Zentralübung
Einsatz und Realisierung von Datenbanksystemen

Maximilian Bandle

Organisatorisches

Disclaimer

Folien enthalten wichtige Themen oder Themen, die viele Fragen aufwarfen.

Falls etwas auf den Folien nicht erwähnt ist, aber in den Übungen / Vorlesung besprochen wurde, kann daraus nicht geschlossen werden, dass es nicht in der Prüfung drankommt.

Auf den Folien dargestellte Übungen können Fehler enthalten.

Organisatorisches

Klausur

Hauptklausur
1. August 2019, 15:30 - 17:00

Wiederholungsklausur
11. Oktober 2019, 15:30 - 17:00

Durchführung
90 Minuten Bearbeitungszeit
Raumeinteilung wird rechtzeitig bekannt gegeben
Keine Hilfsmittel (Taschenrechner, DB-Buch, Spickzettel, ...) erlaubt!

ACID

Atomicity (Atomarität)

„alles oder nichts“-Prinzip: die TA wird komplett geschrieben oder garnicht

Consistency (Konsistenz)

TA hinterlässt(abort/commit) einen konsistenten Zustand der Datenbasis

Isolation (Isolation)

Nebenläufige TA dürfen sich nicht beeinflussen

Durability (Dauerhaftigkeit)

Das Ergebnis einer Transaktion bleibt dauerhaft in der Datenbank erhalten

Kapitel 10

Recovery

Recovery
Write Ahead Logging

- Schreiben der Log-Einträge vor dem Commit
- Vor Auslagerung einer Seite: Schreiben aller zugehörigen Log-Einträge

Recovery
Speicherhierarchie

Ersetzung von Puffer-Seiten

- **¬Steal:** Seiten die noch von einer Transaktion modifiziert werden müssen im Speicher verbleiben.
- **Steal:** Seiten können (fast) immer aus dem Puffer in den Speicher eingelagert werden.

Einbringen von Änderungen abgeschlossener Transaktionen

- **Force:** Änderungen werden direkt nach Durchführung gespeichert
- **¬Force:** Änderungen können im Puffer-Speicher verbleiben

Recovery

Speicherhierarchie - Auswirkungen auf Recovery

	force	¬force
¬steal	kein Undo kein Redo	kein Undo Redo
steal	Undo kein Redo	Undo Redo

Recovery

Speicherhierarchie - Auswirkungen auf Recovery

¬steal & force

– wird eine Seite von 2 TA geändert, so kann die 1. nicht comitten

¬steal & ¬force

– Seiten können nach Transaktionsende ersetzt werden, ohne dass die Änderungen in die DB übernommen werden

– Strategie bei Main Memory DBs

steal & force

– direktes Einlagern beim commit ist teuer, gesammeltes Einlagern ist günstiger

– nicht commitete Daten können festgeschrieben werden

steal & ¬force

– Änderungen von (aktiven) TAs können beim Systemabsturz verloren gehen

– nicht commitete Daten können festgeschrieben werden

Recovery

[LSN, TA, PageID, Redo, Undo, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Logische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A ,A ,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C ,C ,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B ,C ,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A ,A ,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, Redo, Undo, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Logische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A-=50,A+=50,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C ,C ,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B ,C ,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A ,A ,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, Redo, Undo, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Logische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	a_1 := a_1 - 50		
6	w(A, a_1)		[#3,T_1,P_A ,A-=50,A+=50,#1]
7		c_2 := c_2 + 100	
8		w(C, c_2)	[#4,T_2,P_C ,C+=100,C-=100,#2]
9	r(B, b_1)		
10	b_1 := b_1 + 50		
11	w(B, b_1)		[#5,T_1,P_B ,B ,C ,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		a_2 := a_2 - 100	
15		w(A, a_2)	[#7,T_2,P_A ,A ,A ,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, Redo, Undo, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Logische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A-=50,A+=50,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C+=100,C-=100,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B+=50,C-=50,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A ,A ,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, Redo, Undo, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Logische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A-=50,A+=50,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C+=100,C-=100,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B+=50,C-=50,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A-=100,A+=100,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, Redo, Undo, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Logische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A-=50,A+=50,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C+=100,C-=100,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B+=50,C-=50,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A-=100,A+=100,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, After-Image, Before-Image, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Physische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A= ,A= ,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C= ,C= ,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B= ,B= ,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A= ,A= ,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, After-Image, Before-Image, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Physische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A=950,A=1000,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C= ,C= ,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B= ,B= ,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A= ,A= ,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, After-Image, Before-Image, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Physische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A=950,A=1000,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C=3100,C=3000,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B= ,B= ,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A= ,A= ,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, After-Image, Before-Image, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Physische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A=950,A=1000,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C=3100,C=3000,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B=2050,B=2000,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A= ,A= ,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, After-Image, Before-Image, PrevLSN]

Initialwerte: A = 1000, B = 2000, C = 3000

Physische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A=950,A=1000,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C=3100,C=3000,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B=2050,B=2000,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A=850,A=950,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

[LSN, TA, PageID, After-Image, Before-Image, PrevLSN]

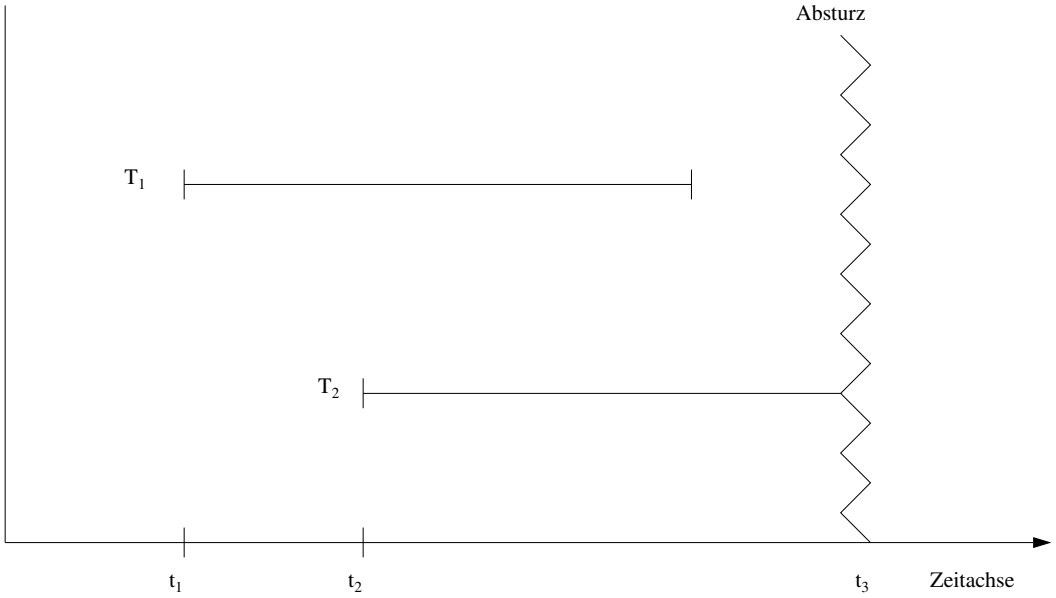
Initialwerte: A = 1000, B = 2000, C = 3000

Physische Protokollierung

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A=950,A=1000,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C=3100,C=3000,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B=2050,B=2000,#3]
12	**commit**		[#6,T_1,**commit**,#5]
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A=850,A=950,#4]
16		**commit**	[#8,T_2,commit,#7]

Recovery

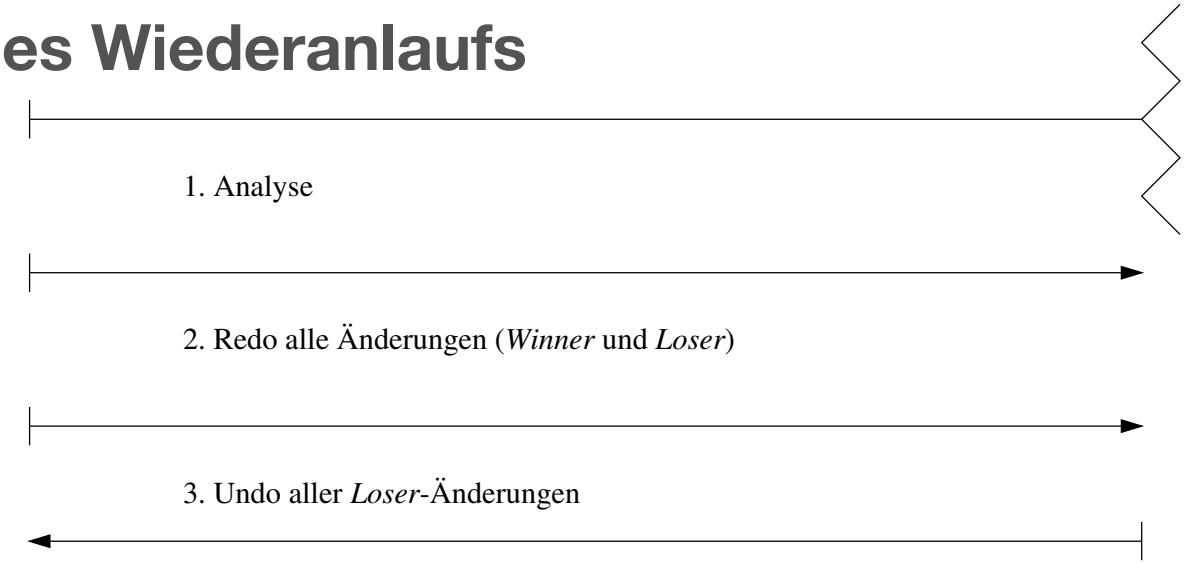
Wiederanlauf nach einem Fehler



- TAs der Art T_1 sind **Winner**: müssen vollständig nachvollzogen werden
- TAs der Art T_2 sind **Loser**: müssen rückgängig gemacht werden

Recovery

Phasen des Wiederanlaufs



1. Analyse:
   Ermitteln der Winner- & Loser-Transaktionen

2. Wiederholung der Historie
   Alle protokollierten Redo-Logs werden in der richtigen Reihenfolge ausgeführt
   => Datenbankzustand während des Absturzes

3. Undo der Losertransaktionen
   Alle uncomitteten TAs werden abgebrochen und ihre Auswirkungen auf die Datenbasis aufgehoben

Recovery

Phasen des Wiederanlaufs

Ermittle die Winner & Loser-Transaktionen:

1. w3[z], r3[x], r2[y], c3, w1[x],  c2, c1

2. w1[x], w1[x], w1[x], c1,  w3[x], c3, w2[y], a2

3. r2[z], w4[z], w2[x], r3[y], w2[y], c2,  r1[y], c4, c3, c1

Recovery

Phasen des Wiederanlaufs

Ermittle die Winner & Loser-Transaktionen:

1. w3[z], r3[x], r2[y], c3, w1[x],  c2, c1

Winner: TA3
Loser: TA1, TA2

2. w1[x], w1[x], w1[x], c1,  w3[x], c3, w2[y], a2

3. r2[z], w4[z], w2[x], r3[y], w2[y], c2,  r1[y], c4, c3, c1

Recovery

Phasen des Wiederanlaufs

Ermittle die Winner & Loser-Transaktionen:

1. w3[z], r3[x], r2[y], c3, w1[x],  c2, c1

Winner: TA3
Loser: TA1, TA2

2. w1[x], w1[x], w1[x], c1,  w3[x], c3, w2[y], a2

Winner: TA1
Loser: TA2, TA3

3. r2[z], w4[z], w2[x], r3[y], w2[y], c2,  r1[y], c4, c3, c1

Recovery

Phasen des Wiederanlaufs

Ermittle die Winner & Loser-Transaktionen:

1. w3[z], r3[x], r2[y], c3, w1[x],  c2, c1

   Winner: TA3
   Loser: TA1, TA2

2. w1[x], w1[x], w1[x], c1,  w3[x], c3, w2[y], a2

   Winner: TA1
   Loser: TA2, TA3

3. r2[z], w4[z], w2[x], r3[y], w2[y], c2,  r1[y], c4, c3, c1

   Winner: TA2
   Loser: TA1, TA3, TA4

Recovery

<LSN, TA, PageID, Redo, PrevLSN, UndoNxtLSN >

Initialwerte: A = 1000, B = 2000, C = 3000

Compensation Log Records

Schritt	T_1	T_2	Log
1	**BOT**		[#1,T_1,**BOT**,0]
2	r(A, a_1)		
3		**BOT**	[#2,T_2,**BOT**,0]
4		r(C, c_2)	
5	$a_1 := a_1 - 50$		
6	w(A, a_1)		[#3,T_1,P_A ,A-=50,A+=50,#1]
7		$c_2 := c_2 + 100$	
8		w(C, c_2)	[#4,T_2,P_C ,C+=100,C-=100,#2]
9	r(B, b_1)		
10	$b_1 := b_1 + 50$		
11	w(B, b_1)		[#5,T_1,P_B ,B+=50,C-=50,#3]
12	**commit**		[#6,T_1,**commit**,#5]
			Absturz
13		r(A, a_2)	
14		$a_2 := a_2 - 100$	
15		w(A, a_2)	[#7,T_2,P_A ,A-=100,A+=100,#4]
16		**commit**	[#8,T_2,commit,#7]

CLR:

<#4′,T_2,P_C,C–=100,#4,#2 >

<#2′,T_2,—,—,#4',0>

Kapitel 11

Mehrbenutzersynchronisation

Mehrbenutzersynchronisation
Formale Definition einer Transaktion

Operationen einer Transaktion TA T_i

- BOT_i Beginn der Transaktion (Begin Of Transaction)
- $\mathbf{r_i(A)}$ Lesen (Read) von Datenobjekt A
- $\mathbf{w_i(A)}$ Schreiben (Write) von Datenobjekt A
- $\mathbf{a_i}$ Abbruch (Abort) der Transaktion
- $\mathbf{c_i}$ Festschreiben (Commit) der Transaktion

Mehrbenutzersynchronisation

Konfliktoperationen

In Konflikt stehende Operationen dürfen nicht parallel ausgeführt werden

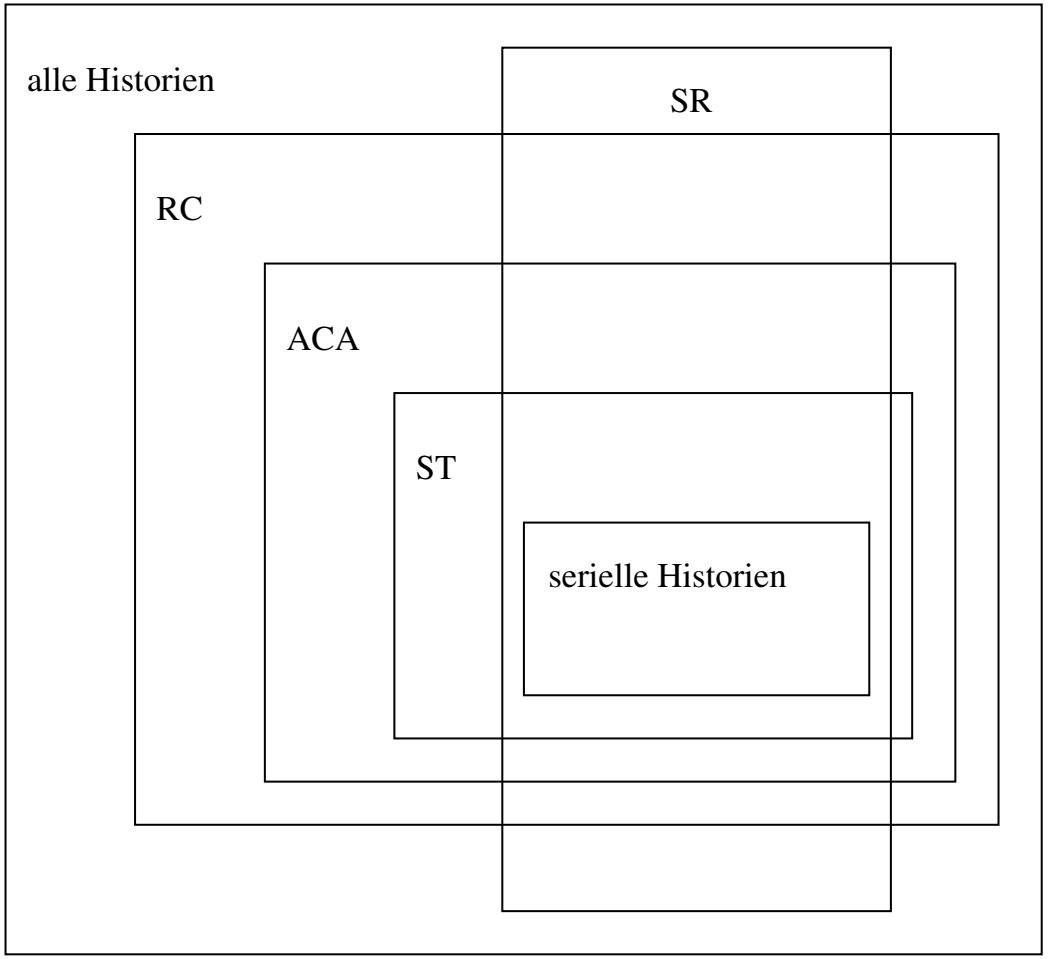
Zwei Operationen stehen in Konflikt, wenn beide auf dem selben Datenobjekt arbeiten wollen und mindestens eine Operation schreibt

T_j \ T_i	$r_i[x]$	$w_i[x]$
$r_j[x]$	✔	✘
$w_j[x]$	✘	✘

✔ Kein Konflikt
✘ Konflikt

Mehrbenutzersynchronisation
Klassifikation von Historien

SR: serialisierbar
RC: rücksetzbar
ACA: vermeidet kaska-
dierendes Rücksetzen
ST: strikt
ST&SR: Seriell



Mehrbenutzersynchronisation

Klassifikation von Historien (Serialisierbar)

$r_2[y], r_1[y], w_2[y], c_2, r_3[x], w_1[x], r_3[y], c_3, c_1$

T_1		$r_1[y]$				$w_1[x]$			c_1
T_2	$r_2[y]$		$w_2[y]$	c_2					
T_3					$r_3[x]$		$r_3[y]$	c_3	

Serialisierbar: Serielle Reihenfolge der Ausführung möglich

Konfliktoperationen

$r_3[x] < w_1[x]$

$r_1[y] < w_2[y]$

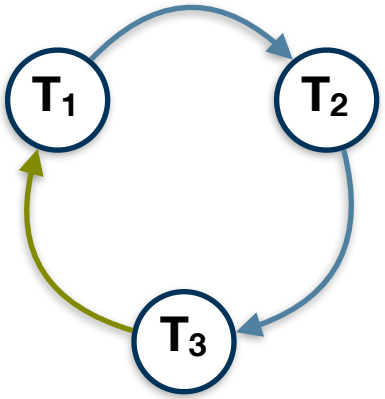
$w_2[y] < r_3[y]$

Auswertungsreihenfolge

3 vor 1

1 vor 2

2 vor 3



TAs zyklisch voneinander abhängig => Nicht Serialisierbar

Mehrbenutzersynchronisation

Klassifikation von Historien (Rücksetzbar)

T_1		$r_1[y]$				$w_1[x]$			c1
T_2	$r_2[y]$		$w_2[y]$	c2					
T_3					$r_3[x]$		$r_3[y]$	c3	

Rücksetzbar: Schreiber von Daten muss vor Leser commiten

Konfliktoperationen

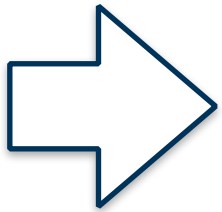
~~$r_3[x] < w_1[x]$~~

~~$r_1[y] < w_2[y]$~~

$w_2[y] < r_3[y]$

Commit-Reihenfolge

$c_2 < c_3 < c_1$



Gewünschte C-Reihenfolge

2 vor 3

Tatsächliche C-Reihenfolge

2 vor 3

Bedingung erfüllt => Rücksetzbar

Mehrbenutzersynchronisation

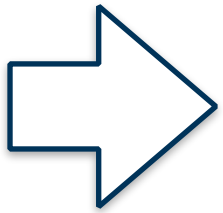
Klassifikation von Historien (ACA)

T_1		$r_1[y]$				$w_1[x]$			c1
T_2	$r_2[y]$		$w_2[y]$	c2					
T_3					$r_3[x]$		$r_3[y]$	c3	

Vermeidet kaskadierendes Rücksetzen:
Schreiber von Daten muss commiten bevor Daten gelesen werden

Konfliktoperationen

~~$r_3[x] < w_1[x]$~~
~~$r_1[y] < w_2[y]$~~
$w_2[y] < r_3[y]$



geforderte Reihenfolge
$w_2[y] < c_2 < r_3[y]$

Geforderte Reihenfolge wird eingehalten => ACA

Mehrbenutzersynchronisation

Klassifikation von Historien (Strikt)

T_1		$r_1[y]$				$w_1[x]$			c1
T_2	$r_2[y]$		$w_2[y]$	c2					
T_3					$r_3[x]$		$r_3[y]$	c3	

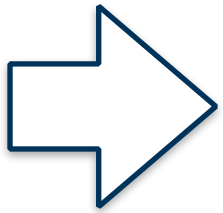
Strikt: Schreiber von Daten muss commiten bevor Daten gelesen oder geschrieben werden

Konfliktoperationen

~~$r_3[x] < w_1[x]$~~

~~$r_1[y] < w_2[y]$~~

$w_2[y] < r_3[y]$



geforderte Reihenfolge

$w_2[y] < c_2 < r_3[y]$

Geforderte Reihenfolge wird eingehalten => Strikt

Mehrbenutzersynchronisation

Online Tool

https://transactions.db.in.tum.de

Kapitel 12

Sicherheitsaspekte

Sicherheitsaspekte
Überblick

RSA

- Welche Schlüssel gibt es? Public and Private Key
- Verschlüsseln, Entschlüsseln, Signieren
- Formeln müssen nicht auswendig gelernt werden

SQL-Injection

- http://db.in.tum.de/~schuele/sql_verzeichnis.html
- SQL Syntax (INSERT, UPDATE, DELETE, DROP)

k-Anonymität

- Angriffsarten und Anfälligkeiten

Kapitel 15

Deduktive Datenbanken

Deduktive Datenbanken
Überblick

- Theorie (Wann ist ein Programm sicher bzw stratifizierbar?)
- **Datalog Programme verstehen und ergänzen**
- **Definition neuer Regeln**
- \ =, not(...), +
- Einfache Regeln zu SQL übersetzen und zurück
- Rekursion
- Domänenkalkül zu Datalog übersetzen
- Keine Aggregationsfunktionen!

http://datalog.db.in.tum.de/

Deduktive Datenbanken
Regeln

Basisrelationen:

vorlesungen(VorlNr, Titel, SWS, PersNr)

professoren(PersNr, Name, Rang, Raum)

Regelerzeugung und Join:

sokLV(T,S) :-vorlesungen(_,T,S,P), professoren(P, „Sokrates“,_,_), S>2.

Deduktive Datenbanken
Rekursion

Datenbasis: direkt(Start, Ziel, Linie)

Ziel: indirekt(Start, Ziel, Stops)

1. **Basisfall** => Fülle die Relation mit Anfangswerten

```
indirekt( Start, Ziel, Stops ) :- direkt( Start, Ziel, _), Stops = 0.
```

2. **Rekursion** => Nutze die Relation selbst und erweitere sie

```
indirekt( Start, Ziel, StopsNeu ) :-
        indirekt( Start, Station, Stops ),
        direkt( Station, Ziel, _),
        StopsNeu = Stops + 1.
```

Kapitel 16

Verteilte Datenbanksysteme

Verteilte Datenbanksysteme
Überblick

horizontale und vertikale Fragmentierung

- Korrektheit
- Rekonstruktion

	Vertikal	Horizontal
Fragmentieren	**π** Projektion	**σ** Selektion
Vereinigen	⋈ Join	∪ Vereinigung

Quorum Consensus

- Lesequorum $Q_r(A)$, Schreibquorum $Q_w(A)$
- $2 * Q_w(A) > W(A)$
- $Q_r(A) + Q_w(A) > W(A)$

Chord Netzwerk

- Finden von Schlüsseln
- Fingertabellen ausfüllen

Verteilte Datenbanksysteme

Bloom-Filter

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V
0
1
2
3
4
5
6
7
8
9

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

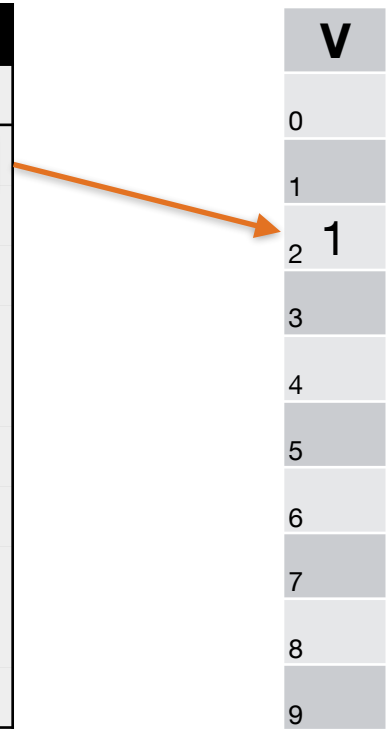
$h(x) = x \bmod 10$

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5



S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

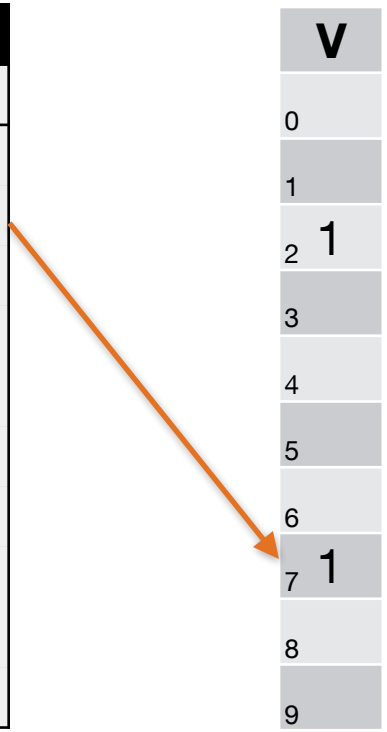
$h(2) = 2 \bmod 10 = 2$

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5



S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

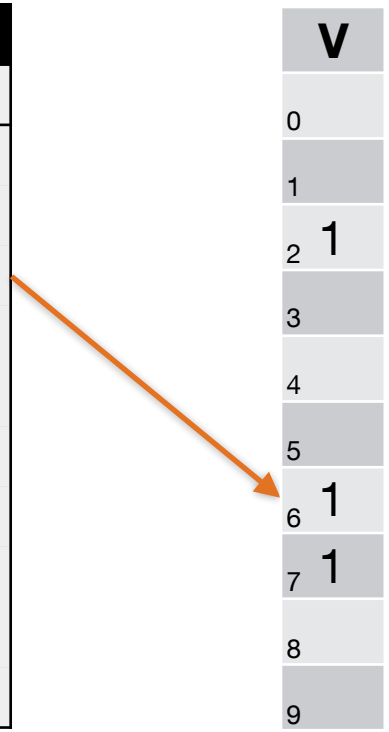
$h(7) = 7 \bmod 10 = 7$

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5



S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

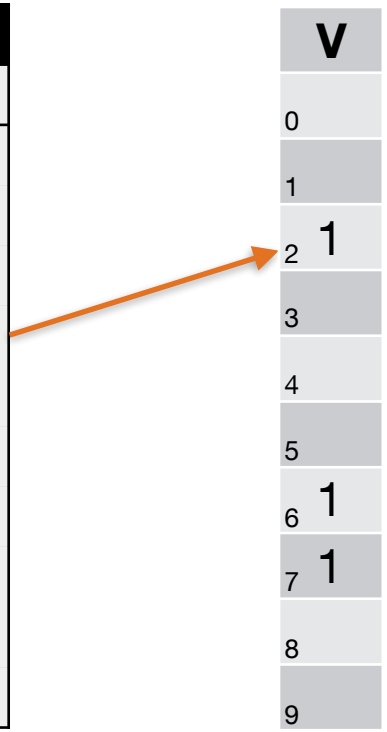
$h(6) = 6 \bmod 10 = 6$

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5



S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

$h(2) = 2 \bmod 10 = 2$

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	
1	
2	1
3	
4	
5	
6	1
7	1
8	1
9	

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

$h(8) = 8 \bmod 10 = 8$

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	
1	1
2	1
3	
4	
5	
6	1
7	1
8	1
9	

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

h(1) = 1 mod 10 = 1

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	
1	1
2	1
3	
4	
5	
6	1
7	1
8	1
9	

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

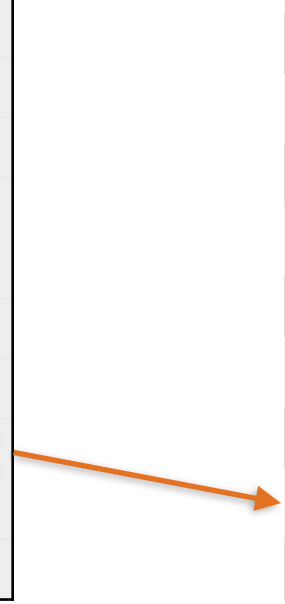
$h(16) = 16 \bmod 10 = 16$

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5



V	
0	
1	1
2	1
3	
4	
5	
6	1
7	1
8	1
9	

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

$h(8) = 8 \bmod 10 = 8$

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	
1	1
2	1
3	
4	
5	
6	1
7	1
8	1
9	

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

h(7) = 7 mod 10 = 7

Verteilte Datenbanksysteme

Bloom-Filter

1. Tabelle R mit h(x) auf V mappen:

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	
1	1
2	1
3	
4	
5	1
6	1
7	1
8	1
9	

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

h(5) = 5 mod 10 = 5

Verteilte Datenbanksysteme

Bloom-Filter

2. Felder in V ohne hash-Treffer mit 0 füllen

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

	V
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

$h(x) = x \bmod 10$

Verteilte Datenbanksysteme

Bloom-Filter

3. Bitvektor V an S schicken

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

$h(x) = x \bmod 10$

Verteilte Datenbanksysteme

Bloom-Filter

4. S überprüft mit h(x) den Bitvektor V

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

	V
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S	
Raum	Gebäude
1	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

$h(x) = x \bmod 10$

Verteilte Datenbanksysteme

Bloom-Filter

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

4. S überprüft mit h(x) den Bitvektor V

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S	
Raum	Gebäude
1 ✔	IMETUM
2	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

h(1) = 1 mod 10 = 1

Verteilte Datenbanksysteme

Bloom-Filter

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

4. S überprüft mit h(x) den Bitvektor V

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

	V
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S	
Raum	Gebäude
1 ✔	IMETUM
2 ✔	MI Büro
4	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

h(2) = 2 mod 10 = 2

Verteilte Datenbanksysteme

Bloom-Filter

4. S überprüft mit h(x) den Bitvektor V

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

	V
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S	
Raum	Gebäude
1 ✔	IMETUM
2 ✔	MI Büro
4 ✘	Physik
6	MW
7	MI Raum
8	ERI
9	MI Bib
10	Physik
11	Chemie

h(4) = 4 mod 10 = 4

Verteilte Datenbanksysteme

Bloom-Filter

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

4. S überprüft mit h(x) den Bitvektor V

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S		
Raum		Gebäude
1	✔	IMETUM
2	✔	MI Büro
4	✘	Physik
6	✔	MW
7		MI Raum
8		ERI
9		MI Bib
10		Physik
11		Chemie

h(6) = 6 mod 10 = 6

Verteilte Datenbanksysteme

Bloom-Filter

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

4. S überprüft mit h(x) den Bitvektor V

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S		
Raum		Gebäude
1	✔	IMETUM
2	✔	MI Büro
4	✘	Physik
6	✔	MW
7	✔	MI Raum
8		ERI
9		MI Bib
10		Physik
11		Chemie

h(7) = 7 mod 10 = 7

Verteilte Datenbanksysteme

Bloom-Filter

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

4. S überprüft mit h(x) den Bitvektor V

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S		
Raum		Gebäude
1	✔	IMETUM
2	✔	MI Büro
4	✘	Physik
6	✔	MW
7	✔	MI Raum
8	✔	ERI
9		MI Bib
10		Physik
11		Chemie

h(8) = 8 mod 10 = 8

Verteilte Datenbanksysteme

Bloom-Filter

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

4. S überprüft mit h(x) den Bitvektor V

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

V	
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S		
Raum		Gebäude
1	✔	IMETUM
2	✔	MI Büro
4	✘	Physik
6	✔	MW
7	✔	MI Raum
8	✔	ERI
9	✘	MI Bib
10		Physik
11		Chemie

h(9) = 9 mod 10 = 9

Verteilte Datenbanksysteme

Bloom-Filter

4. S überprüft mit h(x) den Bitvektor V

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

	V
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S		
Raum		Gebäude
1	✔	IMETUM
2	✔	MI Büro
4	✘	Physik
6	✔	MW
7	✔	MI Raum
8	✔	ERI
9	✘	MI Bib
10	✘	Physik
11		Chemie

h(10) = 10 mod 10 = 0

Verteilte Datenbanksysteme

Bloom-Filter

False positive

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

4. S überprüft mit h(x) den Bitvektor V

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

	V
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S		
Raum		Gebäude
1	✔	IMETUM
2	✔	MI Büro
4	✘	Physik
6	✔	MW
7	✔	MI Raum
8	✔	ERI
9	✘	MI Bib
10	✘	Physik
11	✔	Chemie

h(11) = 11 mod 10 = 1

Verteilte Datenbanksysteme

Bloom-Filter

5. Übermitteln der Treffer zur Station R

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

	v
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

S		
Raum		Gebäude
1	✔	IMETUM
2	✔	MI Büro
4	✘	Physik
6	✔	MW
7	✔	MI Raum
8	✔	ERI
9	✘	MI Bib
10	✘	Physik
11	✔	Chemie

$h(x) = x \bmod 10$

Verteilte Datenbanksysteme

Bloom-Filter

False positives werden übermittelt und von R beim Join verworfen.

R	
Pers	Raum
Max	2
Magda	7
Tom	6
Alex	2
Julius	8
Kathi	1
Anna	16
Gregor	8
Thuy	7
Domi	5

v	
0	0
1	1
2	1
3	0
4	0
5	1
6	1
7	1
8	1
9	0

False positive Rate 1/6

✔ Tupel wird zur Station mit R geschickt
✘ Tupel wird nicht übermittelt

S		
Raum		Gebäude
1	✔	IMETUM
2	✔	MI Büro
4	✘	Physik
6	✔	MW
7	✔	MI Raum
8	✔	ERI
9	✘	MI Bib
10	✘	Physik
11	✔	Chemie

$h(x) = x \bmod 10$

Kapitel 17

Betriebliche Anwendungen

Betriebliche Anwendungen
Überblick

Apriori

- Frequent Itemsets bestimmen
- Konfidenz von Assoziationsregeln ableiten

Skyline

```
select MatrNr from Klausur k skyline of k.Vorbereitungszeit min, k.Note min

select MatrNr from Klausur k where not exists (
  select * from klausur dom where
    dom.Vorbereitungszeit <= k.Vorbereitungszeit and dom.Note <= k.Note and
    (dom.Vorbereitungszeit < k.Vorbereitungszeit or dom.Note < k.Note)
)
```

Threshold/NRA

– Ausführen und Verständnis der Algorithmen

– Unterschiede zwischen den Algorithmen

Betriebliche Anwendungen

Online Transaction Processing

- realisiert „operationale“ Tagesgeschäfte („mission-critical“)

- Charakterisierung
  - Hoher Parallelitätsgrad
  - Viele kurze TA (Tausende pro Sekunde)
  - begrenzte Datenmenge pro TA
  - operieren auf jüngstem, aktuell gültigem Zustand der DB
  - Hohe Verfügbarkeit muss gewährleistet sein

- Normalisierte Relationen (möglichst geringe Update-Kosten)

- Wenige Indexe (Fortschreibungskosten)

Online Analytical Processing

- zur strategischen Unternehmensplanung

- große Datenmengen

- greift häufig auch auf historische Daten zu

➡ gewährt Rückschlüsse auf Entwicklungen

➡ Bestandteil von Decision-Support-Systeme/Management-Informationssysteme

SQL
Window Functions

- Sehr vielseitig und geeignet für
  - Zeitliche Analysen
  - Rangbasierte Anfragen
  - Top-K
  - Gleitender Durchschnitt
  - Kumulative/Wachsende Summe

- Window Functions werden nach **GROUP BY** und vor **ORDER BY** ausgewertet

Betriebliche Anwendungen

Window Funktionen

erdb		
name	**übung**	**punkte**
Thuy	1	1
Anna	1	1
Domi	1	2
Tobi	2	1
Thuy	2	1
Thuy	3	2
Anna	3	1
Domi	3	1
Thuy	4	3
Tobi	5	2
Domi	5	1
Anna	5	1
Tobi	6	2
Thuy	6	1

```
SELECT name, übung, (100.0*punkte)/
    sum(punkte)
    over (partition by übung) as prozent
FROM erdb
```

Betriebliche Anwendungen

Window Funktionen

erdb		
name	**übung**	**punkte**
Thuy	1	1
Anna	1	1
Domi	1	2
Tobi	2	1
Thuy	2	1
Thuy	3	2
Anna	3	1
Domi	3	1
Thuy	4	3
Tobi	5	2
Domi	5	1
Anna	5	1
Tobi	6	2
Thuy	6	1

```
SELECT name, übung, (100.0*punkte)/
       sum(punkte)
       over (partition by übung) as prozent
FROM erdb
```

Ergebnis		
name	**übung**	**prozent**
Thuy	1	25.0
Anna	1	25.0
Domi	1	50.0
Tobi	2	50.0
Thuy	2	50.0
Thuy	3	50.0
Anna	3	25.0
Domi	3	25.0
Thuy	4	100.0
Tobi	5	50.0
Domi	5	25.0
Anna	5	25.0
Tobi	6	66.7
Thuy	6	33.3

Betriebliche Anwendungen

Window Funktionen

erdb		
name	**übung**	**punkte**
Anna	1	1
Anna	3	1
Anna	5	1
Domi	1	2
Domi	3	1
Domi	5	1
Thuy	1	1
Thuy	2	1
Thuy	3	2
Thuy	4	3
Thuy	6	1
Tobi	2	1
Tobi	5	2
Tobi	6	2

SELECT name, übung, sum(punkte)
over (partition by name
order by übung)
FROM erdb

Ergebnis		
name	**übung**	**sum**
Anna	1	1
Anna	3	2
Anna	5	3
Thuy	1	1
Thuy	2	2
Thuy	3	4
Thuy	4	7
Thuy	6	8
Domi	1	2
Domi	3	3
Domi	5	4
Tobi	2	1
Tobi	5	3
Tobi	6	5

Betriebliche Anwendungen

Window Funktionen

erdb		
name	**übung**	**punkte**
Anna	1	1
Anna	3	1
Anna	5	1
Domi	1	2
Domi	3	1
Domi	5	1
Thuy	1	1
Thuy	2	1
Thuy	3	2
Thuy	4	3
Thuy	6	1
Tobi	2	1
Tobi	5	2
Tobi	6	2

```
SELECT name, übung, sum(punkte)
    over ( partition by name
      order by übung
      range between unbounded
      preceding and current row)
FROM erdb
```

Ergebnis		
name	**übung**	**sum**
Anna	1	1
Anna	3	2
Anna	5	3
Thuv	1	1
Thuv	2	2
Thuv	3	4
Thuv	4	7
Thuv	6	8
Domi	1	2
Domi	3	3
Domi	5	4
Tobi	2	1
Tobi	5	3
Tobi	6	5

Betriebliche Anwendungen

Window Funktionen

erdb		
name	**übung**	**punkte**
Anna	1	1
Anna	3	1
Anna	5	1
Domi	1	2
Domi	3	1
Domi	5	1
Thuy	1	1
Thuy	2	1
Thuy	3	2
Thuy	4	3
Thuy	6	1
Tobi	2	1
Tobi	5	2
Tobi	6	2

```
SELECT name, übung, sum(punkte)
    over (partition by name
        order by übung
        range between 1 preceding
            and 1 following)
FROM erdb
```

Ergebnis		
name	**übung**	**sum**
Anna	1	1
Anna	3	1
Anna	5	1
Thuv	1	2
Thuv	2	4
Thuv	3	6
Thuv	4	5
Thuv	6	1
Domi	1	2
Domi	3	1
Domi	5	1
Tobi	2	1
Tobi	5	4
Tobi	6	4

Betriebliche Anwendungen

Window Funktionen

erdb		
name	**übung**	**punkte**
Anna	1	1
Anna	3	1
Anna	5	1
Domi	1	2
Domi	3	1
Domi	5	1
Thuy	1	1
Thuy	2	1
Thuy	3	2
Thuy	4	3
Thuy	6	1
Tobi	2	1
Tobi	5	2
Tobi	6	2

```
SELECT name, übung, sum(punkte)
    over (partition by name
      order by übung
      rows between 1 preceding
          and 1 following)
FROM erdb
```

Ergebnis		
name	**übung**	**sum**
Anna	1	2
Anna	3	3
Anna	5	2
Thuy	1	2
Thuy	2	4
Thuy	3	6
Thuy	4	6
Thuy	6	4
Domi	1	3
Domi	3	4
Domi	5	2
Tobi	2	3
Tobi	5	5
Tobi	6	4

Betriebliche Anwendungen

Window Funktionen

erdb		
name	**übung**	**punkte**
Anna	1	1
Anna	3	1
Anna	5	1
Domi	1	2
Domi	3	1
Domi	5	1
Thuy	1	1
Thuy	2	1
Thuy	3	2
Thuy	4	3
Thuy	6	1
Tobi	2	1
Tobi	5	2
Tobi	6	2

```
SELECT name, sum(punkte) as gesamt
FROM erdb
GROUP BY name
ORDER BY gesamt desc
```

erdb nach group	
name	**gesamt**
Thuy	8
Tobi	5
Domi	4
Anna	3

Betriebliche Anwendungen

Window Funktionen

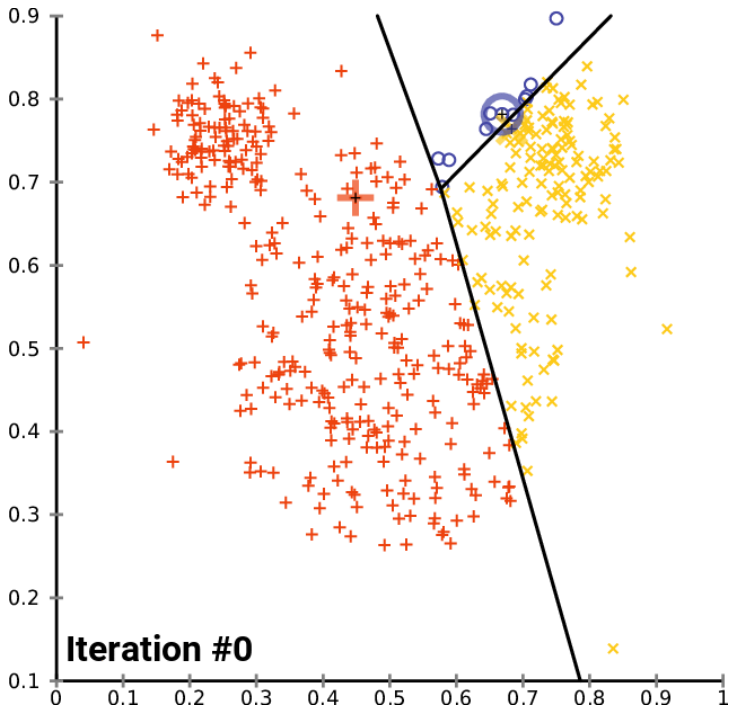
erdb nach group	
name	**gesamt**
Thuy	8
Tobi	5
Domi	4
Anna	3

```
SELECT name, gesamt,
    rank() over (order by gesamt desc)
FROM (
    SELECT name, sum(punkte) as
        gesamt
    FROM erdb
    GROUP BY name desc)
```

Ergebnis		
name	**gesamt**	**rank**
Thuy	8	1
Tobi	5	2
Domi	4	3
Anna	3	4

Clustering
K-Means

- Teilen von Datenpunkten in konvexe Cluster
- Minimiere die Summe der Abstände zu den Cluster Mittelpunkten
- Anzahl der Cluster von Nutzer bestimmt
- Auswahl der Startpunkte bietet viel Potenzial für Optimierung



https://commons.wikimedia.org/wiki/File:K-means_convergence.gif

Kapitel 18

Hauptspeicher-Datenbanken

Betriebliche Anwendungen
Überblick

Adaptive Radix Tree

- Finden von Schlüsseln
- Einfügen von Schlüsseln
- Verschiedene Knoten Typen

MVCC mit Precision Locking

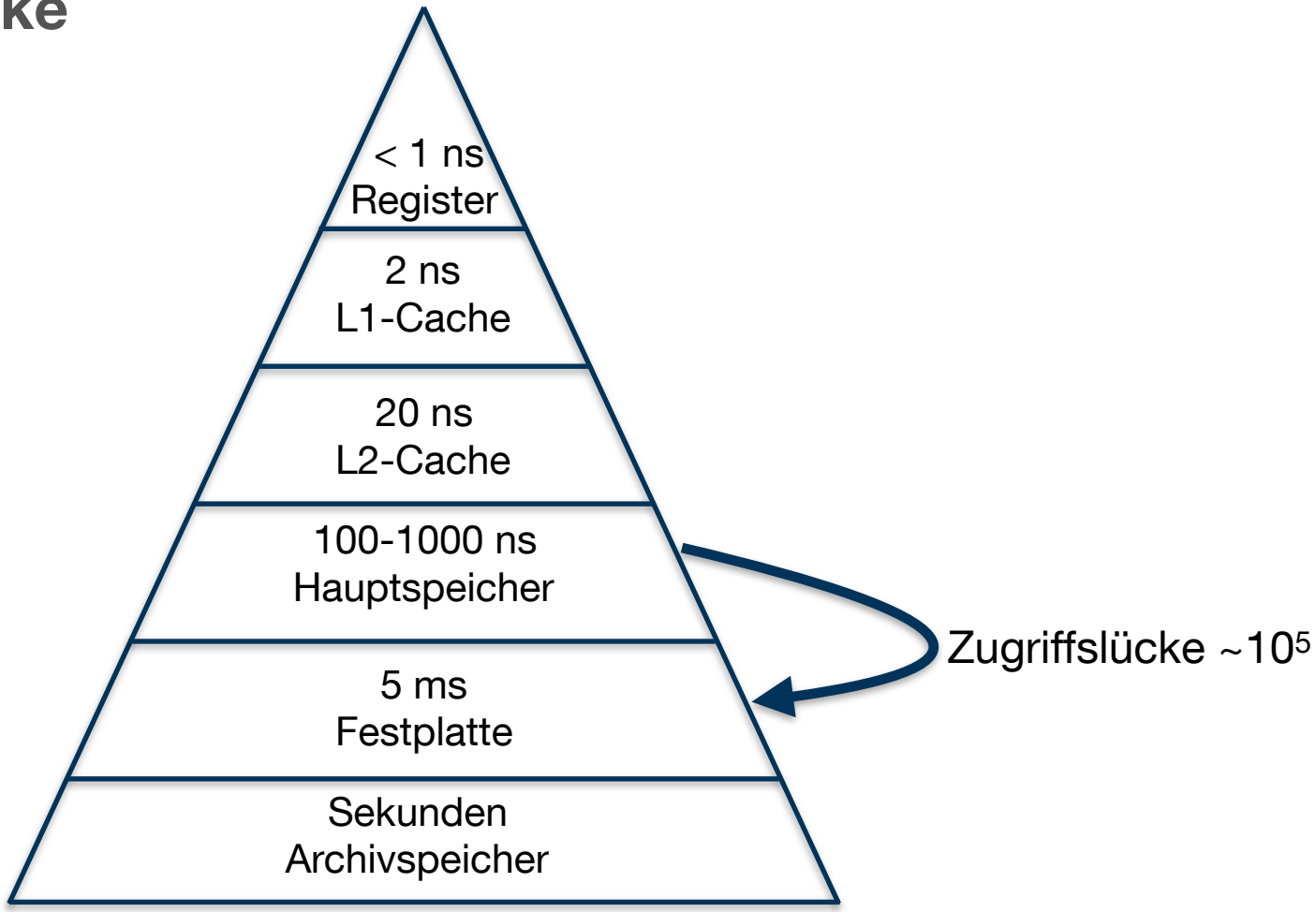
– Lesende Anfragen sind **immer erlaubt**: kein Precision Locking

– Falls schreibende Anfrage: **Überlappender Prädikatbereich**?

– Falls ja, dann **BOT** und **commit-Reihenfolge** beachten

Hauptspeicher-Datenbanken

Zugriffslücke



Hauptspeicher-Datenbanken

Row- vs Column-Store

Column Store

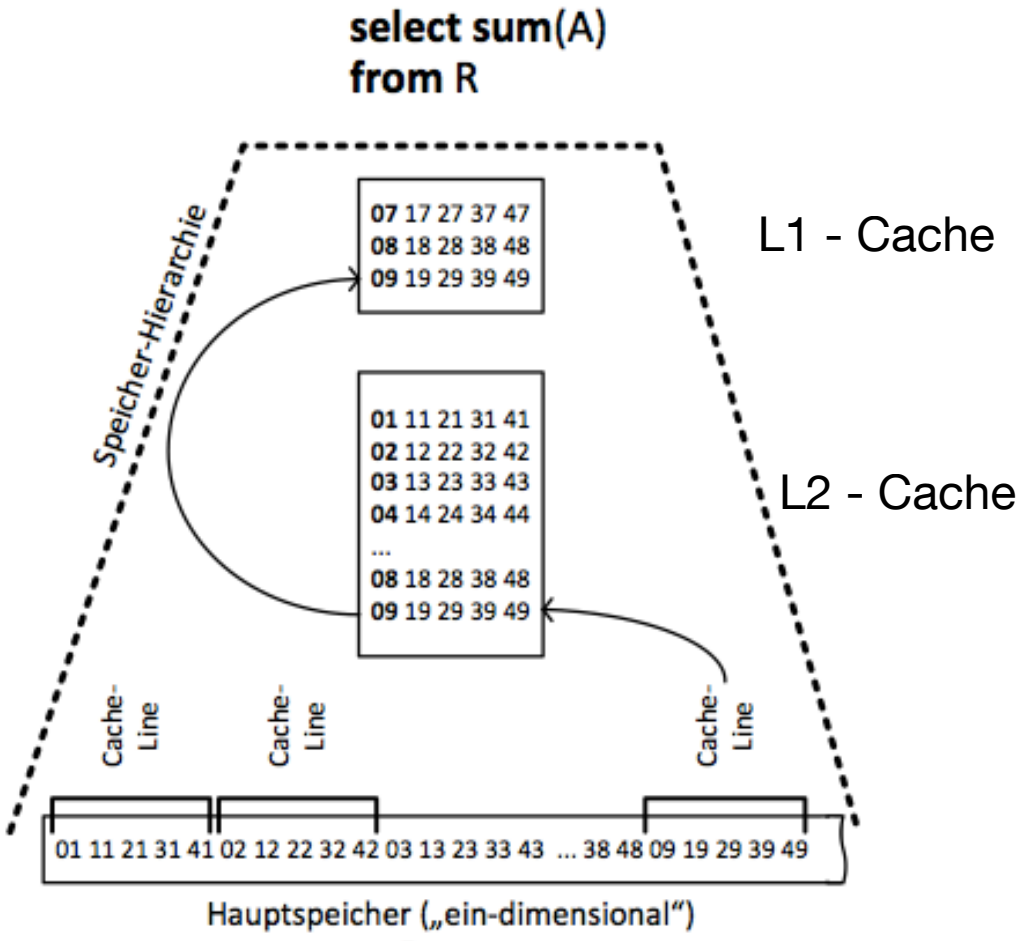
Name	MatrNr	Semester	Fach	Nebenfach
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	362101	10	Info	Mathe

Row Store

Name	MatrNr	Semester	Fach	Nebenfach
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	362101	10	Info	Mathe

Hauptspeicher-Datenbanken

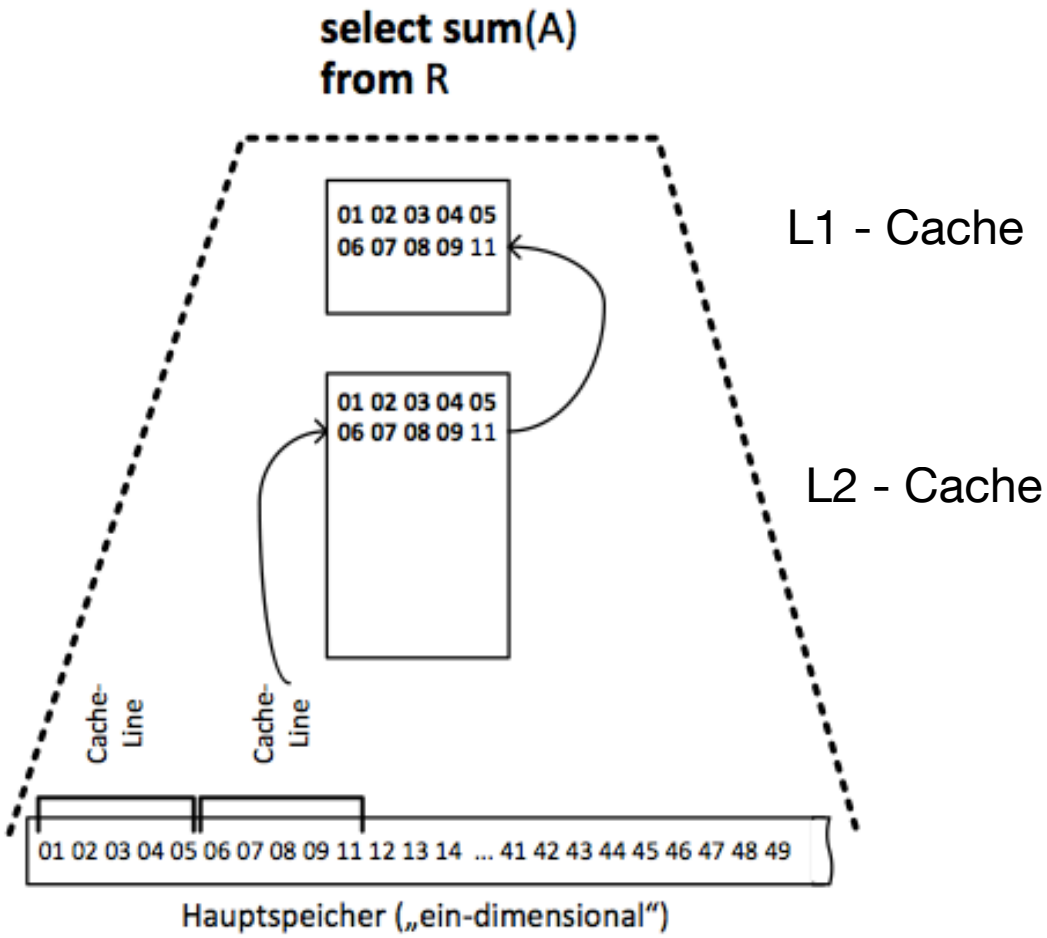
Row-Store



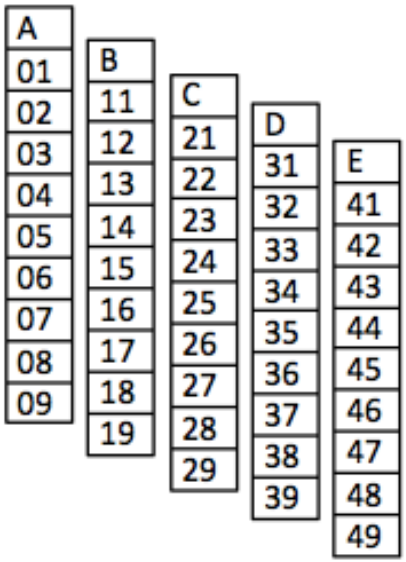
Speicherstruktur

A	B	C	D	E
01	11	21	31	41
02	12	22	32	42
03	13	23	33	43
04	14	24	34	44
05	15	25	35	45
06	16	26	36	46
07	17	27	37	47
08	18	28	38	48
09	19	29	39	49

Hauptspeicher-Datenbanken
Column-Store



Speicherstruktur



Hauptspeicher-Datenbanken
Row vs Column Store

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung. Für die MatrNr existiert ein Index.

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…				

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Hauptspeicher-Datenbanken
Row vs Column Store

```
select *
from Studenten;
```

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…				

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Hauptspeicher-Datenbanken
Row vs Column Store

select *

from Studenten;

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
...				

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

RowStore:

1 Tupel: 32B + 3B + 1B + 4B +16B = 56B

$$\#\text{Cachelines} = \lceil |S| * (56\text{Byte}/64\text{Byte}) \rceil$$
$$= \lceil |S| * (7/8) \rceil$$

Hauptspeicher-Datenbanken
Row vs Column Store

```
select *
from Studenten;
```

ColumnStore:

#Cachelines = ⌈|S|*(32B/64B)⌉ + ⌈|S|*(3B/64B)⌉ + ⌈|S|*(1B/64B)⌉ + ⌈|S|*(4B/64B)⌉ + ⌈|S|*(16B/64B)⌉

= ⌈|S|*(32B+3B+1B+4B+16B)/64B⌉

= ⌈|S|*56B/64B⌉

= ⌈|S|*7/8⌉

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Hauptspeicher-Datenbanken

Row vs Column Store

select Name, MatrNr

from Studenten

where Semester = 10;

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…				

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.

Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Hauptspeicher-Datenbanken
Row vs Column Store

select Name, MatrNr

from Studenten

where Semester = 10;

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
...				

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

RowStore:

$$\#\text{Cachelines} = \lceil |S| * (56\text{Byte}/64\text{Byte}) \rceil$$
$$= \lceil |S| * (7/8) \rceil$$
$$= \lceil |S| * 0{,}875 \rceil$$

Hauptspeicher-Datenbanken

Row vs Column Store

Schätzung der Selektivität von 1/10 ist unrealistisch, insbesondere die Folge das nur 1/10 der CLs gelesen werden. Erfüllt nur den Zweck eines Beispiels.

```
select Name, MatrNr
from Studenten
where Semester = 10;
```

ColumnStore:

$$\#\text{Cachelines} = \lceil |S| * 1B/64B \rceil + \lceil |S| * 32B/64B * 1/10 \rceil + \lceil |S| * 3B/64B * 1/10 \rceil$$

$$= \lceil |S| * (1B/64B + 32B/640B + 3B/640B) \rceil$$

$$= \lceil |S| * (10B + 32B + 3B)/640B \rceil$$

$$= \lceil |S| * 45/640 \rceil$$

$$= \lceil |S| * 0{,}070 \rceil$$

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Hauptspeicher-Datenbanken

Row vs Column Store

```
select Name, MatrNr
from Studenten
where MatrNr = %;
```

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…				

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Hauptspeicher-Datenbanken
Row vs Column Store

```
select Name, MatrNr
from Studenten
where MatrNr = %;
```

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
...				

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

RowStore:

#Cachelines = ⌈56B/64B⌉ = 1

Hier wird der Index von MatrNr genutzt. Deshalb muss nur das Tupel mit der gesuchten MatrNr geladen werden. Dieser umfasst 1 Cacheline.

Hauptspeicher-Datenbanken
Row vs Column Store

select Name, MatrNr
from Studenten
where MatrNr = %;

ColumnStore:

#Cachelines = ⌈32B/64B⌉ + ⌈3B/64B⌉ = 2

Hier wird ebenfalls wieder der Index von MatrNr genutzt, sodass nur der Namen und die MatrNr des Tupels mit der gesuchten MatrNr aus den jeweiligen Tabellengeladen wird.

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Hauptspeicher-Datenbanken
Row vs Column Store

Insert into Studenten VALUES(…);

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…				

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Hauptspeicher-Datenbanken
Row vs Column Store

Insert into Studenten VALUES(…);

Row Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…				

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

RowStore:

#Cachelines = ⌈56B/64B⌉ = 1

Hauptspeicher-Datenbanken
Row vs Column Store

Insert into Studenten VALUES(…);

ColumnStore:

#Cachelines = ⌈32B/64B⌉ + ⌈3B/64B⌉ +

⌈1B/64B⌉ + ⌈4B/64B⌉ + ⌈16B/64B⌉ = 5

Da jedes Attribut muss einzeln in die jeweilige Tabelle eingefügt werden.

Die Anzahl der Tupel in der Relation Studenten ist nicht bekannt, wir verwenden |S| als Abschätzung.
Für die MatrNr existiert ein Index. 1 B = 1 Byte (8Bit)

Column Store

Name (32Byte)	MatrNr (3Byte)	Semester (1 Byte)	Fach (4Byte)	Nebenfach (16 Byte)
Alex	362148	6	Info	Medizin
Max	362139	6	Info	Physik
David	361299	10	Info	MaschBau
Johannes	362033	8	Info	Mathe
Andre	262101	10	Info	Mathe
…	…	…	…	…

Fragen

FAQs

Notenbonus gilt für Haupt- und Wiederholungsklausur
Notenbonus gilt nicht für nächstes Jahr (ERDB 2020)

Taschenrechner ist nicht erlaubt!
Beide Klausuren werden gleich schwer, aber decken womöglich andere Bereiche ab.