Grundlagen: Datenbanken

Zentralübung / Wiederholung / Fragestunde

Linnea Passing Harald Lang

gdb@in.tum.de

WiSe 2017 / 2018

Diese Folien finden Sie online.

Die Mitschrift stellen wir im Anschluss online.

Agenda

- Hinweise zur Klausur
- Stoffübersicht/-Diskussion
- Wiederholung + Übung
  - Mehrbenutzersynchronisation
  - Erweiterbares Hashing
  - Anfragebearbeitung/-optimierung
  - Datenbankentwurf
  - Relationale Algebra
  - Relationale Entwurfstheorie

Hinweise zur Klausur

Termine

- 1. Klausurtermin - **Mi. 28.02.2018, 8:00 bis 9:30 Uhr**
- Notenbekanntgabe / Anmeldung zu Einsicht - **Mi. 7.03.2018 (ab Mittag)**
- Einsicht - **Do. 8.03.2018 (Nachmittags)**
- Anmeldung zur 2. Klausur von - **ab 10.03.2018, bis ... ? (TBA)**
- 2. Klausurtermin - **TBA**

Verschiedenes

- **Raumbekanntgabe**, via TUMonline sowie in Moodle
- 90 Minuten / 90 Punkte
- Sitzplatzvergabe (Aushang: $MatrNr \mapsto Sitzplatz$, KEINE Namensnennung)
- Betrugsfälle
- Notenbekanntgabe
- Einsichtnahme (Instruktionen in Moodle, nach Notenbekanntgabe)
- Bonus: Gilt für beide Klausuren.

Stoffübersicht (1)

Datenbankentwurf / ER-Modellierung

- ER-Diagramme, Funktionalitäten, Min-Max, Übersetzung ER $\leftrightarrow$ Relational, Schemavereinfachung/-verfeinerung

Das Relational Modell

- Stichworte: Schema, Instanz/Ausprägung, Tupel, Attribute,...
- Anfragesprachen
  - Relationale Algebra
    - RA-Operatoren: Projektion, Selektion, Join (Theta, Natural, Outer, Semi, Anti), Kreuzprodukt, Mengendifferenz/-vereinigung/-schnitt, Division
  - Tupelkalkül, Domänenkalkül

Stoffübersicht (2)

SQL

- ...

Relationale Entwurfstheorie

- Definitionen:
  - Funktionale Abhängigkeiten (FDs), Armstrong-Axiome (+Regeln), FD-Hülle, Kanonische Überdeckung, Attribut-Hülle, Kandidaten-/Superschlüssel, Mehrwertige Abhängigkeiten (MVDs), Komplementregel, Triviale FDs/MVDs,...
- Normalformen: 1., 2., 3.NF, BCNF und 4. NF
- Zerlegung von Relationen
  - in 3.NF mit dem Synthesealgorithmus
  - in BCNF/4.NF (zwei Varianten des Dekompositionsalgorithmus)
  - Stichworte: Verlustlos, Abhängigkeitsbewahrend

Stoffübersicht (3)

- **Physische Datenorganisation**
  - Speicherhierarchie
  - HDD/RAID
  - TID-Konzept
  - Indexstrukturen (Bäume, Hashing)

- **Anfragebearbeitung**
  - Kanonische Übersetzung (SQL $\rightarrow$ Relationale Algebra)
  - Logische Optimierung (in relationaler Algebra)
    - Frühzeitige Selektion, Kreuzprodukte durch Joins ersetzen, Joinreihenfolge
  - Implementierung relationaler Operatoren
    - ...
    - Nested-Loop-Join
    - Sort-Merge-Join
    - Hash-Join
    - Index-Join

Stoffübersicht (4)

- **Transaktionsverwaltung**
  - BOT, read, write, commit, abort
  - Rollback (R1 Recovery)
  - ACID-Eigenschaften
- **Fehlerbehandlung (Recovery)**
  - Fehlerklassifikation (R1 - R4)
  - Protokollierung: Redo/Undo, physisch/logisch, Before/After-Image, WAL, LSN
  - Pufferverwaltung: Seite, FIX, Ersetzungsstrategie steal/¬steal, Einbringstrategie force/¬force
  - Wiederanlauf nach Fehler, Fehlertoleranz des Wiederanlaufs, Sicherungspunkte
- **Mehrbenutzersynchronisation**
  - Formale Definition einer Transaktion (TA)
  - Historien (Schedules)
    - Konfliktoperationen, (Konflikt-)Äquivalenz, Eigenschaften von Historien
  - Datenbank-Scheduler
    - pessimistisch (sperrbasiert, zeitstempelbasiert), optimistisch

Mehrbenutzersynchronisation

Transaktionen (High-Level)

- Ein Programm, das auf einem Datenbestand arbeitet.
  - Beispiele: Banküberweisung, Online-Bestellung, Ausleihe (Bib.)
- Daten werden gelesen, verarbeitet (Programmlogik) und geschrieben.

Atomarität

- Eine Transaktion überführt eine Datenbank von einem konsistenten Zustand in einen wiederum konsistenten Zustand.
- Zwischenzeitlich kann die Datenbank in einem inkonsistenten Zustand sein.
- Atomarität (*Alles-oder-nichts-Eigenschaft*):
  - Es werden entweder alle Änderungen übernommen, oder keine.
  - Schlägt während der Ausführung eine Operation fehl, werden alle bisherigen Änderungen in den Ausgangszustand zurück gesetzt.

Transaktionen (aus Sicht des Datenbanksystems)

Eine Transaktion T_i besteht aus folgenden **elementaren Operationen**:

$r_i(A)$	-	**Lesen** des Datenobjekts A
$w_i(A)$	-	**Schreiben** des Datenobjekts A
a_i	-	**Abort** (alle Änderungen rückgängig machen)
c_i	-	**Commit** (alle Änderungen festschreiben)

Die letzte Operation ist entweder ein **commit** oder ein **abort**.

Die dahinterliegende Programmlogik ist hier nebensächlich.

Historie (Schedule)

Eine Historie spezifiziert eine **zeitliche Abfolge von Elementaroperationen** mehrerer **parallel laufender Transaktionen** (*verzahnte Ausführung*).

$$H = r_1(A), r_2(C), w_1(A), w_2(C), r_1(B), w_1(B), r_2(A), w_2(A), c_1, c_2$$

*Eine Historie umfasst nicht zwangsläufig eine totale Ordnung ALLER Operationen, aber mindestens die der **Konfliktoperationen**.*

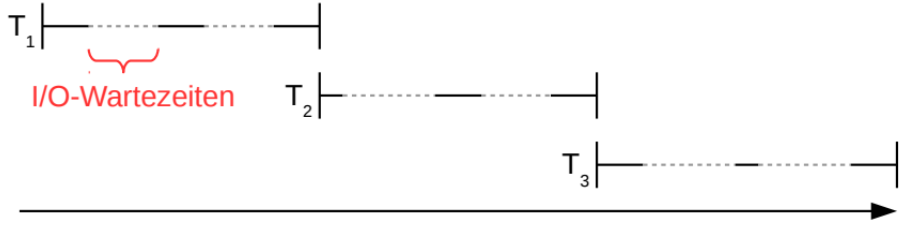
Konfliktoperationen

Zwei Operationen (verschiedener aktiver Transaktionen) auf dem selben Datum stehen zueinander in Konflikt, gdw. mindestens eine Operation schreibend ist.

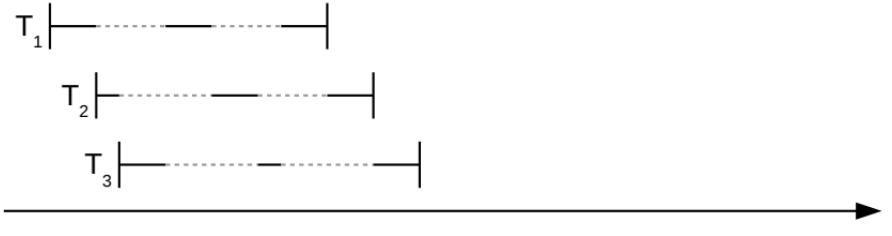
- **Unkontrollierte Nebenläufigkeit** kann zu Inkonsistenzen führen:
  - *lost update*
  - *dirty read*
  - *non-repeatable read*
  - *phantom problem*

Serielle vs. Parallele Ausführung

Eine **serielle Ausführung** verhindert all diese Probleme, da zu jedem Zeitpunkt maximal eine Transaktion aktiv ist und somit keine Konflikte auftreten können.



Eine verzahnte **parallele Ausführung** (im Mehrbenutzerbetrieb) ist effizienter.



Serialisierbarkeit (Konzept)

... soll die Vorzüge der seriellen Ausführung (**Isolation**) mit den Vorteilen des Mehrbenutzerbetriebs (**höherer Durchsatz**) kombinieren.

Serialisierbarkeit

Beispiel (Überweisung von A nach B und von C nach A):

$H = r_1(A), r_2(C), w_1(A), w_2(C), r_1(B), w_1(B), r_2(A), w_2(A), c_1, c_2$

H:

T_1	T_2
$r_1(A)$	
	$r_2(C)$
$w_1(A)$	
	$w_2(C)$
$r_1(B)$	
$w_1(B)$	
	$r_2(A)$
	$w_2(A)$
c_1	
	c_2

$\equiv$

H':

T_1	T_2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
	$r_2(C)$
$w_1(B)$	
c_1	
	$r_2(C)$
	$w_2(C)$
	$r_2(A)$
	$w_2(A)$
	c_2

Serialisierbarkeitsgraph SG(H):

$H \equiv H'$ gdw. Konfliktoperationen in der gleichen Reihenfolge.

H ist **(konflikt-) serialisierbar**.

Serialisierbarkeitstheorem

H ist **serialisierbar**, gdw. SG(H) azyklisch ist.

Weitere Eigenschaften von Historien

Rücksetzbar (RC)

- Commit der schreibenden Transaktion T_j muss vor dem Commit der lesenden Transaktion T_i durchgeführt werden.
- $c_j <_H c_i$

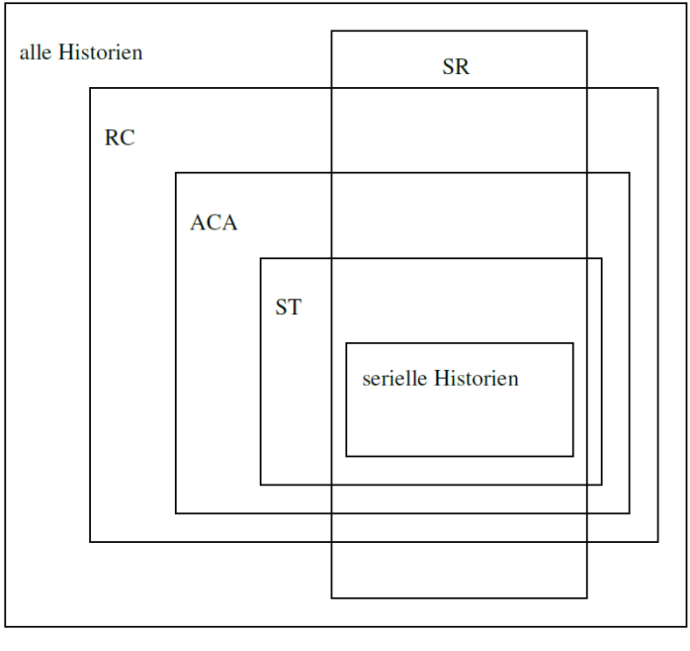
Vermeidet kaskadierendes Rücksetzen (ACA)

- Es wird erst gelesen, wenn die Änderungen der schreibenden Transaktion T_j festgeschrieben wurden (Commit).
- $c_j <_H r_i$

Strikt (ST)

- Wie ACA, verhindert aber zusätzlich blindes Schreiben (ohne vorheriges Lesen).
- $a_j <_H o_i$ oder $c_j <_H o_i$ (Operation o= r oder w)

Eigenschaften von Historien (Zusammenhang)



Eigenschaften von Historien: Übung

H:

Schritt	T_1	T_2	T_3	T_4
1	$w(A)$			
2				$r(B)$
3		$w(A)$		
4				$w(B)$
5	c			
6				c
7		$w(B)$		
8		c		
9			$r(B)$	
10			$w(C)$	
11			c	

wahr	falsch	Aussage
		$H \in SR$
		$H \in RC$
		$H \in ACA$
		$H \in ST$

Eigenschaften von Historien: Übung (2)

H:

Schritt	T_1	T_2	T_3
1	$r(A)$		
2		$w(A)$	
3		$r(B)$	
4	$w(B)$		
5	c		
6		c	
7			$r(A)$
8			$w(A)$
11			c

wahr	falsch	Aussage
		$H \in RC$
		$H \in ACA$
		$H \in ST$
		$H \in SR$

Datenbank-Scheduler

Der Datenbank-Scheduler **ordnet** die (eingehenden) **Elementaroperationen** der Transaktionen so, dass die resultierende Historie bestimmte Eigenschaften hat.

Er implementiert ein Synchronisationsverfahren und sorgt so für **kontrollierte Nebenläufigkeit**.

Synchronisationsverfahren

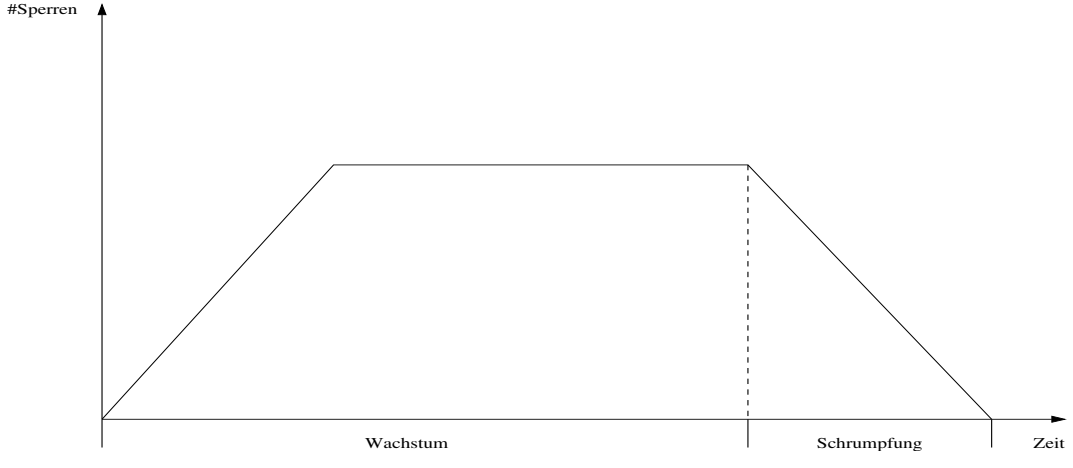
Pessimistisch

- Sperrbasiert
  - **2PL**
  - **Strenges 2PL**
- Zeitstempel-basiert

Optimistisch

- inkl. abgeschwächter Form: Snapshot Isolation

Zwei-Phasen-Sperrprotokoll (2PL)



Strenges 2PL

Verklemmung (Deadlock)

Ein Ablauf zweier parallel laufender TAs:

Schritt	T_1	T_2	Bemerkung
1.	BOT		
2.		BOT	
3.	lockX(A)		
4.		lockX(B)	
5.	w(A)		
6.		w(B)	
7.	lockS(B)		Wartet auf T_2...
8.		lockS(A)	Wartet auf T_1...

Zwei-Phasen-Sperrprotokoll (2PL)

Deadlockbehandlung

- **Vermeidung** durch preclaiming
- **Vermeidung** durch Zeitstempel
  - wound-wait
  - wait-die
- **Erkennung** durch Wartegraph

Erweiterbares Hashing

Erweiterbares Hashing

Hashfunktion ***h*: *S*** → ***B***

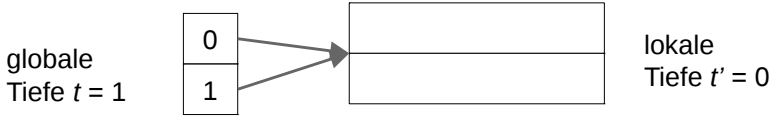
Schlüssel — Bucket

wir betrachten die **Binärdarstellung** des Hashwerts

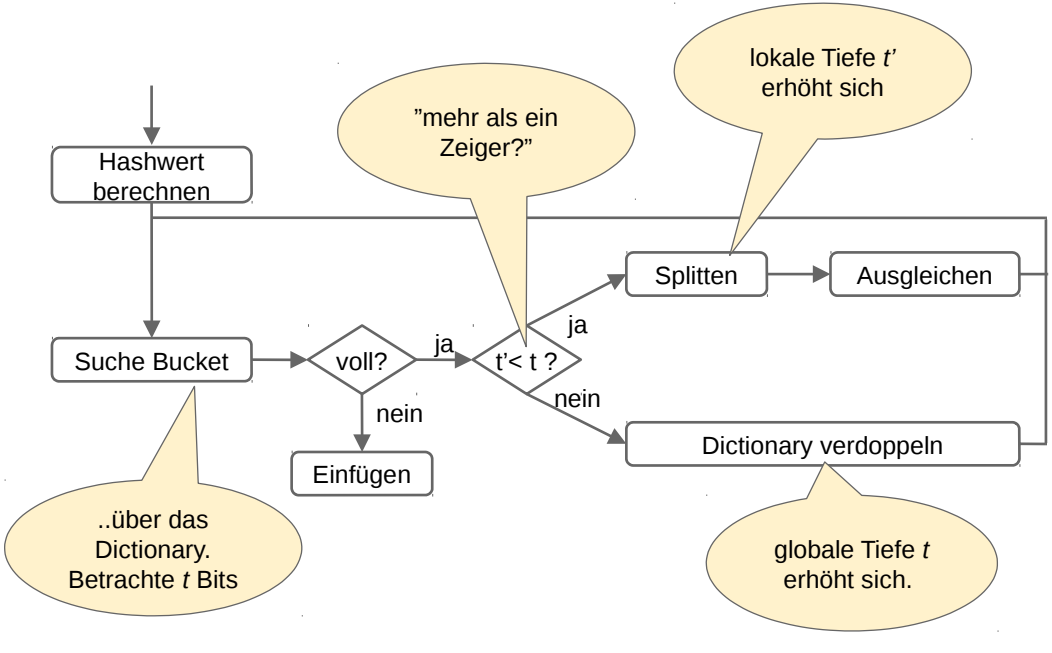
$h(x) = dp$

unbenutzte Bits

Anzahl betrachteter Bits
= globale Tiefe des Dictionaries

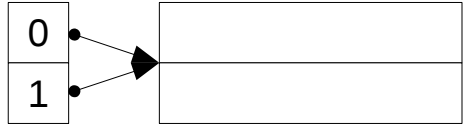


Erweiterbares Hashing / Einfügen



Übung: Erweiterbares Hashing / Einfügen

x	*h(x)*
A	1100
B	0100
C	1101
D	1010



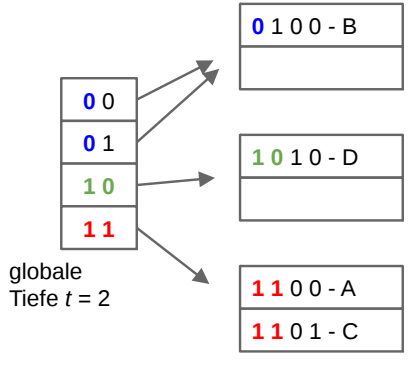
Übung: Erweiterbares Hashing / Einfügen

x	*h(x)*
A	1100
B	0100
C	1101
D	1010

Erweiterbares Hashing / Lösung

x	*h(x)*
A	1100
B	0100
C	1101
D	1010

d *p*



Anfrageoptimierung

Übung: Anfrageoptimierung

Geben Sie die kanonische Übersetzung der folgenden SQL-Anfrage an und optimieren Sie diese logisch:

```
SELECT DISTINCT s.name
  FROM studenten s, hören h, vorlesungen v
 WHERE s.matrnr = h.matrnr
   AND h.vorlnr = v.vorlnr
   AND v.titel = 'Grundzüge'
```

Übung: Anfrageoptimierung (2)

Angenommen

- $|s| = 10000$
- $|h| = 20 * |s| = 200000$
- $|v| = 1000$
- 10% der Studenten haben 'Grundzüge' gehört

Dann ergeben sich

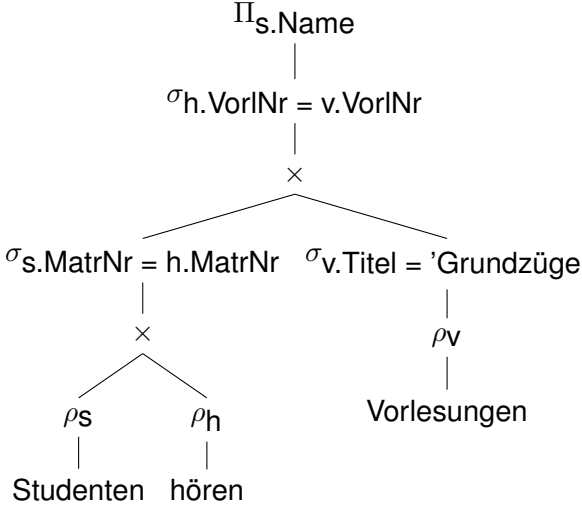
- $|s \times h \times v| = 10000 \cdot 20 \cdot 10000 \cdot 1000 = 2 \cdot 10^{12}$

Nach der Selektion verbleiben noch

- $|\sigma_p(s \times h \times v)| = 0,1 \cdot |s| = 1000$

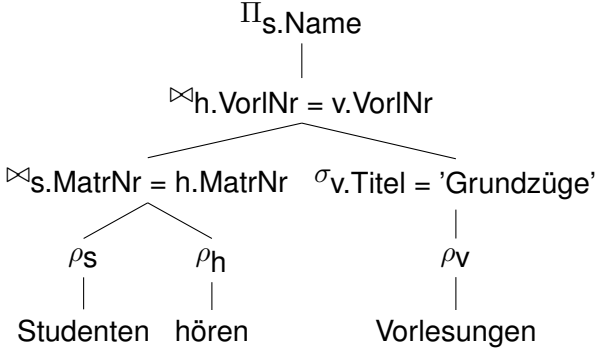
Übung: Anfrageoptimierung (3)

Optimierung 1: Selektionen frühzeitig ausführen (*push selections*):



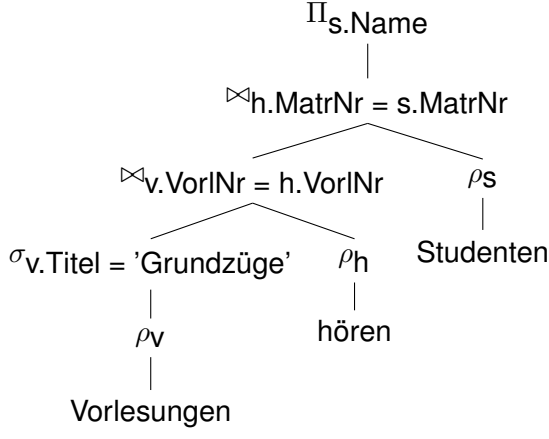
Übung: Anfrageoptimierung (4)

Optimierung 2: Kreuzprodukte durch Joins ersetzen (*introduce joins*):



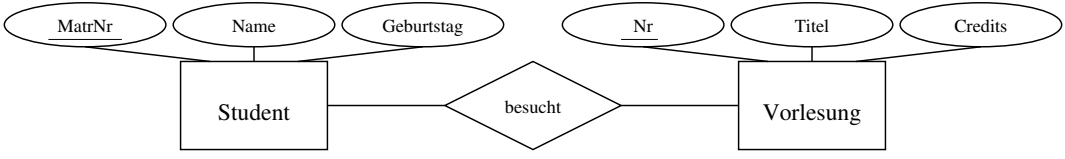
Übung: Anfrageoptimierung (5)

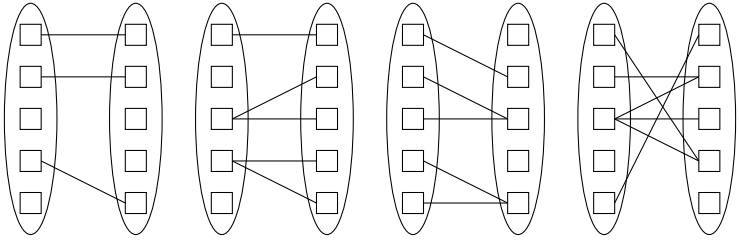
Optimierung 3: Joinreihenfolge optimieren (*join order optimization*), so dass die Zwischenergebnismengen möglichst klein sind:



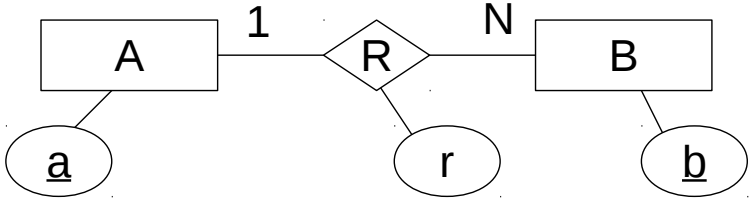
Datenbankentwurf

Datenbankentwurf

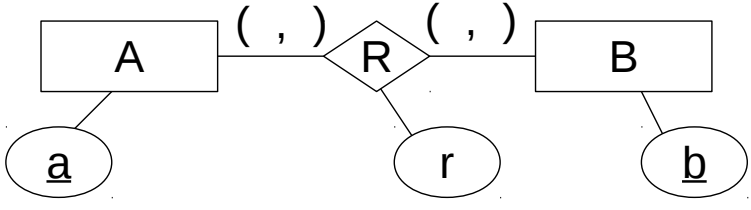




ER-Modell in Schema überführen und verfeinern



(Min,Max) - Angaben



R		
a	b	r

Relationale Algebra

Algebraische Operatoren:

Projektion	$\Pi_{A_1,\ldots,A_n}$
Selektion	σ_p
Kreuzprodukt	$\times$
Verbund (Join)	$\bowtie_\theta$, $\rtimes_\theta$, $\ltimes_\theta$, $\overline{\rtimes}_\theta$, $\overline{\ltimes}_\theta$, $\overline{\bowtie}_\theta$, $\triangleright_\theta$, $\triangleleft_\theta$
Mengenoperationen	$\cup$, $\cap$, $\setminus$
Division	$\div$
Gruppierung/Aggregation	$\Gamma_{A_1,\ldots,A_n\,;\,a_1:f_1,\ldots,a_m:f_m}$
Umbenennung	ρ_N, oder $\rho_{a_1 \leftarrow b_1,\ldots,a_n \leftarrow b_n}$

Anmerkung: Natural-Join vs. allgemeiner Theta-Join

	Natural	Theta
Inner	$\bowtie$	$\bowtie_\theta$
Outer	$\mathbin{⟗}$, $\mathbin{⟕}$, $\mathbin{⟖}$	$\mathbin{⟗_\theta}$, $\mathbin{⟕_\theta}$, $\mathbin{⟖_\theta}$
Semi	$\ltimes$, $\rtimes$	$\ltimes_\theta$, $\rtimes_\theta$
Anti	$\triangleright$, $\triangleleft$	$\triangleright_\theta$, $\triangleleft_\theta$

- Natural
  - Implizite Gleichheitsbedingung auf gleichnamigen Attributen
  - Die gleichnamigen Attribute tauchen im Ergebnis nur einmal auf (inner und outer).
- Theta
  - Explizite (beliebige) Joinbedingung: θ.
  - Im Falle von Inner- und Outer-Join werden alle Attribute der beiden Eingaberelationen in das Ergebnis projiziert.

Übung: Relationale Algebra (1)

Finde Studenten (nur Namen ausgeben), die im gleichen Semester sind wie Feuerbach.

Übung: Relationale Algebra (2)

Finde Studenten (nur MatrNr ausgeben), die alle Vorlesungen gehört haben.

Relationale Entwurfstheorie

Relationale Entwurftheorie

Funktionale Abhängigkeiten (kurz FDs, für functional dependencies):

- Seien α und β Attributmengen eines Schemas $\mathcal{R}$.
- Wenn auf $\mathcal{R}$ die FD $\alpha \rightarrow \beta$ definiert ist, dann sind nur solche Ausprägungen R zulässig, für die folgendes gilt:
  - Für alle Paare von Tupeln $r, t \in R$ mit $r.\alpha = t.\alpha$ muss auch gelten $r.\beta = t.\beta$.

Übung: Relationenausprägung vervollständigen

Gegen seien die folgende Relationenausprägung und die funktionalen Abhängigkeiten. Bestimmen Sie zunächst x und danach y, sodass die FDs gelten.

$$\begin{aligned} B &\rightarrow A \\ AC &\rightarrow D \end{aligned}$$

A	B	C	D
7	3	5	8
x	4	2	8
7	3	6	9
1	4	2	y

Funktionale Abhängigkeiten

Seien α,β,γ,$\delta \subseteq \mathcal{R}$

Axiome von Armstrong:

- *Reflexivität:*
  Falls $\beta \subseteq \alpha$, dann gilt immer $\alpha \rightarrow \beta$
- *Verstärkung*:
  Falls $\alpha \rightarrow \beta$ gilt, dann gilt auch $\alpha\gamma \rightarrow \beta\gamma$
- *Transitivität*:
  Falls $\alpha \rightarrow \beta$ und $\beta \rightarrow \gamma$ gelten, dann gilt auch $\alpha \rightarrow \gamma$

Mithilfe dieser Axiome können alle *geltenden* FDs hergeleitet werden.

Sei F eine FD-Menge. Dann ist F^+ die Menge aller geltenden FDs (Hülle von F)

Funktionale Abhängigkeiten

Nützliche und vereinfachende Regeln:

- *Vereinigungsregel*:
  Falls $\alpha \to \beta$ und $\alpha \to \gamma$ gelten, dann gilt auch $\alpha \to \beta\gamma$
- *Dekompositionsregel*:
  Falls $\alpha \to \beta\gamma$ gilt, dann gilt auch $\alpha \to \beta$ und $\alpha \to \gamma$
- *Pseudotransitivitätsregel*:
  Falls $\alpha \to \beta$ und $\gamma\beta \to \delta$ gelten, dann gilt auch $\gamma\alpha \to \delta$

Schlüssel

- Schlüssel identifizieren jedes Tupel einer Relation $\mathcal{R}$ eindeutig.
- Eine Attributmenge $\alpha \subseteq \mathcal{R}$ ist ein **Superschlüssel**, gdw. $\alpha \rightarrow \mathcal{R}$
- Ist α zudem noch *minimal*, ist es auch ein **Kandidatenschlüssel** (meist mit κ bezeichnet)
  - Es existiert also kein $\alpha' \subset \alpha$ für das gilt: $\alpha' \rightarrow \mathcal{R}$

- I.A. existieren mehrere Super- und Kandidatenschlüssel.
- Man muss sich bei der Realisierung für einen Kandidatenschlüssel entscheiden, dieser wird dann **Primärschlüssel** genannt.
- Der triviale Schlüssel $\alpha = \mathcal{R}$ existiert immer.

Übung: Schlüsseleigenschaft von Attributmengen ermitteln

- Ob ein gegebenes α ein Schlüssel ist, kann mithilfe der Armstrong Axiome ermittelt werden (i.A. zu aufwendig!)
- Besser: Die **Attributhülle** $AH(\alpha)$ bestimmen.

- Beispiel: $\mathcal{R} = \{ A , B , C , D \}$, mit $F_{\mathcal{R}} = \{AB \to CD, B \to C, D \to B\}$

$AH(\{D\})$:

$AH(\{A, D\})$:

$AH(\{A, B, D\})$:

Mehrwertige Abhängigkeiten

multivalued dependencies (MVDs)

"Halb-formal":

- Seien α und β disjunkte Teilmengen von $\mathcal{R}$
- und $\gamma = (\mathcal{R} \backslash \alpha) \backslash \beta$
- dann ist β mehrwertig abhängig von α ($\alpha \twoheadrightarrow \beta$), wenn in jeder gültigen Ausprägung von $\mathcal{R}$ gilt:
- Bei zwei Tupeln mit gleichem α-Wert kann man die β-Werte vertauschen, und die resultierenden Tupel müssen auch in der Relation enthalten sein.

Wichtige Eigenschaften:

- Jede FD ist auch eine MVD (gilt i.A. nicht umgekehrt)
- wenn $\alpha \twoheadrightarrow \beta$, dann gilt auch $\alpha \twoheadrightarrow \gamma$ (Komplementregel)
- $\alpha \twoheadrightarrow \beta$ ist trivial, wenn $\beta \subseteq \alpha$ ODER $\alpha \cup \beta = \mathcal{R}$ (also $\gamma = \emptyset$)

Beispiel: Mehrwertige Abhängigkeiten

Beispiel: $R = \{\text{Professor}, \text{Vorlesung}, \text{Assistent}\}$

ProfessorIn	Vorlesung	AssistentIn
K	GDB	Linnea
K	WebDB	Linnea

Normalformen: 1NF $\supset$ 2NF $\supset$ 3NF $\supset$ BCNF $\supset$ 4NF

- **1. NF**: Attribute haben nur atomare Werte, sind also nicht mengenwertig.
- **2. NF**: Jedes Nichtschlüsselattribut (NSA) ist voll funktional abhängig von jedem Kandidatenschlüssel.
  - β hängt **voll funktional** von α ab ($\alpha \overset{\bullet}{\rightarrow} \beta$), gdw. $\alpha \rightarrow \beta$ und es existiert kein $\alpha' \subset \alpha$, so dass $\alpha' \rightarrow \beta$ gilt.
- **3. NF**: Frei von transitiven Abhängigkeiten (*in denen NSAe über andere NSAe vom Schlüssel abhängen*).
  - für alle geltenden nicht-trivialen FDs $\alpha \rightarrow \beta$ gilt entweder
    - α ist ein Superschlüssel, oder
    - jedes Attribut in β ist in einem Kandidatenschlüssel enthalten
- **BCNF**: Die linken Seiten (α) aller geltenden nicht-trivialen FDs sind Superschlüssel.
- **4. NF**: Die linken Seiten (α) aller geltenden nicht-trivialen MVDs sind Superschlüssel.

Übung: Höchste NF bestimmen

$$\mathcal{R} : \{[\ A, B, C, D, E\ ]\}$$

$$\begin{aligned} A &\rightarrow BCDE \\ AB &\rightarrow C \end{aligned}$$

- ○ 1. NF
- ○ 2. NF
- ○ 3. NF
- ○ BCNF
- ○ 4. NF
- ○ keine der angegebenen

Übung: Höchste NF bestimmen (2)

$\mathcal{R} : \{[\ A, B, C, D, E\ ]\}$

$$A \rightarrow BCDE$$
$$B \rightarrow C$$

- ○ 1. NF
- ○ 2. NF
- ○ 3. NF
- ○ BCNF
- ○ 4. NF
- ○ keine der angegebenen

Schema in 3. NF überführen

Synthesealgorithmus

- Eingabe:
  - **Kanonische Überdeckung** $\mathcal{F}_c$
    - Linksreduktion
    - Rechtsreduktion
    - FDs der Form $\alpha \to \emptyset$ entfernen (sofern vorhanden)
    - FDs mit gleicher linke Seite zusammenfassen
- Algorithmus:
  1. Für jede FD $\alpha \to \beta$ in $\mathcal{F}_c$ forme ein Unterschema $\mathcal{R}_\alpha = \alpha \cup \beta$, ordne $\mathcal{R}_\alpha$ die FDs $\mathcal{F}_\alpha := \{\alpha' \to \beta' \in \mathcal{F}_c \mid \alpha' \cup \beta' \subseteq \mathcal{R}_\alpha\}$ zu
  2. Füge ein Schema $\mathcal{R}_\kappa$ mit einem Kandidatenschlüssel hinzu
  3. Eliminiere redundante Schemata, d.h. falls $\mathcal{R}_i \subseteq \mathcal{R}_j$, verwerfe $\mathcal{R}_i$
- Ausgabe:
  - Eine Zerlegung des unsprünglichen Schemas, wo alle Schemata in 3.NF sind.
  - Die Zerlegung ist **abhängigkeitsbewahrend** und **verlustfrei**.

Übung: Synthesealgorithmus

$$\mathcal{R} : \{[\ A, B, C, D, E, F\ ]\}$$

$$\begin{aligned} B &\rightarrow ACDEF \\ EF &\rightarrow BC \\ A &\rightarrow D \end{aligned}$$

Schema in BCNF überführen

BCNF-Dekompositionsalgorithmus (nicht abhängigkeitsbewahrend)

- Starte mit $Z = \{\mathcal{R}\}$
- Solange es noch ein $\mathcal{R}_i \in Z$ gibt, das nicht in BCNF ist:
  - Finde eine FD $(\alpha \to \beta) \in F^+$ mit
    - $\alpha \cup \beta \subseteq \mathcal{R}_i$ (FD muss in $\mathcal{R}_i$ gelten)
    - $\alpha \cap \beta = \emptyset$ (linke und rechte Seite sind disjunkt)
    - $\alpha \to \mathcal{R}_i \notin F^+$ (linke Seite ist kein Superschlüssel)
  - Zerlege $\mathcal{R}_i$ in $\mathcal{R}_{i.1} := \alpha \cup \beta$ und $\mathcal{R}_{i.2} := \mathcal{R}_i - \beta$
  - Entferne $\mathcal{R}_i$ aus Z und füge $\mathcal{R}_{i.1}$ und $\mathcal{R}_{i.2}$ ein, also $Z := (Z - \{\mathcal{R}_i\}) \cup \{\mathcal{R}_{i.1}\} \cup \{\mathcal{R}_{i.2}\}$

Schema in 4.NF überführen

4NF-Dekompositionsalgorithmus (nicht abhängigkeitsbewahrend)

- Starte mit $Z = \{\mathcal{R}\}$
- Solange es noch ein $\mathcal{R}_i \in Z$ gibt, das nicht in 4NF ist:
  - Finde eine MVD $\alpha \twoheadrightarrow \beta \in \mathcal{F}^+$ mit
    - $\alpha \cup \beta \subset \mathcal{R}_i$ (FD muss in $\mathcal{R}_i$ gelten)
    - $\alpha \cap \beta = \emptyset$ (linke und rechte Seite sind disjunkt)
    - $\alpha \rightarrow \mathcal{R}_i \notin \mathcal{F}^+$ (linke Seite ist kein Superschlüssel)
  - Zerlege $\mathcal{R}_i$ in $\mathcal{R}_{i.1} := \alpha \cup \beta$ und $\mathcal{R}_{i.2} := \mathcal{R}_i - \beta$
  - Entferne $\mathcal{R}_i$ aus Z und füge $\mathcal{R}_{i.1}$ und $\mathcal{R}_{i.2}$ ein, also $Z := (Z - \{\mathcal{R}_i\}) \cup \{\mathcal{R}_{i.1}\} \cup \{\mathcal{R}_{i.2}\}$

Übung: BCNF-Dekompositionsalgorithmus

$\mathcal{R} = \{A, B, C, D, E, F\}, F_{\mathcal{R}} = \{B \to AD, DEF \to B, C \to AE\}$

Fragen ?

Wir wünschen viel Erfolg. :-)