zwei Zeilen den gleichen Dozentennamen, so müssen sie auch die gleiche Telefonnummer haben.

  Haben zwei Zeilen nicht den gleichen Wert für `DNAME`, so ist die Bedingung für sie nichtig (automatisch erfüllt).

Funkt. Abhängigkeiten (5)

- Ein Schlüssel bestimmt jedes Attribut eindeutig, d.h. die FAen "`VNR` $\longrightarrow$ `TITEL`", "`VNR` $\longrightarrow$ `DNAME`", "`VNR` $\longrightarrow$ `TELEFON`" sind trivialerweise erfüllt:
  - ◇ Es gibt keine zwei verschiedenen Zeilen, die den gleichen Wert für einen Schlüssel haben (`VNR`).
  - ◇ Wenn also zwei Zeilen X und Y im Schlüssel (`VNR`) übereinstimmen, müssen sie identisch sein, d.h. in allen anderen Attributen übereinstimmen.
- Anstelle der drei FAen oben kann man die einzelne FA "`VNR` $\longrightarrow$ `TITEL, DNAME, TELEFON`" schreiben.

Funkt. Abhängigkeiten (6)

- Im Beispiel-Zustand ist die FA "DNAME $\longrightarrow$ TITEL" nicht erfüllt: Es gibt zwei Zeilen mit gleichem Dozentennamen, aber verschiedenen Werten für TITEL.
- Im Beispiel ist die FA "TITEL $\longrightarrow$ VNR" erfüllt.
- FAen sind, wie Schlüssel, Integritätsbedingungen: Sie müssen in allen möglichen DB-Zuständen gelten, nicht nur in einem Beispielzustand.

  Wenn natürlich eine FA im Beispielzustand nicht erfüllt ist, ist klar, daß sie auch allgemein nicht erfüllt sein kann. Z.B. braucht die FA "DNAME $\longrightarrow$ TITEL" nicht weiter betrachtet zu werden.

Funkt. Abhängigkeiten (7)

- Es ist Aufgabe des Datenbankentwurfs festzulegen, welche FAen gelten sollten. Dies kann nicht automatisch entschieden werden. Die FAen werden als Eingabe für den Test auf Normalisierung benötigt.
- Im Beispiel muß der DB-Entwerfer herausfinden, ob es vorkommen kann, daß zwei Vorlesungen mit dem gleichen Titel angeboten werden (z.B. parallele Veranstaltungen einer überbelegten Vorlesung).
- Wenn das vorkommen kann, so gilt die FA "TITEL $\longrightarrow$ VNR" im Allgemeinen nicht.

FAen vs. Schlüssel

- FAen sind Verallgemeinerungen von Schlüsseln: $A_1, \ldots, A_n$ ist Schlüssel von $R(A_1, \ldots, A_n, B_1, \ldots, B_m)$, gdw. die FA "$A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$" gilt.

  Unter der Annahme, daß es keine Duplikatzeilen gibt. Zwei verschiedene Zeilen, die in jedem Attribut identisch sind, würden die FAen nicht verletzen, aber den Schlüssel. In der Theorie kann dies nicht vorkommen, da Relationen Tupelmengen sind, und Tupel nur durch ihre Attributwerte definiert werden. In der Praxis erlaubt SQL zwei identische Zeilen in einer Tabelle solange kein Schlüssel definiert ist.
- Sind für eine Relation FAen gegeben, so kann man einen Schlüssel berechnen, indem man eine Menge von Attributen $A_1, \ldots, A_n$ findet, die alle anderen Attribute bestimmen.

Triviale FAen

- Eine funktionale Abhängigkeit $\alpha \longrightarrow \beta$ mit $\beta \subseteq \alpha$ nennt man trivial.
- Beispiele sind:
  - ◇ TITEL $\longrightarrow$ TITEL
  - ◇ DNAME, TELEFON $\longrightarrow$ TELEFON
- Triviale funktionale Abhängigkeiten sind automatisch immer erfüllt (in jedem DB-Zustand, egal ob es andere Bedingungen gibt oder nicht).

  Triviale Abhängigkeiten entsprechen in der Logik einer Tautologie.
- Triviale Abhängigkeiten sind uninteressant.

Implikation von FAen (1)

- VNR $\longrightarrow$ TELEFON ist nichts neues, wenn man schon VNR $\longrightarrow$ DNAME und DNAME $\longrightarrow$ TELEFON kennt.

  Immer wenn $A \longrightarrow B$ und $B \longrightarrow C$ gilt, gilt automatisch auch $A \longrightarrow C$.

- Eine Menge von FAen $\{\alpha_1 \longrightarrow \beta_1, \ldots, \alpha_n \longrightarrow \beta_n\}$ impliziert eine FA $\alpha \longrightarrow \beta$ genau dann, wenn in jedem DB-Zustand, in dem $\alpha_i \longrightarrow \beta_i$ für $i = 1, \ldots, n$ gilt, auch $\alpha \longrightarrow \beta$ gilt.

  α und β stehen hier für Mengen von Attributen/Spalten.

  Dies ist genau die aus Kapitel 3 bekannte Definition der logischen Implikation. Wenn man also die Notation $\alpha \longrightarrow \beta$ nur als Abkürzung für eine logische Formel auffasst, ist die obige Definition überflüssig.

Implikation von FAen (2)

- Normalerweise interessiert man sich nicht für alle geltenden FAen, sondern nur für eine repräsentative Teilmenge, die alle anderen FAen impliziert.
- Implizierte Abhängigkeiten können durch Anwendung der Armstrong Axiome berechnet werden:
  - ◇ Ist $\beta \subseteq \alpha$, dann gilt $\alpha \longrightarrow \beta$ trivial (Reflexivität).
  - ◇ Gilt $\alpha \longrightarrow \beta$, dann $\alpha \cup \gamma \longrightarrow \beta \cup \gamma$ (Erweiterung).
  - ◇ Gilt $\alpha \longrightarrow \beta$ und $\beta \longrightarrow \gamma$, dann auch $\alpha \longrightarrow \gamma$ (Transitivität).

Implikation von FAen (3)

- Die Anwendung der Armstrong-Axiome erfordert Geschick oder Probieren. Zum Test, ob eine Menge $\mathcal{F}$ von FAen eine FA $\alpha \longrightarrow \beta$ impliziert, benutzt man einfacher die Attribut-Hülle.
- Die Attribut-Hülle α^+ einer Menge von Attributen α ist die Menge aller Attribute B, die eindeutig durch α bestimmt sind (bzgl. der geg. FAen $\mathcal{F}$).

  $\alpha^+ := \{B \mid \text{Die geg. FAen } \mathcal{F} \text{ implizieren } \alpha \longrightarrow B\}.$

  Natürlich hängt α^+ von $\mathcal{F}$ ab. Ggf. schreibe man $\alpha^+_{\mathcal{F}}$.

- $\mathcal{F}$ impliziert $\alpha \longrightarrow \beta$ gdw. $\beta \subseteq \alpha^+_{\mathcal{F}}$.

Implikation von FAen (4)

- Die Attribut-Hülle α^+ wird wie folgt berechnet:

  Eingabe: α (Menge von Attributen)
  $\mathcal{F} = \{\alpha_1 \longrightarrow \beta_1, \ldots, \alpha_n \longrightarrow \beta_n\}$ (FAen)

  Ausgabe: α^+ (Menge von Attributen)

  Methode: $x := \alpha$;
  while x hat sich geändert **do**
    for each FA $\alpha_i \longrightarrow \beta_i$ **do**
      if $\alpha_i \subseteq x$ **then**
        $x := x \cup \beta_i$;
  output x;

Implikation von FAen (5)

- Beispiel: Gegeben seinen folgende FAen:

$$\texttt{ISBN} \longrightarrow \texttt{TITEL, VERLAG}$$
$$\texttt{ISBN, NR} \longrightarrow \texttt{AUTOR}$$
$$\texttt{VERLAG} \longrightarrow \texttt{VERLAG\_URL}$$

- Es sei jetzt $\{\texttt{ISBN}\}^+$ zu berechnen.
- Wir starten mit $x = \{\texttt{ISBN}\}$.

  x ist die Menge von Attributen, für die wir wissen, daß sie nur einen einzelnen Wert annehmen können. Wir beginnen mit der Annahme, daß es für das gegebene Attribut in α, d.h. ISBN, nur einen Wert gibt. Dann ist die Hülle α^+ die Menge von Attributen, für die wir unter dieser Annahme ableiten können, daß ihr Wert eindeutig bestimmt ist (unter Verwendung der gegebenen FAen).

Implikation von FAen (6)

- Die erste der gegebenen FAen, nämlich

$$\texttt{ISBN} \longrightarrow \texttt{TITEL, VERLAG}$$

  hat eine linke Seite (ISBN), die in der aktuellen Menge x ($x = \{\texttt{ISBN}\}$) enthalten ist.

  D.h. es gibt einen eindeutigen Wert für dieses Attribut. Dann bedeutet die FA, daß auch die Attribute auf der rechten Seite einen eindeutigen Wert haben.

- Damit können wir x durch die Attribute der rechten Seite der FA, d.h. TITEL und VERLAG, erweitern:

$$x = \{\texttt{ISBN}, \texttt{TITEL}, \texttt{VERLAG}\}.$$

Implikation von FAen (7)

- Nun kann man die dritte der FAen, nämlich

$$\texttt{VERLAG} \longrightarrow \texttt{VERLAG\_URL},$$

  angewenden, da die linke Seite in x enthalten ist.

- Also kann man die rechte Seite der FA zu x zufügen

$$x = \{\texttt{ISBN}, \texttt{TITEL}, \texttt{VERLAG}, \texttt{VERLAG\_URL}\}.$$

- Die letzte FA, nämlich

$$\texttt{ISBN, NR} \longrightarrow \texttt{AUTOR}$$

  kann auch jetzt noch nicht angewendet werden, da NR in x fehlt.

Implikation von FAen (8)

- Nach erneutem Prüfen, ob die Menge x noch durch die gegebenen FAen erweitert werden kann, bricht der Algorithmus ab, und gibt aus:

$$\{\texttt{ISBN}\}^+ = \{\texttt{ISBN}, \texttt{TITEL}, \texttt{VERLAG}, \texttt{VERLAG\_URL}\}.$$

- Daher gilt, daß z.B. "ISBN $\longrightarrow$ VERLAG_URL" durch die gegebenen FAen inpliziert wird.
- Auf die gleiche Weise kann man z.B. die Attribut-Hülle von $\{\texttt{ISBN}, \texttt{NR}\}$ berechnen. Das ist die Menge aller Attribute der Relation.

  Das bedeutet, daß $\{\texttt{ISBN}, \texttt{NR}\}$ ein Schlüssel der Relation ist.

Inhalt

Schlüssel bestimmen (1)

- Mit einer geg. Menge von FAen (und der Menge aller Attribute $\mathcal{A}$ einer Relation), kann man alle möglichen Schlüssel für diese Relation bestimmen.

  Auch hier müssen Duplikatzeilen ausgeschlossen sein.

- $\alpha \subseteq \mathcal{A}$ ist ein Schlüssel gdw. $\alpha^+ = \mathcal{A}$.
- Es sind aber nur minimale Schlüssel interessant.

  Die Obermenge eines Schlüssels ist wieder ein Schlüssel, z.B. wenn $\{\texttt{ISBN}, \texttt{NR}\}$ alle anderen Attribute eindeutig identifiziert, gilt dies auch für $\{\texttt{ISBN}, \texttt{NR}, \texttt{TITEL}\}$. Somit benötigt man zusätzlich die Bedingung, daß jedes $A \in \alpha$ notwendig sein muss, d.h. $(\alpha - \{A\})^+ \neq \mathcal{A}$. Bei den meisten Autoren ist die Minimalitätsbedingung Teil der Schlüsseldefinition. Dann ist ein Schlüssel aber nicht nur eine Integritätsbedingung, sondern bedeutet auch, daß stärkere Bedingungen nicht gelten.

Schlüssel bestimmen (2)

- Halb-intuitive Bestimmung der Schlüssel:
  - ◇ Man beginnt mit der Menge der Attribute, die auf keiner rechten Seite stehen.

    Diese Attribute sind notwendig in jedem Schlüssel enthalten. Im Beispiel ist man damit schon fertig: `ISBN` und `NR` tauchen auf keiner rechten Seite auf, aber ihre Hülle ist die Menge aller Attribute.

  - ◇ Bilden diese Attribute noch keinen Schlüssel, so fügt man weitere Attribute hinzu: Die linke Seite einer FA oder direkt ein fehlendes Attribut.

    Man muss nur sicherstellen, daß die Menge am Ende auch minimal ist. Falls Attribute enthalten sind, die funktional durch andere Attribute der Menge bestimmt werden, entfernt man sie.

Schlüssel bestimmen (3)

- Wenn man eine Obermenge $\mathcal{S}$ eines Schlüssels hat (z.B. die Menge aller Attribute $\mathcal{A}$), kann man mit folgendem Algorithmus eine Teilmenge $\mathcal{K} \subseteq \mathcal{S}$ bestimmen, die ein minimaler Schlüssel ist:
  - ◇ Man startet mit $\mathcal{K} := \mathcal{S}$. Entsprechend der Voraussetzung gilt $\mathcal{K}^+ = \mathcal{A}$ (Menge aller Attribute).
  - ◇ Man probiert einfach in irgendeiner Reihenfolge alle Attribute $A \in \mathcal{S}$ durch: Wenn man A weglassen kann, ohne die Schlüssel-Eigenschaft zu zerstören, d.h. $(\mathcal{K} - \{A\})^+ = \mathcal{A}$, dann streicht man A, setzt also $\mathcal{K} := \mathcal{K} - \{A\}$.

Schlüssel bestimmen (4)

Algorithm `minimize`($\mathcal{S}$): (mit globalen Variablen $\mathcal{A}$, $\mathcal{F}$)

$\mathcal{K} := \mathcal{S}$;
assert $\mathcal{S}^+_{\mathcal{F}} = \mathcal{A}$; /* S muss Schlüssel sein */
foreach $A \in \mathcal{A}$ **do**
  if $(\mathcal{K} - \{A\})^+_{\mathcal{F}} = \mathcal{A}$ **then**
    $\mathcal{K} := \mathcal{K} - \{A\}$; /* A ist nicht nötig */
return $\mathcal{K}$; /* Minimaler Schlüssel */

Aufgabe/Beispiel:

- $\mathcal{F} = \{A \longrightarrow B,\ B \longrightarrow A,\ A \longrightarrow C,\ D \longrightarrow C\}$.

  $\mathcal{A} := \{A, B, C, D\}$, $\mathcal{S} := \mathcal{A}$. Probieren Sie in der Reihenfolge A, B, C, D, die Attribute wegzulassen. Probieren Sie dann D, C, B, A.

Schlüssel bestimmen (5)

- Das obige Verfahren liefert nur einen Schlüssel.

  Wenn man nicht alle Reihenfolgen probieren will.

- Oft braucht man aber alle (minimalen) Schlüssel.
- Im schlechtesten Fall kann es exponentiell viele verschiedene minimale Schlüssel geben, z.B.

$$\begin{aligned} \mathcal{A} &:= \{A_i \mid i = 1, \ldots, n\} \cup \{B_i \mid i = 1, \ldots, n\} \\ \mathcal{F} &:= \{A_i \longrightarrow B_i \mid i = 1, \ldots, n\} \cup \\ & \{B_i \longrightarrow A_i \mid i = 1, \ldots, n\}. \end{aligned}$$

- Die Frage, ob ein gegebenes Attribut A in einem minimalen Schlüssel enthalten ist, ist NP-vollständig.

Schlüssel bestimmen (6)

- Man kann aber in polynomieller Zeit den jeweils nächsten Schlüssel berechnen.
- Für jeden Schlüssel $\mathcal{K}$ und FA $\alpha \longrightarrow \beta$ ist $\alpha \cup (\mathcal{K} - \beta)$ auch ein Schlüssel (nicht unbedingt minimal).

  Vorher galt ja $\mathcal{K}^+ = \mathcal{A}$. Wenn man β durch α ersetzt, bekommt man β gleich im ersten Schritt der Attributhüllenbestimmung zurück.

- Man muss also nur prüfen, ob einer der bekannten Schlüssel eine Teilmenge von $\alpha \cup (\mathcal{K} - \beta)$ ist.
- Wenn das nicht der Fall ist, wendet man den obigen Minimierungsalgorithmus auf diese Menge an und bekommt einen weiteren Schlüssel.

Schlüssel bestimmen (7)

Satz:

- Sei $\mathcal{A}$ die Menge aller Attribute, $\mathcal{F}$ die Menge der FAen und $\mathcal{K}_1, \ldots, \mathcal{K}_n$, $n \geq 1$ eine Menge von minimalen Schlüsseln bezüglich $\mathcal{A}$ und $\mathcal{F}$.
- Dann gibt es einen weiteren minimalen Schlüssel $\mathcal{K}'$ nicht in $\{\mathcal{K}_1, \ldots, \mathcal{K}_n\}$ gdw. es ein $i \in \{1, \ldots, n\}$ gibt und eine FA $\alpha \longrightarrow \beta \in \mathcal{F}$ so dass $\alpha \cup (\mathcal{K}_i - \beta)$ keine Obermenge eines der $\mathcal{K}_1, \ldots, \mathcal{K}_n$ ist.

  Der zusätzliche Schlüssel $\mathcal{K}'$ ist eine Teilmenge von $\alpha \cup (\mathcal{K}_i - \beta)$.
  Dieser Satz ist eine leicht umformulierte Version von Lemma 4 in:
  Cláudio L. Lucchesi, Sylvia L. Osborn: Candidate keys for relations. Journal of Computer and System Sciences 17:2, Oct. 1978, 270–279.

Übung

- Die folgende Relation dient zu Speicherung von Bestellungen (Aufträgen einer Versandhandlung):

```
BESTELLT(AUFTRAGS_NR, DATUM, KUND_NR,
         PROD_NR, PROD_BEZEICHNUNG, MENGE)
```

- Geben Sie alle funktionalen Abhängigkeiten an, die für diese Relation gelten.

  In einem Auftrag können mehrere Produkte bestellt werden.

- Implizieren diese FAen die folgende FA?

  $\texttt{AUFTRAGS\_NR, PROD\_NR} \longrightarrow \texttt{DATUM}$

- Bestimmen Sie einen Schlüssel der Relation.

Inhalt

Motivation (1)

- Man betrachte erneut das Beispiel

VORLESUNGEN			
VNR	TITEL	DNAME	TELEFON
22268	Datenbanken I	Brass	24740
42232	Grundlagen des WWW	Brass	24740
31822	Rechnerarchitektur	Molitor	24710

- Wie oben erklärt, wird die Telefonnummer von Dozenten, die mehrere Vorlesungen halten, in dieser Tabelle mehrfach abgespeichert.
- Dies folgt aus der FA: $\texttt{DNAME} \longrightarrow \texttt{TELEFON}$.

Motivation (2)

- Die FA "$\texttt{DNAME} \longrightarrow \texttt{TELEFON}$" führt genau dann zu Redundanzen, wenn es mehrere Zeilen mit dem gleichen Wert für DNAME (linke Seite der FA) gibt.
- Dann müssen diese Zeilen auch den gleichen Wert für TELEFON (rechte Seite der FA) haben.
- Davon sind alle bis auf eine Kopie redundant.

  Formal ist jeder dieser Einträge für sich genommen (einzeln) redundant. Man kann ihn nicht unabhängig ändern. Da Relationen Mengen sind, ist es aber schwierig, Redundanz so zu definieren, daß der erste Eintrag nicht redundant ist, aber alle weiteren schon (obwohl dies intuitiv ja durchaus richtig ist: Nachdem man den ersten Eintrag gelesen hat, geben die weiteren keine neue Information mehr).

Motivation (3)

- Dies gilt auch allgemein. Wenn man Redundanzen aufgrund von FAen vermeiden will, muß für jede FA

$$A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$$

gelten, daß es niemals zwei Zeilen geben kann, die in den Attributen $A_1, \ldots, A_n$ übereinstimmen.

Es sei denn, die FA wäre trivial: Wenn $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$ folgt nicht automatisch eine redundante Speicherung: Wenn man Werte in den B_j von einer der beiden Zeilen ändert, würde sich dann ja auch das A_i ändern (das in Wirklichkeit das gleiche Attribut ist), und die FA wäre nicht verletzt (man kann triviale Abhängigkeiten ja gar nicht verletzen).

- D.h. $A_1, \ldots, A_n$ müssen einen Schlüssel bilden.

Motivation (4)

- Eine Ursache für das Problem ist auch, daß Informationen über verschiedene Konzepte (Dozenten und Vorlesungen) zusammen gespeichert werden.
- Formal folgt dies auch aus "`DNAME` $\longrightarrow$ `TELEFON`":
  - ◇ `DNAME` ist eine Art Schlüssel für einen Teil der Attribute.
  - ◇ Er identifiziert Dozenten, und `TELEFON` hängt nur vom Dozenten, nicht von der Vorlesung, ab.
- Also: Die linke Seite der FA sollte ein Schlüssel sein.

Es ist kein Problem, wenn eine Relation zwei Schlüssel hat: Dann gibt es nur zwei Möglichkeiten, das gleiche Konzept zu identifizieren.

Boyce-Codd Normalform

- Eine Relation R ist in BCNF genau dann, wenn alle ihre FAen schon durch Schlüssel impliziert sind.

Damit braucht man bei einer Relation in BCNF keine FAen als Integritätsbedingungen, sondern nur Schlüsselbedingungen.

- D.h. für jede FA "$A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$" muß eine der folgenden Bedingungen gelten:
  - ◇ Die FA ist trivial, d.h. $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$.
  - ◇ $\{A_1, \ldots, A_n\}$ oder eine Teilmenge davon ist ein Schlüssel.

Da hier "Schlüssel" auch nicht minimal sein können, ist "oder eine Teilmenge davon" eigentlich überflüssig. Man beachte, daß es irgendein Schlüssel sein kann, nicht notwendig der Primärschlüssel.

Prüfung auf BCNF

- Man kann aus den gegebenen FAen $\mathcal{F}$ zuerst die Schlüssel der Relation bestimmen, und dann die Definition direkt anwenden:
  - ◇ Für jede FA $\alpha \longrightarrow \beta$ aus $\mathcal{F}$ muß gelten: Falls $\beta \not\subseteq \alpha$, dann enthält α einen der Schlüssel (plus eventuell weitere Attribute).
- Man kann aber auch für jede FA $\alpha \longrightarrow \beta$ mit $\beta \not\subseteq \alpha$ prüfen, ob die Attribut-Hülle α^+ schon die Menge aller Attribute der Relation ist.

Dann ist α oder eine Teilmenge davon gerade ein Schlüssel.

Beispiele (1)

- `VORLESUNGEN(VNR, TITEL, DNAME, TELEFON)` mit
  - ◇ `VNR` ⟶ `TITEL, DNAME, TELEFON`
  - ◇ `DNAME` ⟶ `TELEFON`

  ist nicht in BCNF, da die FA "`DNAME` ⟶ `TELEFON`" nicht durch einen Schlüssel impliziert wird:
  - ◇ "`DNAME`" ist kein Schlüssel der ganzen Relation.
  - ◇ Die FA ist nicht trivial.
- Aber ohne das Attribut `TELEFON` (und die FA), ist die Relation in BCNF:
  - ◇ `VNR` ⟶ `TITEL, DNAME` entspricht dem Schlüssel.

Beispiele (2)

- Angenommen, jede Lehrveranstaltung findet nur einmal die Woche statt. Dann erfüllt

  `RAUM_BELEGUNG(VNR, TITEL, TAG, ZEIT, RAUM)`

  die folgenden FAen (plus implizierte):
  - ◇ `VNR` ⟶ `TITEL, TAG, ZEIT, RAUM`
  - ◇ `TAG, ZEIT, RAUM` ⟶ `VNR`
- Die Schlüssel sind "`VNR`" und "`TAG, ZEIT, RAUM`".
- Beide FAen haben einen Schlüssel auf der linken Seite. Damit ist diese Relation in BCNF.

Beispiele (3)

- Man betrachte `PRODUKT(NR, NAME, PREIS)` mit:

  (1) `NR` ⟶ `NAME` (3) `PREIS, NAME` ⟶ `NAME`

  (2) `NR` ⟶ `PREIS` (4) `NR, PREIS` ⟶ `NAME`
- Diese Relation ist in BCNF:
  - ◇ FA (1) und (2) zeigen, daß `NR` Schlüssel ist. Also sind sie kein Problem: Schlüssel auf linker Seite.
  - ◇ FA (3) ist trivial (kann ignoriert werden).
  - ◇ FA (4) hat eine Obermenge des Schlüssels auf der linken Seite, was auch kein Problem ist.

    Sie wird von FA (1) logisch impliziert, ist also auch überflüssig.

Übungen

- Ist `BEWERTUNGEN(SID, ANR, PUNKTE, MAX_PUNKTE)` mit den folgenden FAen in BCNF?

  (1) `SID, ANR` ⟶ `PUNKTE`

  (2) `ANR` ⟶ `MAX_PUNKTE`

  Bestimmen Sie zuerst alle minimalen Schlüssel (es gibt nur einen).
- Ist die Relation

  `BESTELLT(AUFTRAGS_NR, DATUM, KUND_NR, PROD_NR, PROD_BEZEICHNUNG, MENGE)`

  mit den zuvor bestimmten FAen in BCNF?

Dritte Normalform (1)

- Eine Relation ist in dritter Normalform (3NF) genau dann, wenn für jede funktionale Abhängigkeit $A_1, \ldots, A_n \longrightarrow B$ (mindestens) eine der folgenden drei Bedingungen gilt:

  Zur Vereinfachung wird hier angenommen, daß alle FAen rechts nur ein Attribut haben. Wie oben erläutert, ist das keine Einschränkung.

  ◇ $B \in \{A_1, \ldots, A_n\}$ (die Abhängigkeit ist trivial),

  ◇ $\{A_1, \ldots, A_n\}$ (oder eine Teilmenge davon) ist ein Schlüssel,

  ◇ B ist in einem minimalen Schlüssel enthalten (d.h. B ist ein Schlüsselattribut).

Dritte Normalform (2)

- Gegenüber BCNF kommt der dritte Punkt hinzu.
- Daher gilt:
  Ist eine Relation in BCNF, so ist sie auch in 3NF (BCNF ist eine stärkere Forderung als 3NF).
- In der Praxis sind Fälle selten, die in 3NF, aber nicht in BCNF sind.

  "BCNF ist die 3.2-te Normalform." Die Bedeutung von 3NF liegt darin, daß sie sich immer "mit Erhaltung der FAen" herstellen läßt. BCNF läßt sich immer ohne Informationsverlust herstellen, aber u.U. mit Verlust von FAen. Ob die Erhaltung der FAen praktisch relevant ist, kann bezweifelt werden. Genaueres zum Vergleich in der Vorlesung DB IIA.

Inhalt

Relationen aufspalten (1)

- Eine Tabelle, die nicht in BCNF ist, kann in zwei Tabellen aufgespalten werden ("Dekomposition"), z.B. kann man `VORLESUNGEN` aufspalten in

  `VORL_NEU(VNR, TITEL, DNAME→DOZENTEN)`
  `DOZENTEN(DNAME, TELEFON)`

- So wird die von der FA erzeugte Redundanz vermieden: Die Telefonnummer eines Dozenten wird nur noch ein Mal abgespeichert.
- Es werden auch die verschiedenen Konzepte (Vorlesungen und Dozenten) klar getrennt.

Relationen aufspalten (2)

VORLESUNGEN			
VNR	TITEL	DNAME	TELEFON
22268	Datenbanken I	Brass	24740
42232	Grundlagen des WWW	Brass	24740
31822	Rechnerarchitektur	Molitor	24710

VORL_NEU		
VNR	TITEL	DNAME
22268	Datenbanken I	Brass
42232	Grundlagen des WWW	Brass
31822	Rechnerarchitektur	Molitor

DOZENTEN	
DNAME	TELEFON
Brass	24740
Molitor	24710

Relationen aufspalten (3)

- Allgemein: Verletzt $A_1...A_n \longrightarrow B_1...B_m$ die BCNF in der Relation R, so erstellt man eine neue Relation

$$S(\underline{A_1}, \ldots, \underline{A_n}, B_1, \ldots, B_m)$$

  und entfernt $B_1, \ldots, B_m$ aus der Relation R.

  $B_1, \ldots, B_m$ sollten alle Attribute sein, die durch $A_1, \ldots, A_n$ bestimmt werden (sonst erzeugt man unnötig viele Relationen). Natürlich darf kein B_i unter den A_j sein (dieser Teil der FA wäre dann ja auch trivial).

- $A_1, \ldots, A_n$ wird Fremdschlüssel in der Relation R, der auf die neue Relation S verweist.

Relationen aufspalten (4)

- In seltenen Fällen (bei mehrfachen Verletzungen der Normalform) muss man eine (oder beide) der resultierenden Tabellen erneut aufspalten.

  Dann müssen auch implizierte FAen betrachtet werden.

  Beispiel: $R(A, B, C, D)$ mit FAen $B \longrightarrow C$ und $C \longrightarrow D$. Schlüssel ist A, B, also verletzen beide FAen BCNF. Beseitigt man das Problem durch die erste FA zuerst, bekommt man $R_1(A, B, D)$ und $R_2(B, C)$. Die gegebenen FAen implizieren aber $B \longrightarrow D$, also ist R_1 noch nicht in BCNF und muß erneut aufgespalten werden in $R_{11}(A, B)$ und $R_{12}(B, D)$. Das Attribut B in R_{11} referenziert jetzt gleichzeitig R_{12} und R_2. Formal wäre das Ergebnis so in BCNF, schön ist es aber nicht. Man hätte ein natürlicheres Ergebnis bekommen, wenn man mit der anderen funktionalen Abhängigkeit begonnen hätte. Es ist ganz typisch, daß man im Prinzip aus den FAen Tabellen baut. Für 3NF wird ein entsprechender Algorithmus auf den nächsten Folien gezeigt.

Algorithmus für 3NF (1)

- Der folgende Algorithmus produziert eine korrekte Aufspaltung einer Relation in 3NF Relationen.

  Korrekt bedeutet "verlustlos" (s.u.), "Erhaltung von FAen" (s. DB II A).

- Zuerst bestimmt man eine minimale Menge von FAen, die äquivalent zu den gegebenen FAen sind ("kanonische Überdeckung"):
  - ◇ Ersetze jede FA $\alpha \longrightarrow B_1, \ldots, B_m$ durch die entsprechenden FAen $\alpha \longrightarrow B_i$ $(i = 1, \ldots, m)$. Initialisiere die Variable $\mathcal{F}$ mit dem Resultat.
  - ◇ . . . (Fortsetzung auf der nächsten Folie) . . .

Algorithmus für 3NF (2)

- Berechnung der kanonischen Überdeckung, Forts.:
  - ◇ Minimiere die linke Seite jeder FA.

    Für jede FA $A_1, \ldots, A_n \longrightarrow B$ und jedes $i = 1, \ldots, n$ berechne die Attribut-Hülle $\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n\}^+$ (bezüglich $\mathcal{F}$). Enthält das Ergebnis B, so ist die Menge $\mathcal{F}$ von FAen äquivalent zu $\mathcal{F}' := (\mathcal{F} - \{A_1, \ldots, A_n \longrightarrow B\}) \cup \{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \longrightarrow B\}$: Die stärkere FA (ohne das Attribut A_i) wird von den anderen FAen ohnehin impliziert. Rechne in diesem Fall weiter mit $\mathcal{F} := \mathcal{F}'$.
  - ◇ Entferne logisch implizierte FAen.

    Für jede FA $\alpha \longrightarrow B$ in $\mathcal{F}$, berechne die Attribut-Hülle α^+ bezüglich $\mathcal{F}' := \mathcal{F} - \{\alpha \longrightarrow B\}$. Falls α^+ das Attribut B enthält, entferne die FA $\alpha \longrightarrow B$, d.h. rechne mit $\mathcal{F} := \mathcal{F}'$ weiter ($\alpha \longrightarrow B$ ist von den anderen FAen logisch impliziert, wieder sind $\mathcal{F}$ und $\mathcal{F}'$ äquivalent).

Algorithmus für 3NF (3)

- Synthesealgorithmus für 3NF:
  - ◇ Berechne die kanonische Überdeckung $\mathcal{F}$ (s.o.).
  - ◇ Für jede linke Seite α einer FA in $\mathcal{F}$, erzeuge eine Relation mit Attributen $\mathcal{A} := \alpha \cup \{B \mid \alpha \longrightarrow B \in \mathcal{F}\}$.

    Zu dieser Relation gehören alle $\alpha' \longrightarrow B' \in \mathcal{F}$ mit $\alpha' \cup \{B'\} \subseteq \mathcal{A}$.
  - ◇ Enthält keine der konstruierten Relationen einen Schlüssel der Ausgangsrelation, füge eine Relation mit allen Attributen eines Schlüssels hinzu.
  - ◇ Enthält eine der konstruierten Relationen eine Teilmenge der Attribute einer anderen Relation, entferne die Relation mit der Teilmenge.

Verlustlosigkeit (1)

- Beim Aufspalten von Relationen ist es natürlich wichtig, daß die Transformation verlustlos ist, d.h. daß die Ausgangsrelation durch eine Anfrage wieder hergestellt werden kann (kein Informationsverlust):
  - ◇ Gegeben alte Relation "VORLESUNG" mit Inhalt.
  - ◇ Man füllt die neuen Relationen durch Anfragen mit den Daten aus der alten Relation:

    $$\texttt{VORL\_NEU} := \pi_{\texttt{VNR, TITEL, DNAME}}(\texttt{VORLESUNGEN})$$
    $$\texttt{DOZENTEN} := \pi_{\texttt{DNAME, TELEFON}}(\texttt{VORLESUNGEN})$$
  - ◇ Dann gilt:

    $$\texttt{VORLESUNGEN} = \texttt{VORL\_NEU} \bowtie \texttt{DOZENTEN}.$$

Verlustlosigkeit (2)

- Man kann nun die ursprüngliche Relation löschen und eine Sicht mit gleichem Namen definieren.
- Zum Beispiel würde das in SQL so aussehen:

```
CREATE VIEW VORLESUNGEN(VNR,TITEL,DNAME,TELEFON)
AS
SELECT V.VNR, V.TITEL, D.DNAME, D.TELEFON
FROM   VORL_NEU V, DOZENTEN D
WHERE  V.DNAME = D.DNAME
```

- Man kann nun die "virtuelle Tabelle" VORLESUNGEN in Anfragen wie die ursprüngliche Tabelle verwenden.

Verlustlosigkeit (3)

Definition:

- Die Aufspaltung einer Relation

  $$R(A_1, \ldots, A_k, B_1, \ldots, B_m, C_1, \ldots, C_n)$$

  in Relationen

  ◇ $R_1(A_1, \ldots, A_k, B_1, \ldots, B_m)$

  ◇ $R_2(A_1, \ldots, A_k, C_1, \ldots, C_n)$

  ist genau dann verlustlos, wenn

  $$R = \pi_{A_1,\ldots,A_k,B_1,\ldots,B_m}(R) \bowtie \pi_{A_1,\ldots,A_k,C_1,\ldots,C_n}(R)$$

  für alle gültigen Datenbank-Zustände gilt.

  Gültiger Zustand: erfüllt Integritätsbedingungen, insbesondere FAen.

Verlustlosigkeit (4)

- Nicht jede Aufspaltung ist verlustlos. Z.B. ergibt die Aufspaltung folgender Relation $R(A, B, C)$ in $R_1(A, B)$ und $R_2(B, C)$:

R:

A	B	C
1	3	4
2	3	5

→

R_1:

A	B
1	3
2	3

R_2:

B	C
3	4
3	5

→

$R_1 \bowtie R_2$:

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

- Der Vergleich über B ist nicht selektiv genug. Daher entstehen auch falsche Tupelkombinationen, wenn man versucht, die Tabelle R zu rekonstruieren.

Verlustlosigkeit (5)

- Dekompositionssatz: Das Aufspalten von Relationen ist garantiert verlustlos, falls der Schnitt der Attribute der neuen Tabellen ein Schlüssel von mindestens einer der beiden ist. Z.B.:

  $$\{\texttt{VNR}, \texttt{TITEL}, \texttt{DNAME}\} \cap \{\texttt{DNAME}, \texttt{TELEFON}\} = \{\texttt{DNAME}\}.$$

- Bei der obigen Methode zur Transformation von Relationen in BCNF finden nur Aufspaltungen statt, die dieser Bedingung genügen.
- Eine Relation kann immer verlustlos in BCNF aufgespalten werden (falls notwendig auch mehrmals).

Verlustlosigkeit (6)

- Nicht jede verlustlose Aufspaltung ist sinnvoll:

STUDENTEN		
<u>SID</u>	VORNAME	NACHNAME
101	Lisa	Weiss
102	Michael	Grau

- Das Aufspalten in `STUD_VORNAME(`<u>`SID`</u>`,VORNAME)` und `STUD_NACHNAME(`<u>`SID`</u>`,NACHNAME)` ist verlustlos, aber
  - ◇ nicht notwendig zur Herstellung einer NF und
  - ◇ erzwingt zusätzliche Verbunde (aufwändige Verknüpfung beider Tabellen) in späteren Anfragen.

Schema-Inäquivalenz (1)

- Alle vorher möglichen Zustände können in das neue Schema (nach der Aufspaltung) abgebildet werden.

  Man kann Zustände vom alten in das neue Schema übersetzen (falls die FA erfüllt war). Das neue Schema erlaubt alle Anfragen, die das alte Schema unterstützte: Die alten Relationen können als Sichten definiert werden. Durch die Verlustlosigkeit bekommt man die gleichen Antworten.

- Aber das neue Schema erlaubt Zustände, die im alten Schema nicht möglich waren: Jetzt können Dozenten ohne Vorlesungen gespeichert werden.
- Damit sind die beiden Schemas nicht äquivalent: Das neue ist allgemeiner.

Schema-Inäquivalenz (2)

- Falls Dozenten ohne Vorlesungen in dem zu modellierenden Weltausschnitt tatsächlich möglich sind, beseitigt die Dekomposition einen Fehler des alten Schemas (Einfüge- und Löschanomalie).
- Falls nicht,
  - ◇ wird eine neue Integritätsbedingung benötigt, die nicht unbedingt leichter zu sichern ist als eine funktionale Abhängigkeit.

    Beide können nicht deklarativ im `CREATE TABLE` spezifiziert werden.
  - ◇ Zumindest ist aber die Redundanz vermieden (Updateanomalie).