Kapitel 7 Normalformen und DB-Entwurf

Kap. 7.1 Normalformen – Theorie

Funktionale Abhängigkeit:

$f \subset X \cdot Y$ f als Relation, d.h. Menge von Paaren

{(x,y)}

x: Definitions-Stelle, y: Funktionswert

f ist Funktion $\Leftrightarrow (x_1, y_1), (x_2, y_2) \in f \ \wedge \ x_1 = x_2 \rightarrow y_1 = y_2$

d.h. zu Definitions-Stelle eindeutiger Funktionswert

$\forall x \in X \, \exists! \, y \in Y : (x,y) \in f$

umgekehrt: $(x_1,y), (x_2,y) \in f$ ist o.k.

$X \xrightarrow{f} Y$ $f(x) = y$

Def: Sei $R \subseteq (D_1 \times D_2 \times ... \times D_m)$ mit Attributen

a1 a2 ... am

dann ist a_j ***funktional abhängig*** von a_i

wenn $\exists$ Funktion g mit:

$R.a_i \xrightarrow{g} R.a_i$

meist geschrieben als

$a_i \longrightarrow a_j$

ohne explizite Angabe von g, g kann sich ändern

$\Rightarrow \forall r \in R : r.a_j = g\,(r.a_i)$

Genauer: Zu jedem Zeitpunkt der Lebensdauer einer DB existiert eine Funktion g mit:

$R.a_i \xrightarrow{g} R.a_j$

g änderbar zu g‘ durch updates der DB.

Konkret: seien $r_1, r_2, ..., r_k \in R$ mit

$r_1.a_i = r_2.a_i = ... = r_k.a_i = d_i$

$\Rightarrow$ $r_1.a_j = r_2.a_j = ... = r_k.a_j = d_j$

und $d_j = g(d_i)$

Folge: Update Anomalie:

Wertepaare (d_i, d_j) an vielen Stellen in R nicht unabhängig änderbar:

1) für 1 Tupel r_k: $(r_k.a_i, r_k.a_j) := (d_k, g(d_k))$

ohne Änderung von g wird $g(d_k)$ automatisch berechnet oder gar nicht gespeichert

2) oder Änderung von a_j an vielen Stellen, i.e. aus g wird g‘

update R
set $a_i = d_k$‘
where $a_i = d_k$

i.e. ändere alle Tupel mit $a_i = d_k$

bzw. g zu g‘ mit $g'(d_k') = g(d_k)$

z.B.

1	2	a
2	3	b
3	2	a
4	5	c

bei funktionaler Abhängigkeit $a_2 \rightarrow a_3$

Anmerkung:
Funktionale Abhängigkeit von Primärschlüssel ist immer erfüllt

Konsequenz:
mehrfache Speicherung von (d_i, d_j) in $r_1, ..., r_k$ nicht nötig,
Attribut a_i verbleibt in R,

$a_i \xrightarrow{g} a_j$ abspalten u.

in weiterer Relation getrennt speichern

$R(k, a_i , a_j)$ wird zerlegt in 2 Relationen

$R1 (k, a_i)$ und $R2 (a_i , a_j)$ mit

$R(k, a_i , a_j) = R1 (k, a_i) \ \iota \ R2 (a_i , a_j)$

Beispiel:

LIEF : L# → LNAME, STATUS, STADT

TEILE : T# → TNAME, FARBE, GEWICHT

LT : L#, T# → ANZAHL

Werteänderung: z.B.

L3 Blake 10 Paris

nur 1 Tupel, da L# Schlüssel und L# sich nicht ändert

Schema Änderung:

TNAME ⟶ FARBE

STADT ⟶ STATUS

Funkt. Abhängigkeiten sind grundsätzliche Entwurfs- u. Modellierungsentscheidungen, nicht Zufälligkeit des DB-Zustandes.

1. Normalform:

Def: Relation R ist in ***1. Normalform*** **(1NF)**, wenn alle Domänen einfache Wertemengen sind.

Def: ***Volle funktionale Abhängigkeit****:* Attribut d ist in R voll funktional abhängig von $(a_1, a_2, ..., a_k)$, wenn d nicht schon von echter Untermenge von $(a_1, ..., a_k)$ funktional abhängig ist.

Hinw: Jedes Attribut ist von Primärschlüssel funktional abhängig, nicht immer voll funktinal abhängig.

Hinw: Falls Primärschl. nicht zusammengesetzt ist und Attr. ***a*** nicht konstant ist, ist ***a*** voll funktional abhängig von Primärschlüssel.

Bedeutung: Sei R (a, b, c, d)

key is (a, b)

und c funkt. abhängig von b, d.h.

b → c

d.h. Attr. Wert r.c für r ∈ R ist der Gruppe von Entities mit Attr. – Wert r.b gemeinsam, unabhängig von r.a d.h. (r.b., r.c) ist Aussage über Gruppe von Entities r.b.

TNAME ⟶ FARBE in der Relation

TEILE :	**T#**	**TNAME**	**FARBE**	**GEWICHT**
	327	Schraube	silbern	12
	347	Schraube	silbern	9
	2431	Reifen	schwarz	3619
	2432	Reifen	schwarz	3730

⇒ Gruppen-Aussagen herausfaktorisieren: z.B.

alle Schrauben sind silbern

alle Reifen sind schwarz

R (a, b, c, d) mit Abhängigkeiten

a,b → c, d zerlegen in

b → c

a,b → d

weitere Abhängigkeit a → d

führt zu a,b

a → d

b → c

$\Rightarrow$ R zerlegt in 3 Relationen

R1 (**a**,**b**) R2 (**b**, c) R3 (**a**, d)

R2, R3 sind schwache Entitäten,
sie sind Aussagen über Gruppen und verschwinden mit letztem Gruppenmitglied in R1.

Def: D ist *transitiv funktional abhängig* von X (über C),
wenn

1. $X \rightarrow C \rightarrow D$
2. X, C, D sind verschiedene Attribute
3. $X \not\leftarrow C$

d.h. es gibt Entity-Mengen $C \rightarrow D$ und d ist Eigenschaft von c

z.B. L# $\rightarrow$ Stadt $\rightarrow$ STATUS
$\not\leftarrow$ $\not\leftarrow$

Strategien: Relation R (X, C, D) **key** X

mit $X \rightarrow C \rightarrow D$

$\not\leftarrow \quad \not\leftarrow$

zerlegen (faktorisieren) in:

table R‘ (X, C) **key** X
table R“ (C, D) **key** C

dann gilt: $R = R' \iota_{2,1} R''$

Def: Sei X Primärschlüssel, dann heißt Y ***Schlüsselkandidat***,
wenn $X \leftrightarrow Y$
d.h. $\exists$ bijektive Abb. zwischen X und Y
alternativ: Y Primärschl., X Schlüsselkandidat;
in Vereinbarung (Schema Definition) festlegen.

Anm: Theoretisch sind alle Schlüssel gleich gut zur Tupelidentifizierung, z.B.
(Name, Tel#) oder
(Name, Adresse)

Anm: Zerlegung nur sinnvoll bei transitiven Abhängigkeiten
der Form $X \rightarrow Y \rightarrow Z$

$\not\leftarrow$

sonst mit $X \leftrightarrow Y \rightarrow Z$ in R (X, Y, Z)
$|R'(X,Y)| = |R''(Y, Z)|$

d.h. Zerlegung lohnt nicht, mehr Speicher u. zusätzliche Joins.

Erklärung: X, Y identifizieren diesselbe Entity,
$Y \rightarrow Z$ beschreibt Eigenschaft, aber keine neue Entity.

Def: R ist *in zweiter Normalform (2NF)* nach **Codd**,
wenn:

1. R ist in 1NF
2. die Nicht-Schlüssel-Attribute sind voll funktional abhängig vom Primärschlüssel

(d.h. keine Gruppeneigenschaften aus Primärschlüssel ersichtlich und deshalb auch nicht herausfaktorisierbar)

Def.: R ist in *2NF* nach **Kent**,
wenn:

1. R ist in 1NF
2. ∀ Schlüsselkandidaten gilt:
   Attribute im Kompliment des Schlüsselkandidaten sind voll funktional abhängig von Schlüsselkandidat.

(d.h. Kent entspricht Codd. Def. erweitert auf alle Schlüsselkandidaten)

Anm: 2NF Kent heißt: es gibt keine Gruppeneigenschaften, die nicht aus Primärschlüssel ersichtlich wären.

Beispiel: ***Welche Normalformen sind verletzt?***

```
create table Kunden
        (K#             integer
        Kname   =       (Vname      string,
                         Fname      string),
        Adresse =        ( PLZ      char (7),
                         Stadt      string,
                         Straße     string,
                         Haus#      string ),
        Tel     =        ( Vorwahl  integer,
                         Tel#       integer ),
        Fax     =        ( Vorwahl  integer
                         Fax#       integer )
        ) key is K#
```


Beispiele für Abhängigkeiten:

PLZ	$\rightarrow$	Stadt
Stadt, Straße	$\rightarrow$	PLZ
Vorwahl, Tel#	$\rightarrow$	?

Def: R ist in *dritter Normalform* 3NF, wenn

1. R ist in 2NF
2. R enthält keine transitiven funktionalen Abhängigkeiten.

Hinweis: 3NF entsteht bei guter E/R Modellierung meistens automatisch ! Für zusätzliche automatische Nachprüfung siehe spätere Kapitel.

Manipulationsanomalien:

bei R in 1NF:

1. Gruppeneigenschaft verschwindet mit letztem Gruppenmitglied, z.B. (PLZ, Stadt)
2. Änderung einer Gruppeneigenschaft an mehreren Stellen, sonst Inkonsistenz, z.B. (PLZ, Stadt), Post hat vor einigen Jahren die PLZ geändert

bei R in 2NF:

1. Eigentlich unabhängige Entity kann nur als Bestandteil einer anderen existieren, z.B. (Stadt, Vorwahl) verschwindet mit letzer PLZ
2. Änderung an mehreren Stellen
   z.B. Änderung von Vorwahl durch Telekom

Beispiel: LTS in 1NF.
nicht in 2NF:

LTS (L#, T#, A, Stadt) **key is** L#, T#
mit funktionalen Abhängigkeiten:

L#, T# $\rightarrow$ A
L# $\rightarrow$ Stadt

d.h. ‚London' ist gemeinsame Eigenschaft der Gruppe der Lieferungen des Lieferanten L4

1. Tatsache, daß L4 in London sitzt, verschwindet mit letzter Lieferung
2. An mehreren Stellen (Tupeln) zu ändern, falls L4 umzieht, weil Informationen redundant gespeichert sind.

Anmerkung:

Normalformen-Theorie bezieht sich nur auf permanente, modell-inhärente funktionale Abhängigkeiten, nicht auf zufällige Wertekonstellationen.

⇒ *Funktionale Abhängigkeit ist **nachprüfbar**, per Programm, aber **nicht entdeckbar**. Muß mit Schema-Definition als Invariante der Datenbank spezifiziert werden !!*

Bedeutung der Normalformen:

1. Funktionale Abhängigkeiten festlegen,
   als Teil eines Datenmodells, Unternehmensmodells, z.B. in Bank

2. Mit funktionalen Abhängigkeiten relationales Schema fixieren:
   2 a. Rel in 3NF definieren:
   ⇒ keine zu überwachenden Abhängikeiten
   mehr, nur noch referentielle Integrität.
   2 b. Rel in 1NF oder 2NF definieren:
   ⇒ funktinale Abhängigkeiten durch DBS überwachen mit
   automatischer Beachtung von Update-Anomalien.

Offene Frage: *Effizienz u. Programmier-Bequemlichkeit???*

Idee: *3NF entspricht am besten der natürlichen Semantik.*

Beispiel: Schema in 3NF

table Kunden3 (K#
Kname
Plz
Straße
Haus#
Tel#
Fax#)

table Städte (PLZ,
Stadt)

table Vorwahlen (Stadt,
Vorwahl)

Programmier Beispiele:

„Finde Telefon-Nr. von Kunde 17“

```
select      v.Vorwahl,  k.Tel#
   from Kunden3  k,  Städte s,   Vorwahlen v
        where k. K#          = 17        and
              k. PLZ         = s. PLZ    and
              s. Stadt       = v. Stadt
```

mit 1NF: Schema : Kunden

```
select  Vorwahl, Tel#
   from Kunden
        where K# = 17
```

mit 3NF und View Kunden:

```
create view    KundenView ( . . .)
    as
    select  . . .
        from Kunden3  k, Städte  s, Vorwahlen  v
            where       k.PLZ        = s. PLZ and
                        s. Stadt     = v. Stadt
```

Query :

```
    select Vorwahl, Tel#
        from KundenView
            where K# = 17
```

Fazit: *Programmierung einfach, Effizienz schlecht!*