Grundlagen: Datenbanken

2. Zentralübung / Fragestunde

Harald Lang

Diese Folien finden Sie online.

Die Mitschrift erhalten Sie im Anschluss.

Agenda

- Hinweise zur Klausur
- Stoffübersicht/-Diskussion
- Anmerkungen
  - B-Baum / B^+-Baum
- Übung
  - Abfrageoptimierung
  - Erweiterbares Hashing
  - Mehrbenutzersynchronisation (inkl. Wiederholung)
  - Normalformen und Zerlegungsalgorithmen

Hinweise zur Klausur

Termine

- 1. Klausurtermin
  - Mi. 18.02.2015, 08:00 Uhr
- 2. Klausurtermin
  - Do. 02.04.2014, 13:00 Uhr (Anmeldung von 09.03. bis 21.03.2015)
- **Raumbekanntgabe**, jeweils (spätestens) eine Woche vorher!
  (via TUMonline sowie auf der Homepage)

Verschiedenes

- 90 Minuten / 90 Punkte
- Sitzplatzvergabe (Aushang: $MatrNr \mapsto Sitzplatz$, KEINE Namensnennung)
- Betrugsfälle
- Notenbekanntgabe (via TUMonline)
- Einsichtnahme (Instruktionen auf der Homepage, nach Notenbekanntgabe)

Soffübersicht (1)

Datenbankentwurf / **ER-Modellierung**

- .., Funktionalitäten, Min-Max, Übersetzung ER $\rightarrow$ 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**

Soffübersicht (2)

SQL

- ... darin sind Sie alle bereits Experten :-)

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

*** Es folgt noch eine Übungsaufgabe dazu.**

Soffübersicht (3)

- Physische Datenorganisation
  - Speicherhierarchie
  - HDD/**RAID**
  - TID-Konzept (slotted pages)
  - **Indexstrukturen (Bäume, Hashing*)**
- Anfragebearbeitung
  - **Kanonische Übersetzung*** (SQL $\rightarrow$ Relationale Algebra)
  - Logische **Optimierung*** (in relationaler Algebra)
  - Implementierung relationaler Operatoren
    - ...
    - **Nested-Loop-Join**
    - **Sort-Merge-Join**
    - Hash-Join
    - Index-Join

*** Es folgt noch eine Übungsaufgabe dazu.**

Soffübersicht (4)

- Transaktionsverwaltung
  - **ACID-Eigenschaften**
- ~~Recovery~~
- Mehrbenutzersynchronisation
  - **Formale Definition einer Transaktion (TA)**
  - **Historien (Schedules)***
    - Konfliktoperationen
    - (Konflikt-)Äquivalenz
    - Eigenschaften von Historien
  - **Datenbank-Scheduler***
    - pessimistisch (sperrbasiert, zeitstempelbasiert)
    - optimistisch

*** Es folgt noch eine Übungsaufgabe dazu.**

Anmerkungen zum B-Baum/B$^+$-Baum

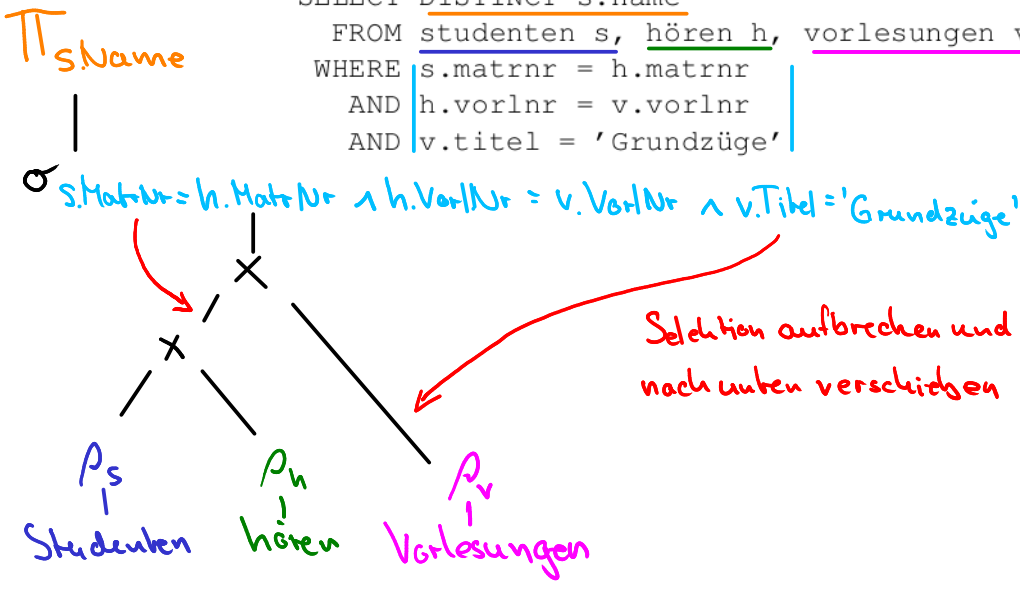
- Inkonsistenz: Folien $\longleftrightarrow$ Buch u. Übung
  - B-Baum mit Grad k
    - Parameter k gibt an (**Grad**), dass jeder Knoten mindestens k Elemente und höchstens $2k$ Elemente enthält. (nur Wurzel kann unterbelegt sein)
    - Auf den Folien wird der Parameter i genannt.
  - B$^+$-Baum vom Typ (k, k^*)
    - Parameter k^* bezieht sich auf die **Blattknoten** (mindestens k^*, höchstens $2k^*$ Elemente)
- Kurze Wiederholung: B-Baum vs. B$^+$-Baum
  - Im B$^+$-Baum werden **Daten ausschließlich in den Blattknoten** gespeichert. In den inneren Knoten nur Schlüssel.
    - Höherer Verzeigungsgrad der inneren Knoten
    - Baumhöhe wird reduziert
    - Schnellere Suche (Weg von der Wurzel zu den Blättern kürzer)
  - Die **Blattknoten** des B$^+$-Baums sind **verkettet**
    - Ermöglicht effiziente Bereichsanfragen: ...WHERE a >= 10 AND a <=100

Anfragebearbeitung/-optimierung

Anfrageoptimierung / Übung

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'
```



Anfrageoptimierung / Übung (2)

Angenommen

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

Dann ergeben sich

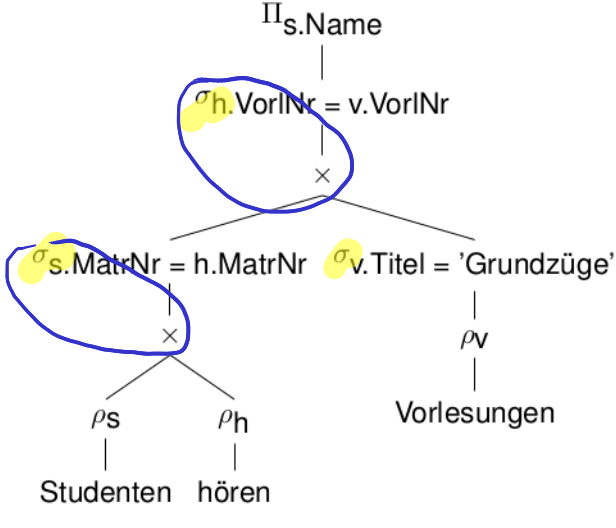
- $|s \times h \times v| = 10000 \cdot 20 \cdot 10000 \cdot 1000 = 2 \cdot 10^{12}$

Nach der Selektion verbleiben noch

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

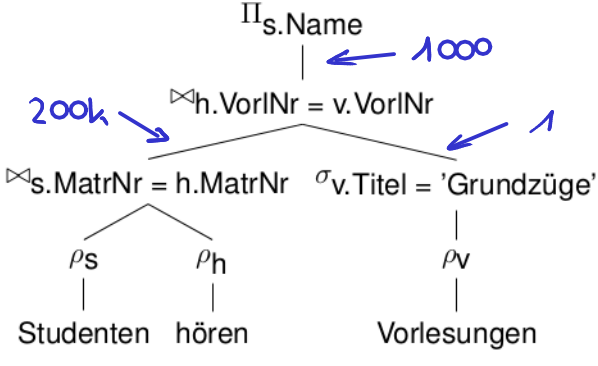
Anfrageoptimierung / Übung (3)

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



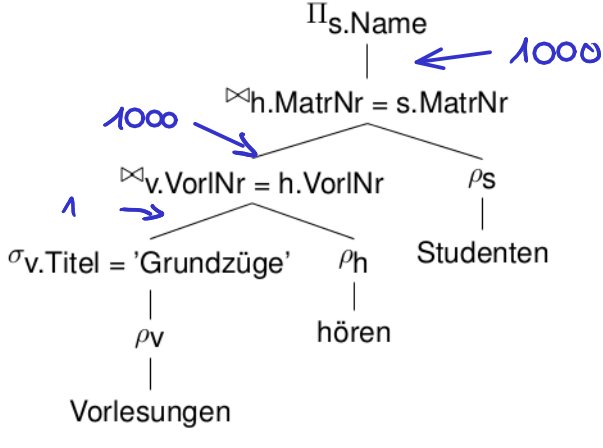
Anfrageoptimierung / Übung (4)

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



Anfrageoptimierung / Übung (5)

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



Erweiterbares Hashing

Erweiterbares Hashing

Hashfunktion ***h*: *S*** → ***B***

Schlüssel — Bucket

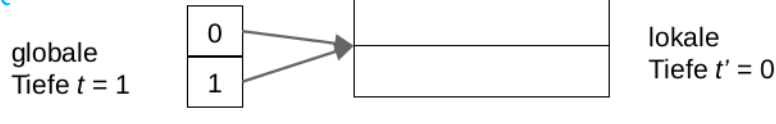
wir betrachten die **Binärdarstellung** des Hashwerts

$h(x) = dp$

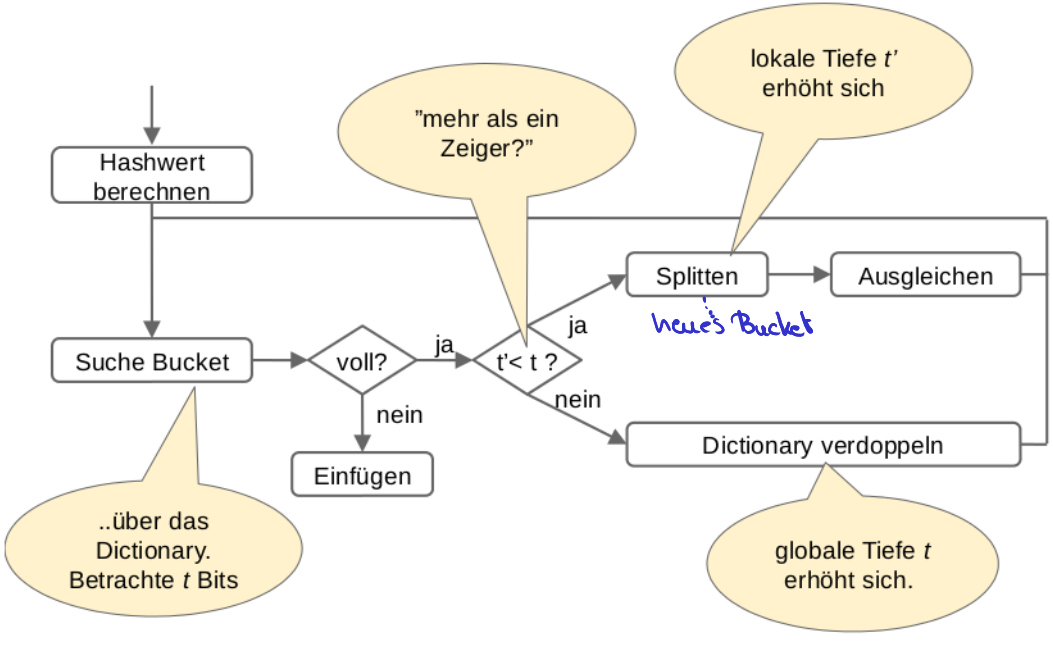
unbenutzte Bits

Anzahl betrachteter Bits
= globale Tiefe des Dictionaries

Initialer Zustand:

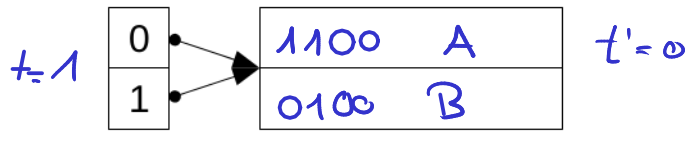


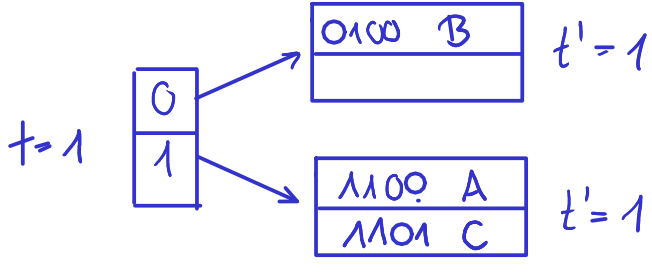
Erweiterbares Hashing / Einfügen



Erweiterbares Hashing / Einfügen (Übung)

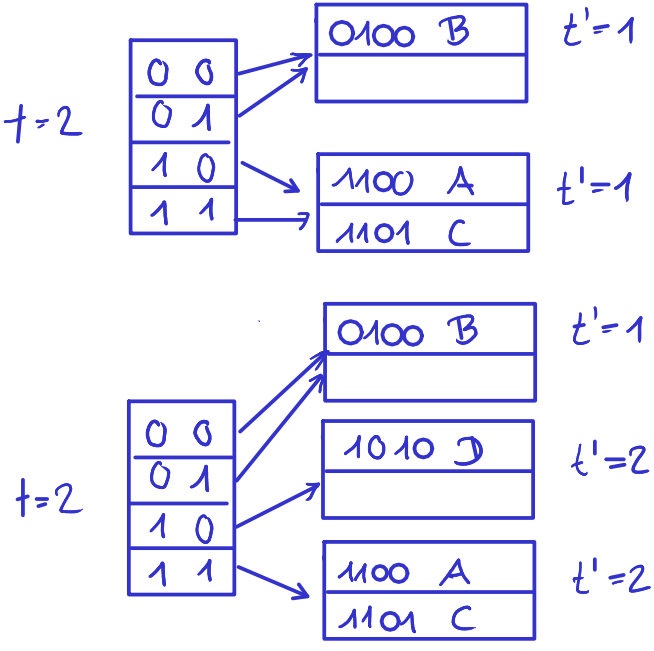
x	*h(x)*
A	1100
B	0100
C	1101
D	1010



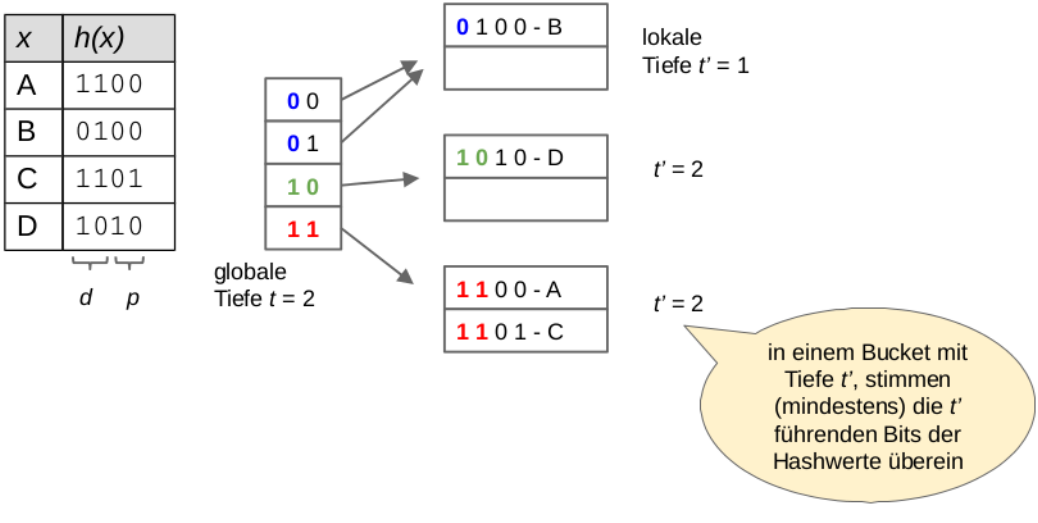


Erweiterbares Hashing / Einfügen (Übung)

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



Erweiterbares Hashing / Lösung



Mehrbenutzersynchronisation

Wiederholung + Übung

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*).

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

T_1 T_2

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

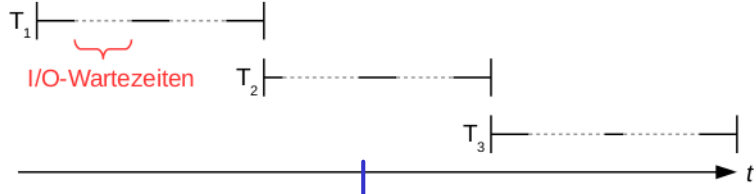
Konfliktoperationen

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

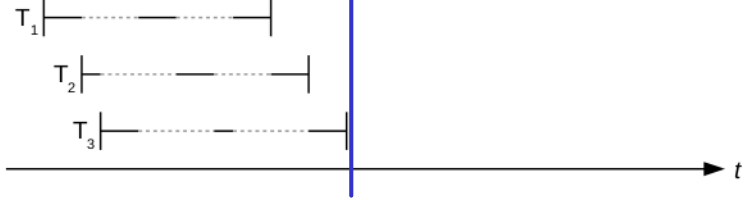
- **Unkontrollierte Nebenläufigkeit** kann zu Inkonsistenzen führen:
  - *lost update*
  - *dirty read*
  - *non-repeatable read*
  - *phantom problem*

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.



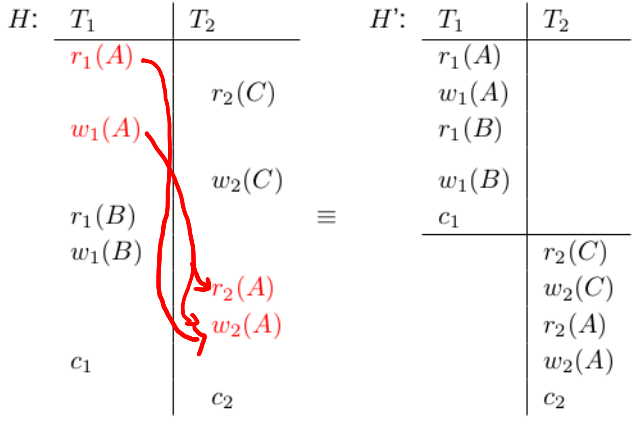
Serialisierbarkeit (Konzept)

... 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):

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



$H' = T_1 \mid T_2$

Serialisierbarkeitsgraph SG(H):

$T_1 \rightarrow T_2$

kein Zyklus ⇒ Serialisierbar

$H \equiv H'$ 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.
- $c_j <_H c_i$

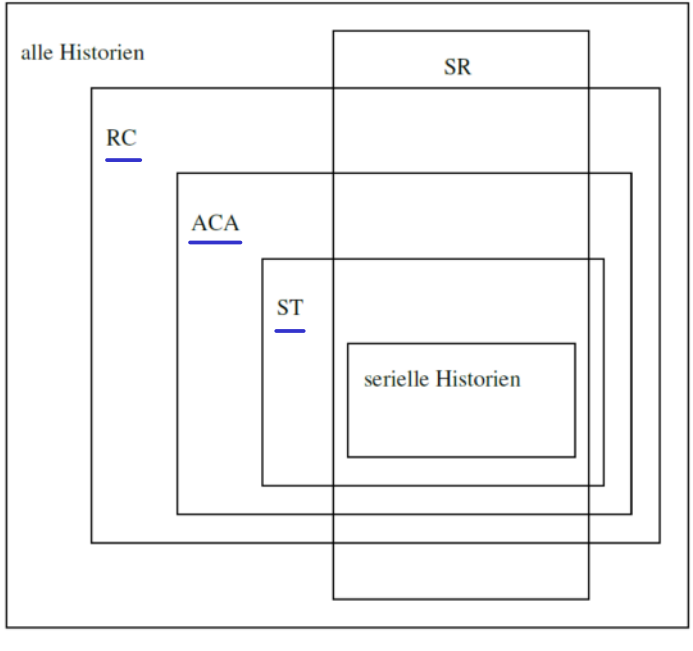
Vermeidet kaskadierendes Rücksetzen (ACA)

- 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

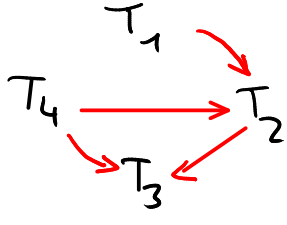
T_3 liest von T_2 und T_4

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
✓		$H \in SR$
✓		$H \in RC$
✓		$H \in ACA$
	✓	$H \in ST$

SG(H):



$H = T_1 | T_4 | T_2 | T_3$

$\equiv T_4 | T_1 | T_2 | T_3$

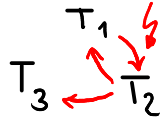
Eigenschaften von Historien: Übung (2)

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
✓		$H \in RC$
✓		$H \in ACA$
✓		$H \in ST$
	✗	$H \in SR$

SG(H):



nur T_3 liest von T_2

$c_2 <_H c_3$

RC

$c_2 <_H r_3(A)$

ACA

$w_2(A) <_H w_3(A)$

aber $c_2 <_H w_3(A)$

ST

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

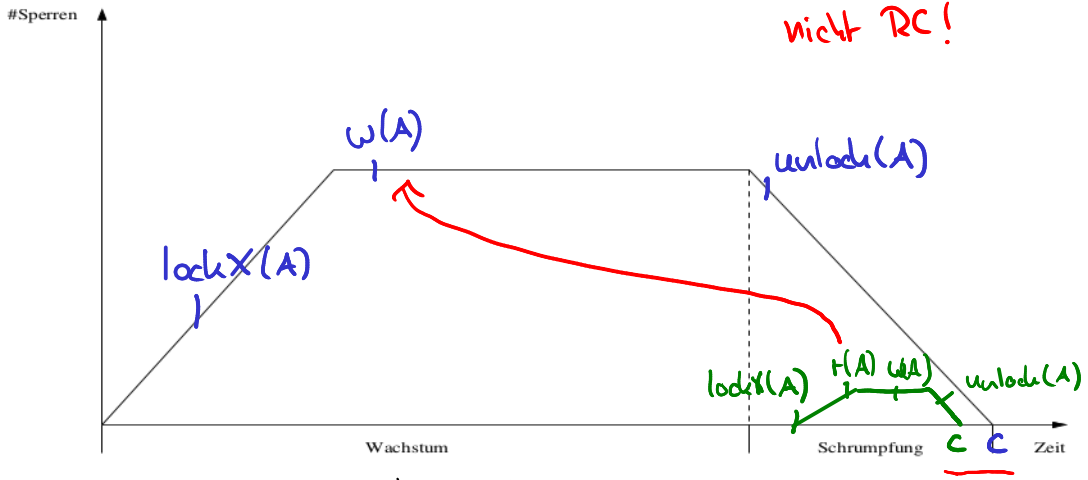
- Sperrbasiert
  - **2PL**
  - **Strenges 2PL**
- Zeitstempel-basiert

Optimistisch

- inkl. abgeschwächter Form: Snapshot Isolation

siehe Lösung Blatt 13 | 14

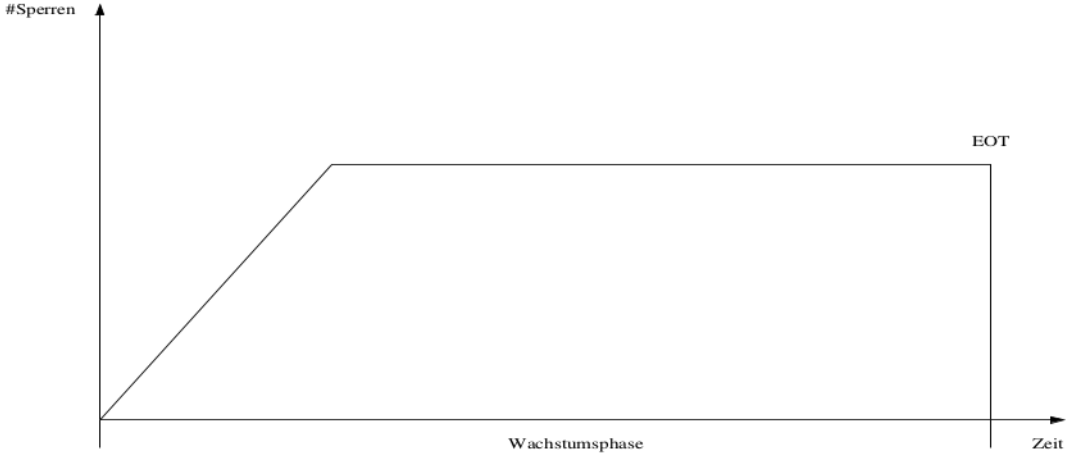
Zwei-Phasen-Sperrprotokoll (2PL)



$T_{grün}$ liest von T_{blau}

$C_{grün} <_H C_{blau}$

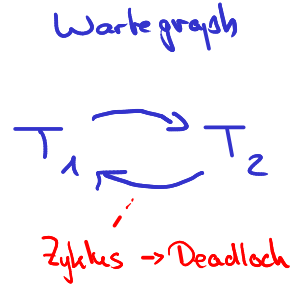
Strenges 2PL



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

$(T_{alt} \rightarrow T_{jung}) - (T_{jung} \rightarrow T_{alt})$

wound – wait

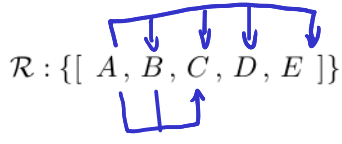
wait – die

Normalformen

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 ($\alpha \xrightarrow{\bullet} \beta$), 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-trivalen FDs sind Superschlüssel.
- **4. NF**: Die linken Seiten (α) aller geltenden nicht-trivalen MVDs sind Superschlüssel.

Höchste NF bestimmen: Übung



$$A \rightarrow BCDE$$
$$AB \rightarrow C$$

1NF: ✓

2NF: ✓

3NF: ✓

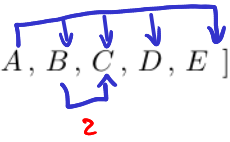
BCNF: ✓

4NF: ✓

$X = \{A\}$ $NSA_e = \{BCDE\}$

Höchste NF bestimmen: Übung (2)

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



$A \rightarrow BCDE$

$B \rightarrow C$

Lösung

1NF ✓

2NF ✓

3NF ✗

$k = \{A\}$ $NSA_e = \{BCDE\}$

FD 2 verletzt 3.NF

Schema in 3. NF bringen

Synthesealgorithmus

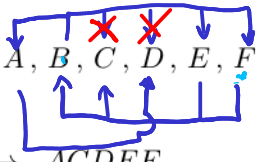
- Eingabe:
  - **Kanonische Überdeckung** $\mathcal{F}_c$
    - Linksreduktion
    - Rechtsreduktion
    - FDs der Form $\alpha \to \emptyset$ entfernen (sofern vorhanden)
    - FDs mit gleicher linke Seite zusammenfassen
- Algorithmus:
  1. Für jede FD $\alpha \to \beta$ in $\mathcal{F}_c$ forme ein 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.

Synthesealgorithmus: Übung

kanonische Überdeckung

F_c:

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



$B \rightarrow ACDEF$

$EF \rightarrow BC$

$A \rightarrow D$

LR: $EF \rightarrow BC$

RR: $B \rightarrow ACDEF$

$EF \rightarrow BC$

$A \rightarrow D$

} F_c

$K_1 = \{B\}$

$\alpha_2 = \{E, F\}$

$R_1 : \{B, A, E, F\}$

$R_2 : \{E, F, BC\}$

$R_3 : \{A, D\}$

Schema in BCNF bringen

BCNF-Dekompositionsalgorithmus

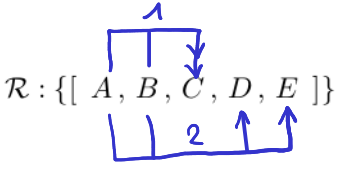
- Starte mit $Z = \{\mathcal{R}\}$
- Solange es noch ein $\mathcal{R}_i \in Z$ gibt, das nicht in BCNF ist:
  - Finde eine FD $(\alpha \to \beta) \in F^+$ mit
    - $\alpha \cup \beta \subseteq \mathcal{R}_i$
    - $\alpha \cap \beta = \emptyset$
    - $\alpha \to \mathcal{R}_i \notin F^+$
  - Zerlege $\mathcal{R}_i$ 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 bringen

4NF-Dekompositionsalgorithmus

- 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
    - $\alpha \cup \beta \subset \mathcal{R}_i$
    - $\alpha \cap \beta = \emptyset$
    - $\alpha \rightarrow \mathcal{R}_i \notin \mathcal{F}^+$
  - 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 $Z := (Z - \{\mathcal{R}_i\}) \cup \{\mathcal{R}_{i.1}\} \cup \{\mathcal{R}_{i.2}\}$

Dekompositionsalgorithmus für 4.NF: Übung



$AB \twoheadrightarrow C$

$AB \rightarrow DE$

MVD 1 ist nicht trivial und linke Seite ist kein Superschlüssel → verletzt 4.NF

Zerlege anhand MVD 1 in

$R_1 = \{A, B, C\}$ u. $R_2 = \{A, B, D, E\}$

kann nicht weiter zerlegt werden, da $AB \twoheadrightarrow C$ in R_1 trivial ist.

analog...

⚠ Schlüssel ist $\{A, B, C\}$