Freie Universität Berlin

Prof. Dr. Agn‘es Voisard, Nicolas Lehmann

Datenbanksysteme, SoSe 2017

Übungsblatt 6

Tutor: Nicolas Lehmann
Tutorium 10

Boyan Hristov, Julian Habib

17. Juni 2017

Link zum Git Repository: https://github.com/BoyanH/Freie-Universitaet-Berlin/tree/master/Datenbanksysteme/Solutions/homework6

1. Aufgabe

a) z.z.: R_1 ist in 1NF $\Leftrightarrow$ alle Attribute sind atomar.

$R_1(A, B, C, D) \subseteq A \times B \times C \times D \Rightarrow$ A,B,C,D sind atomar.

Alle Attributen in R_1 haben atomäre Domäne, sind also keine Relationen. Z.B. alle Einträge haben für den Attribut A die Werte a_1, a_2, a_3 und keine Werte wie z.B. (a_1, a_2).

□

b) z.z.: R_1 ist nicht in 2NF

R_1 ist in 2NF $\Leftrightarrow$ R_1 ist in 1NF und $\forall(X \rightarrow A) \in F_+ | A \notin X : X \not\subset$ Superschlüssel $\vee A \subseteq$ Schlüssel.

Beweis durch Gegenbeispiel:

$$
\begin{aligned}
FD(R_1) = \{ & \\
& A \rightarrow B \\
& A \rightarrow C \\
& D \rightarrow C \\
& BD \rightarrow A \\
\} & \\
\Rightarrow & \text{ AD und BD sind Superschlüssel und auch beide Kandidatschlüsseln.} \\
& \text{Primäre Attributen sind A, B und D.}
\end{aligned}
$$

Da C kein Primärattribut ist, von A abhängig ist und da A eine Untermenge eines Schlüssels ist (AD), ist R_1 nicht in 2NF.

□

c) z.z.: R_2 ist in 3NF $\Leftrightarrow \forall FD(X \to A)$: X ist Superschlüssel von R oder A ist Primärattribut von R

Direkter Beweis:

$$
\begin{aligned}
FD(R_2) = \{ & \\
& E \to F \\
& E \to G \\
& FG \to E \\
\} & \\
\Rightarrow & \text{ E und FG sind Superschlüssel, Primärattributen sind E, F und G.}
\end{aligned}
$$

In den ersten zwei funktionalen Abhängigkeiten ist die linke Seite ein Superschlüssel, in der 3. Abhängigkeit ist die rechte Seite ein Primärattribut. $\Rightarrow R_2$ ist in 3NF.

□

d) z.z: R_2 ist in BCNF $\Leftrightarrow \forall X \to A \in F_+$: X ist Superschlüssel.

Direkter Beweis:

$$
\begin{aligned}
FD^+(R_2) = \{ & \\
& E \to F \\
& E \to G \\
& E \to FG \\
& FG \to E \\
\} &
\end{aligned}
$$

Da wir schon aus c) kennen, dass E und FG Superschlüssel sind und da diese alle mögliche linke Seiten von einer funktionalen Abhängigkeit sind, ist R_2 in Boyce-Codd Normalform (BCNF).

□

2. Aufgabe

a) Mit dem Algorithmus aus der Vorlesung für "Well-behaved 3NF Decomposition"

$$
\begin{aligned}
FD(R_3) = \{ & \\
& H \to JK \\
& I \to HJ \\
& K \to L \\
\} &
\end{aligned}
$$

1. For each FD (X → A) in F create a relation with schema (XA).

$$(HJK), (IHJ), (KL)$$

2. If none of the keys appears in one of the schemas of 1 then add a relation with schema Y, with Y a key.

   $\Rightarrow$ fertig, da H ein Schlüssel ist und in (HJK) vorhanden ist.

3. If for relations created in 1. there exists a relation R1 whose schema is included in the schema of another relation, then remove R1.

   Alles ist in Ordnung, die Attributen von keinem Schema sind eine Untermenge von den Attributen eines anderen Schemas.

4. Replace relations (X A1), ..., (X Ak) with a single relation (X A1 ... Ak).

$$(HJK) \wedge (IHJ) \Rightarrow (HJIK)$$

Gute Zerlegung: $R_{31}(H, J, I, K)$ und $R_{32}(K, L)$

b)

$R_{31} \cap R_{32} \equiv K$
$R_{31} - R_{32} \equiv HJI$ $\qquad\qquad$ $R_{32} - R_{31} \equiv L$

$R_{31} \cap R_{32} \rightarrow KL$ (wegen K $\rightarrow$ L) $\rightarrow L \equiv R_{31} - R_{32}$
$\Rightarrow$ Unsere Zerlegung ist verlustlos

c)

$FD(R_{31}) = \{H \rightarrow JK, I \rightarrow HJ\}$
$FD(R_{32}) = \{K \rightarrow L\}$
$FD(R_{31}) \cup FD(R_{32}) \equiv \{H \rightarrow JK, I \rightarrow HJ, K \rightarrow L\} \equiv FD(R_3)$
$\Rightarrow$ Unsere Zerlegung ist abhängigkeitserhaltend

d) Da wir dieses Algorithmus benutzt haben, ist jede davon entstandene Relation in 3NF. Wegen b), c) und d) ist die Zerlegung gut. Für R_{32} sind H und I Schlüssel und an alle funktionalen Abhängigkeiten auf der linken Seite vorhanden, deswegen ist R_{32} in 3NF. R_{32} hat nur eine funktionale Abhängigket und nur 2 Attributen, ist deswegen trivialerweise in 3NF.

3. Aufgabe

a) Aus jedem Item kann man durch eine Hash-Funktion ein Hash erzeugen durch mathematische Umformungen aus allen Suchschlüssel. Mehrere Items werden in dem selben Bucket gespeichert (nacheinandere folgende Blöcke im Speicher) wenn die Hashfunktion das gleiche Hash aus ihren Suchschlüssel erzeugt. Danach, beim Suchen von Lösch / Einfüge / Lese Position müssen alle Einträge in dem entsprechenden Bucket nach einander geprüft werden. Deswegen erzeugen gute Hashfunktione randomisierte Hashes, damit es ungefähr genau so viele Items pro Bucket gibt.