Ludwig Maximilians Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Vorlesung

Datenbanksysteme I

Wintersemester 2014/2015

Kapitel 7: Normalformen

Vorlesung: PD Dr. Arthur Zimek
Übungen: Sebastian Goebl, Jan Stutzki

ergänzt von Matthias Schubert 2005, Arthur Zimek 2014

http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme_I

Relationaler Datenbank-Entwurf

- Schrittweises Vorgehen:
  - Informelle Beschreibung: Pflichtenheft
  - Konzeptioneller Entwurf: E/R-Diagramm
  - Relationaler DB-Entwurf: Relationenschema
- In diesem Kapitel:

  Normalisierungstheorie als formale Grundlage für den relationalen DB-Entwurf
- Zentrale Fragestellungen:
  - Wie können Objekte und deren Beziehungen ins relationale Modell überführt werden
  - Bewertungsgrundlagen zur Unterscheidung zwischen „guten“ und „schlechten“ relationalen DB-Schemata

Motivation Normalisierung

Pflichtenheft (Beispiel):

Die Datenbank speichert Lieferanten aus verschiedenen Städten und Ländern. Jeder Lieferant hat Waren zu bestimmten Preisen im Angebot.

E/R-Diagramm (Beispiel):

Motivation Normalisierung

- **Relation Lieferant:**

LNr	LName	LStadt	LLand	Ware	Preis
103	Huber	Berlin	Deutschland	Schraube	50
103	Huber	Berlin	Deutschland	Draht	20
103	Huber	Berlin	Deutschland	Nagel	40
…	…	…	…	…	…
762	Maier	Zürich	Schweiz	Nagel	45
762	Maier	Zürich	Schweiz	Schraube	55

- **Hier gibt es offensichtlich einige Redundanzen:**
  - Wenn bei mehreren Tupeln der Attributwert LNr gleich ist, dann müssen auch die Attributwerte von LName, LStadt und LLand gleich sein.
  - Wenn (auch bei verschiedenen LNr) LStadt gleich ist, dann muss auch LLand gleich sein
- **Redundanzen durch funktionale Abhängigkeiten**
  - Wir sagen: Die Attribute LName, LStadt und LLand sind **funktional abhängig** vom Attribut LNr (ebenso LLand von LStadt);
    Exakte Definition siehe Seite 9*ff*.

Motivation Normalisierung

- **Relation Lieferant:**

LNr	LName	LStadt	LLand	Ware	Preis
103	Huber	Berlin	Deutschland	Schraube	50
103	Huber	Berlin	Deutschland	Draht	20
103	Huber	Berlin	Deutschland	Nagel	40
…	…	…	…	…	…
762	Maier	Zürich	Schweiz	Nagel	45
762	Maier	Zürich	Schweiz	Schraube	55

- Redundanzen führen zu Speicherplatzverschwendung;
- Das eigentliche Problem sind aber Anomalien (Inkonsistenzen durch Änderungsoperationen) und dass das Schema nicht intuitiv ist:
  - **Update-Anomalie:** Änderung der Adresse (LName, LStadt, LLand) in nur *einem* Tupel statt in *allen* Tupeln zu einer LNr.
  - **Insert-Anomalie:** Einfügen eines Tupels mit inkonsistenter Adresse; Einfügen eines Lieferanten erfordert Ware.
  - **Delete-Anomalie:** Löschen der letzten Ware löscht die Adresse.

Verbesserung

- Neues Datenbankschema:

LieferantAdr	(LNr, LName, LStadt)
Stadt	(LStadt, LLand)
Angebot	(LNr, Ware, Preis)

Anmerkung: **Angebot** wäre nach Kap. 5, Seite 31 gleich als eigene Relation angelegt worden.

- Vorteile:
  - keine Redundanz
  - keine Anomalien
- Nachteil:
  - Um zu einer Ware die Länder der Lieferanten zu finden, ist ein zweifacher Join nötig (teuer auszuwerten und umständlich zu formulieren)

Ursprüngliche Relation

- Die ursprüngliche Relation Lieferant kann mit Hilfe einer View simuliert werden:

```
create view Lieferant as
    select    L.LNr, LName, L.LStadt, LLand, Ware, Preis
    from      LieferantAdr L, Stadt S, Angebot A
    where     L.LNr = A.LNr  and  L.LStadt = S.LStadt
```

- Dies spart die Schreibarbeit beim Formulieren von Anfragen und macht die Anfragen übersichtlicher.
- In diesem Fall ist die View *nicht updatable* (die Änderungsoperationen verursachen ja die Probleme).
- Da die View nur eine *virtuelle Relation* ist, muss der Join trotzdem ausgewertet werden (weniger effizient).

Schema-Zerlegung

- Anomalien entstehen durch Redundanzen
- Entwurfsziele:
  - Vermeidung von Redundanzen
  - Vermeidung von Anomalien
  - evtl. Einbeziehung von Effizienzüberlegungen
- Vorgehen:
  Schrittweises Zerlegen des gegebenen Schemas (Normalisierung) in ein äquivalentes Schema ohne Redundanz und Anomalien
- Formalisierung von Redundanz und Anomalien:
  Funktionale Abhängigkeit

Funktionale Abhängigkeit

(engl. Functional Dependency, FD)

- beschreibt Beziehungen zwischen den Attributen einer Relation
- Schränkt das Auftreten gleicher bzw. ungleicher Attributwerte innerhalb einer Relation ein
  → spezielle Integritätsbedingung (nicht in SQL)

Wiederholung Integritätsbedingungen in SQL:

- Primärschlüssel
- Fremdschlüssel (referenzielle Integrität)
- **not null**
- **check**

Wiederholung *Schlüssel*

Definition:

- Eine Teilmenge *S* der Attribute eines Relationenschemas *R* heißt ***Schlüssel***, wenn gilt:
  - ***Eindeutigkeit***
    Keine Ausprägung von *R* kann zwei verschiedene Tupel enthalten, die sich in ***allen*** Attributen von *S* gleichen.
  - ***Minimalität***
    Keine echte Teilmenge von *S* erfüllt bereits die Bedingung der Eindeutigkeit

Konventionen zur Notation

- Ab jetzt gilt die Notation:
  - A, B, C bezeichnen einzelne Attribute
  - X, Y, Z bezeichnen Mengen von Attributen

- Zur Vereinfachung gilt außerdem:
  - $A, B \rightarrow C$ bezeichne $\{A, B\} \rightarrow \{C\}$
  - $X \rightarrow Y, Z$ bezeichne $X \rightarrow Y \cup Z$
  - $t.A$ bezeichne das Attribut A des Tupels t
  - $t.X$ bezeichne die Menge X von Attributen des Tupels t
  - $t.X = r.X$ bezeichne $\forall A \in X : t.A = r.A$

Definition: *funktional abhängig*

- **Gegeben:**
  - Ein Relationenschema R
  - X, Y: Zwei *Mengen* von Attributen von R ($X, Y \subseteq R$)
- **Definition:**

  Y ist von X funktional abhängig ($X \rightarrow Y$)
  gdw. $\forall$Tupel t und r : $t.X = r.X \Rightarrow t.Y = r.Y$

  (für alle möglichen Ausprägungen von R gilt:
  Zu jedem Wert in X existiert genau ein Wert von Y.)
- Bsp.: **Lieferant** (<u>LNr</u>, LName, LStadt, LLand, <u>Ware</u>, Preis):
  - LNr → LName
  - LNr → LStadt
  - LStadt → LLand
  - LNr, Ware → LName
  - LNr, Ware → LStadt
  - LNr, Ware → Preis

Vergleich mit *Schlüssel*

- Gemeinsamkeiten zwischen dem *Schlüssel* im relationalen Modell und *Funktionaler Abhängigkeit*:
  - Für jeden Schlüsselkandidaten $S = \{A,B,...\}$ gilt: Alle Attribute der Relation sind von S funktional abhängig (wegen der Eindeutigkeit):

    $$S \rightarrow R$$

  - Jede Menge, von der R FD ist, ist Superschlüssel.
- Unterschied:
  - Aber es gibt u.U. weitere funktionale Abhängigkeiten: Ein Attribut B kann z.B. auch funktional abhängig sein
    - von Nicht-Schlüssel-Attributen
    - von nur einem Teil eines Schlüssels

➔ **FD ist Verallgemeinerung des Schlüssel-Konzepts.**

Vergleich mit *Schlüssel*

- Wie der Schlüssel ist auch die funktionale Abhängigkeit eine **semantische Eigenschaft** des Schemas:
  - FD nicht aus aktueller DB-Ausprägung entscheidbar
  - sondern muss für alle möglichen Ausprägungen gelten

Prime Attribute

- Definition:
  Ein Attribut heißt **prim**,
  wenn es Teil eines Schlüsselkandidaten ist

Partielle und *volle* FD

- Ist ein Attribut *B* funktional von *A* abhängig,
  dann auch von jeder Obermenge von *A*.
  Man ist interessiert, minimale Mengen zu finden, von denen *B* abhängt (vgl. Schlüsseldefinition)

- **Definition:**
  - Gegeben: Eine funktionale Abhängigkeit $X \rightarrow Y$
  - Wenn es keine echte Teilmenge $X' \subset X$ gibt,
    von der *Y* ebenfalls funktional abhängt,
  - dann heißt $X \rightarrow Y$ eine
    volle funktionale Abhängigkeit ($X \dot{\rightarrow} Y$)
  - andernfalls eine **partielle funktionale Abhängigkeit**

Partielle und *volle* FD

- Beispiele:
  - LNr $\rightarrow$ LName
  - LNr, Ware $\rightarrow$ LName
  - Ware ? Preis
  - LNr ? Preis
  - LNr, Ware $\rightarrow$ Preis

Volle FD vs. Minimalität des Schlüssels

- Definitionen sehr ähnlich:
  - Schlüssel: Minimale Menge, die Eindeutigkeit erfüllt
  - Volle FD: Minimale Menge, von der eine Attribut-Menge abhängt

- Folgt aus der Minimalität des Schlüssels, dass alle Attribute (immer) voll funktional abhängig sind?

- Der Bildbereich ist bei beiden Definitionen verschieden:
  - Schlüssel S: Minimale Menge, von der alle Attribute der Relation R funktional abhängig sind; $S \subseteq R$
  - Volle FD $X \dot{\rightarrow} Y$: X ist die minimale Menge, von der alle Attribute von Y funktional abhängig sind;
    $X \subseteq R, Y \subseteq R$; über Teilmengenbeziehungen zwischen X und Y ist nichts gesagt

Herleitung funktionaler Abhängigkeit

Armstrong Axiome

- Reflexivität (R): Falls Y eine Teilmenge von X ist ($Y \subseteq X$), dann gilt immer $X \rightarrow Y$. Insbesondere gilt also immer $X \rightarrow X$.
- Verstärkung (VS): Falls $X \rightarrow Y$ gilt, dann gilt auch $XZ \rightarrow YZ$. Hierbei steht XZ für $X \cup Z$.
- Transitivität (T): Falls $X \rightarrow Y$ und $Y \rightarrow Z$ gilt, dann gilt auch $X \rightarrow Z$.

Diese Axiome sind **vollständig** und **korrekt** :

Sei F eine Menge von FDs:

- Es lassen sich nur FDs von F ableiten, die von jeder Relationen-Ausprägung erfüllt werden, für die auch F erfüllt ist.
- Es sind alle FDs ableitbar, die durch F impliziert sind.

Herleitung funktionaler Abhängigkeit

Triviale funktionale Abhängigkeit:

- Wegen Reflexivität ist jedes Attribut funktional abhängig:
  - von sich selbst
  - von jeder Obermenge von sich selbst

Solche Abhängigkeiten bezeichnet man als **trivial.**

Symmetrieeigenschaften funktionaler Abhängigkeiten

- Bei den funktionalen Abhängigkeiten gibt es kein Gesetz zur Symmetrie oder Antisymmetrie, d.h. bei zwei beliebigen Attribut (-Mengen) X, Y sind alle 4 Fälle möglich:
  - Es gilt nur $X \rightarrow Y$,
  - Es gilt nur $Y \rightarrow X$,
  - Es gilt $X \rightarrow Y$ und $Y \rightarrow X$,
  - Es gibt keine FD. zw. X und Y.

Hülle einer Attributmenge

- Eingabe: eine Menge F von FDs und eine Menge von Attributen X.
- Ausgabe: die vollständige Menge von Attributen X^+, für die gilt $X \rightarrow X^+$.

```
AttrHülle(F,X)
 Erg := X
 while( Änderungen an Erg) do{
   foreach FD Y→Z ∈ F do{
       if Y ⊆ Erg then Erg:=Erg ∪ Z
   }
 }
 Ausgabe X+ = Erg
```

Hülle einer Attributmenge

- **Beispiel:** AttrHülle(F, {LNr}) mit

  F = {LNr → LName; LNr → LStadt; LStadt → LLand;
  LNr, Ware → Preis}

- Erg_i = Erg nach *i*-tem Durchlauf der **while**-Schleife

- Erg_0 = { LNr}
- Erg_1 = { LNr, LName, LStadt }
- Erg_2 = { LNr, LName, LStadt, LLand }
- Erg_3 = { LNr, LName, LStadt, LLand } = Erg_2

Herleitung funktionaler Abhängigkeit

Weitere Axiome

Vereinfachen Herleitungsprozess, aber nicht notwendig (da Armstrong-Axiome vollständig)

- Vereinigungsregel (VE):
  Falls $X \rightarrow Y$ und $X \rightarrow Z$ gilt, dann gilt auch $X \rightarrow YZ$.
- Dekompositionsregel (D):
  Falls $X \rightarrow YZ$ gilt, dann gilt auch $X \rightarrow Y$ und $X \rightarrow Z$.
- Pseudotransitivitätsregel (P):
  Falls $X \rightarrow Y$ und $ZY \rightarrow V$ gilt, dann gilt auch $XZ \rightarrow V$.

Herleitung funktionaler Abhängigkeit

Beispiel: Zeige, dass (LNr, Ware) Schlüsselkandidat ist.

- Gegeben: F = { LNr $\rightarrow$ LName; LNr $\rightarrow$ LStadt; LStadt $\rightarrow$ LLand; LNr, Ware $\rightarrow$ Preis }.
- Zu zeigen: (LNr, Ware) eindeutig und minimal.
- Beweis:

(I) (LNr, Ware) eindeutig gdw.
(LNr, Ware $\rightarrow$ LNr, LName, LStadt, LLand, Ware, Preis) $\in F^+$,
d.h. ist aus F herleitbar.

(a) (LNr $\rightarrow$ LName) $\in F \subseteq F^+ \overset{VS}{\Rightarrow}$ (LNr, Ware $\rightarrow$ LName, Ware) $\in F^+$

(b) (LNr $\rightarrow$ LStadt) $\in F^+ \overset{VS}{\Rightarrow}$ (LNr, Ware $\rightarrow$ LStadt, Ware) $\in F^+$

(c) (LNr $\rightarrow$ LStadt) $\in F^+$ und (LStadt $\rightarrow$ LLand) $\in F^+$
$\overset{T}{\Rightarrow}$ (LNr $\rightarrow$ LLand) $\in F^+ \overset{VS}{\Rightarrow}$ (LNr, Ware $\rightarrow$ LLand, Ware) $\in F^+$

(d) Wegen R gilt (LNr $\rightarrow$ LNr) $\in F^+ \overset{VS}{\Rightarrow}$ (LNr, Ware $\rightarrow$ LNr, Ware) $\in F^+$

(e) aus (a) bis (d) und (LNr, Ware $\rightarrow$ Preis) folgt (I) nach VE, qed.

Herleitung funktionaler Abhängigkeit

Beispiel: Zeige, dass (LNr, Ware) Schlüsselkandidat ist.

- Gegeben: F = { LNr → LName; LNr → LStadt; LStadt → LLand;
  LNr, Ware → Preis }.
- Zu zeigen: (LNr, Ware) eindeutig und minimal.
- Beweis:

  (II) (LNr, Ware) minimal gdw.
  (LNr → LNr, LName, LStadt, LLand, Ware, Preis) und
  (Ware → LNr, LName, LStadt, LLand, Ware, Preis) gelten nicht.

  (a) Preis ist nur von LNr und Ware gemeinsam funktional abhängig.

  (b) Weder LNr kann aus Ware hergeleitet werden, noch Ware von LNr.

  (c) aus (a) und (b) folgt (LNr, Ware) minimal, qed.

Normalisierung

- In einem Relationenschema sollen also möglichst keine funktionalen Abhängigkeiten bestehen, außer vom gesamten Schlüssel
- Verschiedene Normalformen beseitigen unterschiedliche Arten von funktionalen Abhängigkeiten bzw. Redundanzen/Anomalien
  - 1. Normalform
  - 2. Normalform
  - 3. Normalform
  - Boyce-Codd-Normalform
  - 4. Normalform

impliziert
impliziert
impliziert
impliziert

- Herstellung einer Normalform durch verlustlose Zerlegung des Relationenschemas

1. Normalform

- Keine Einschränkung bezüglich der FDs
- Ein Relationenschema ist in erster Normalform, wenn alle Attributwerte ***atomar*** sind
- In relationalen Datenbanken sind nicht-atomare Attribute ohnehin nicht möglich
- Nicht-atomare Attribute z.B. durch **group by**

A	B	C	D
1	2	3 4	4 5
2	3	3	4
3	3	4 6	5 7

„nested relation“
non first normal form
In SQL nur temporär erlaubt

2. Normalform

- Motivation:
  Man möchte verhindern, dass Attribute nicht vom gesamten Schlüssel voll funktional abhängig sind, sondern nur von einem Teil davon.
- Beispiel:

LNr	LName	LStadt	LLand	Ware	Preis
103	Huber	Berlin	Deutschland	Schraube	50
103	Huber	Berlin	Deutschland	Draht	20
103	Huber	Berlin	Deutschland	Nagel	40
…	…	…	…	…	…
762	Maier	Zürich	Schweiz	Nagel	45
762	Maier	Zürich	Schweiz	Schraube	55

Konsequenz: In den abhängigen Attributen muss dieselbe Information immer wiederholt werden

2. Normalform

- Dies fordert man vorerst nur für Nicht-Schlüssel-Attribute (für die anderen z.T. schwieriger, siehe unten Boyce-Codd)

- Definition
  Ein Schema ist in zweiter Normalform, wenn jedes Attribut
  - voll funktional abhängig von ***allen*** Schlüsselkandidaten

  oder
  - prim ist

- Beobachtung:
  Zweite Normalform kann nur verletzt sein, wenn...

  ...ein zusammengesetzter Schlüssel (-Kandidat) existiert
  ...und wenn nicht-prime Attribute existieren

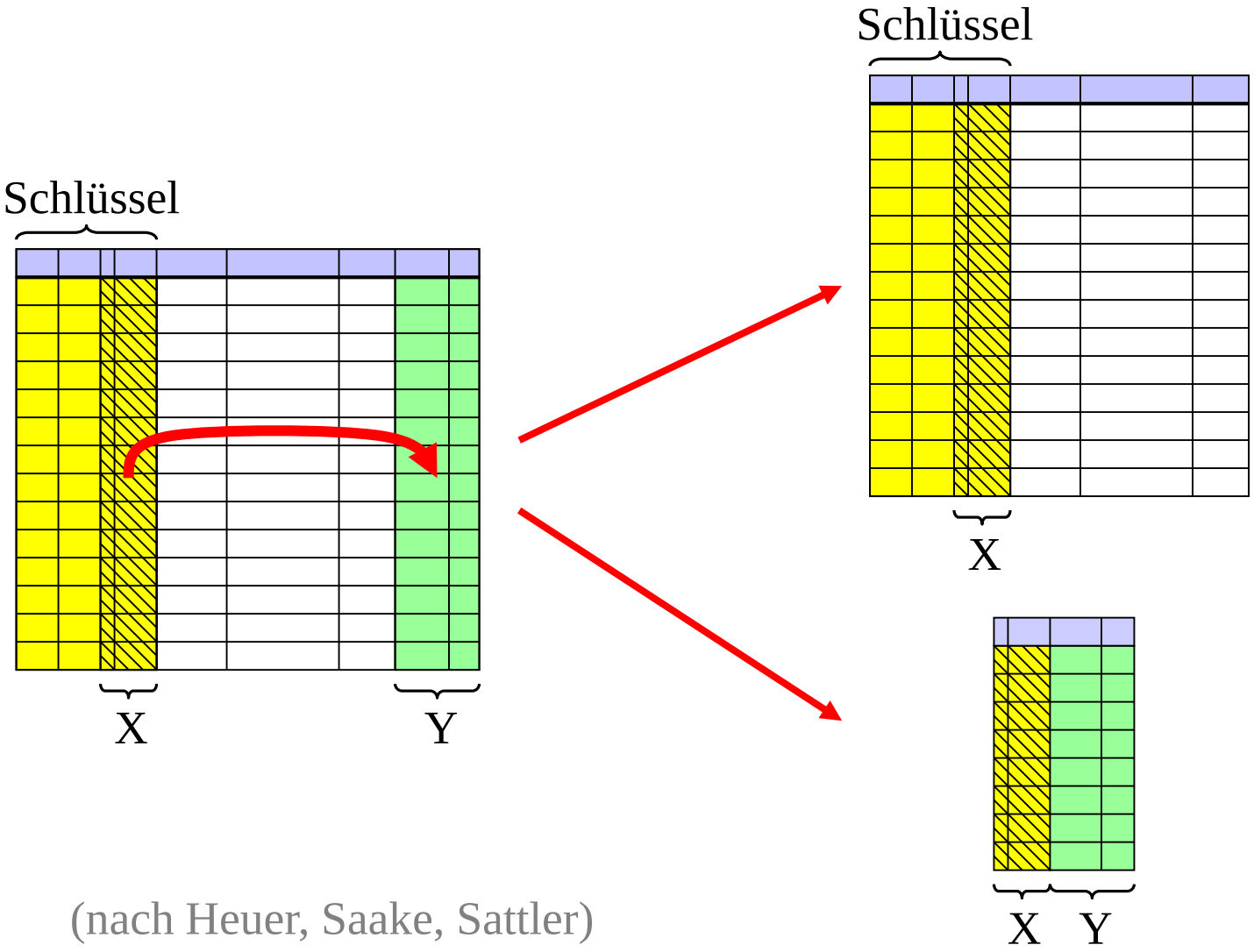
2. Normalform

- Zur Transformation in 2. Normalform spaltet man das Relationenschema auf:
  - Attribute, die voll funktional abhängig vom Schlüssel sind, bleiben in der Ursprungsrelation R
  - Für alle Abhängigkeiten $X_i \rightarrow Y_i$ von einem Teil eines Schlüssels ($X_i \subset S$) geht man folgendermaßen vor:
    - Lösche die Attribute Y_i aus R
    - Gruppiere die Abhängigkeiten nach gleichen linken Seiten X_i
    - Für jede Gruppe führe eine neue Relation ein mit allen enthaltenen Attributen aus X_i und Y_i
    - X_i wird Schlüssel in der neuen Relation

2. Normalform

- Beispiel: partielle Abhängigkeiten
  Lieferant (LNr, LName, LStadt, LLand, Ware, Preis)
- Vorgehen:
  - LName, LStadt und LLand werden aus Lieferant gelöscht
  - Gruppierung:
    Nur eine Gruppe mit LNr auf der linken Seite
    - es könnten Attribute von Ware abhängen (2. Gruppe)
  - Erzeugen einer Relation mit LNr, LName, LStadt und LLand
- Ergebnis: **Lieferung** (LNr, Ware, Preis)
  LieferAdr (LNr, LName, LStadt, LLand)

Grafische Darstellung



(nach Heuer, Saake, Sattler)