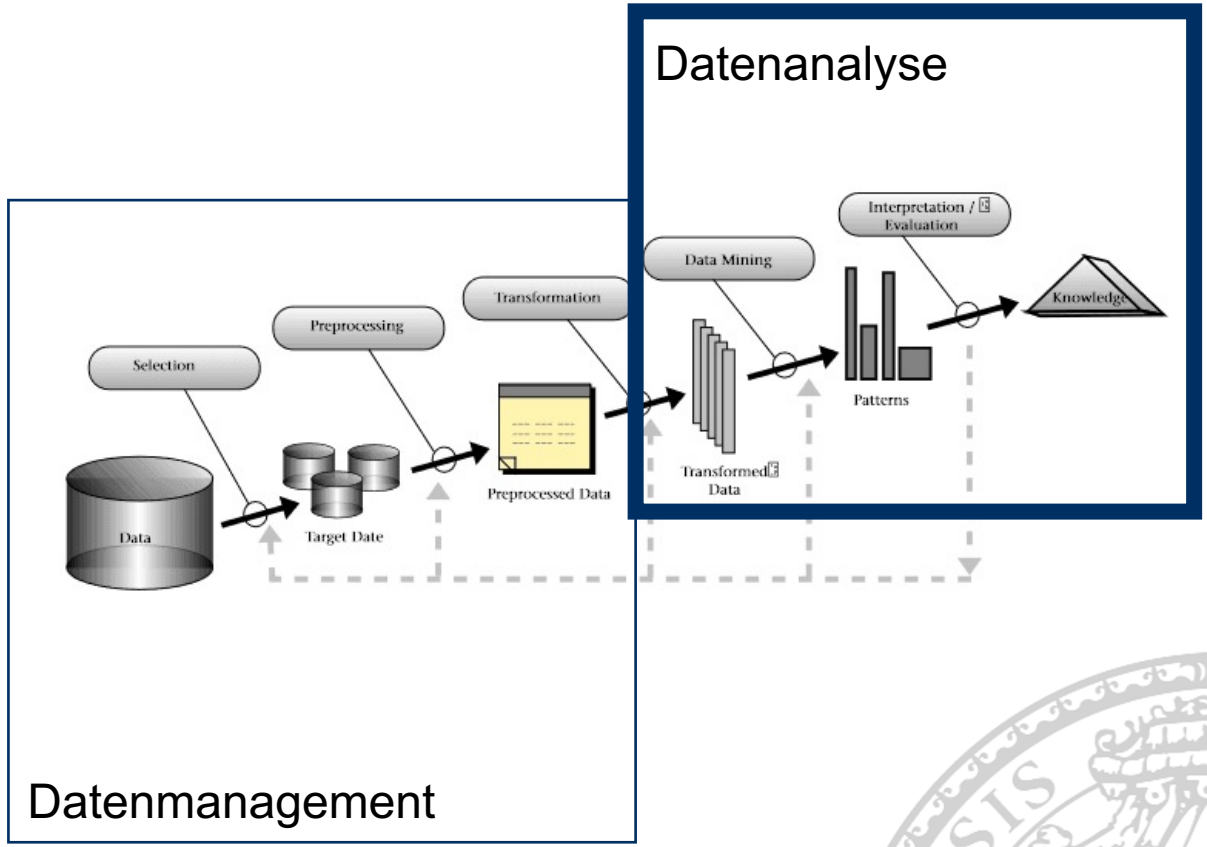
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Datenmanagement & -analyse

DB-Transaktionen

Zwei Schritte vor, einer zurück

- Konzepte klassischer Datenbankarchitekturen
- Datenmodellierung und Normalisierung
- Einführung in eine Anfragesprache
- **Nutzung von Datenbanken**
- Hypothesengetriebene und modellbildende Datenanalyse
- Datenanalyseprozesse und deren Vergleich
- Überwachte und unüberwachte Lernverfahren
- Konzeption und Umsetzung (komplexer) Datenanalysen

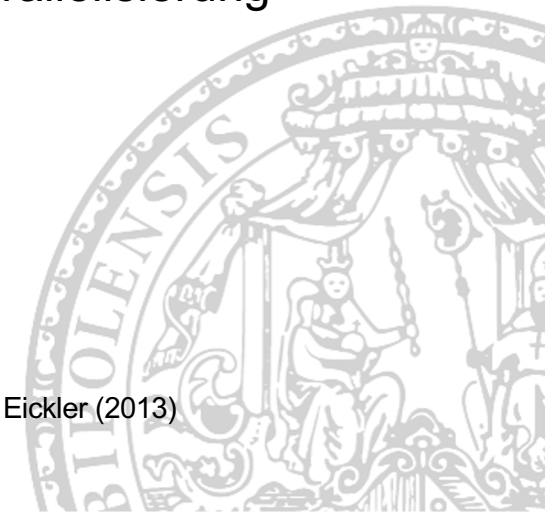


Transaktionen und ACID

- Wozu Transaktionen?
  - Häufig wollen **mehrere Nutzer/Anwendungen parallel** auf DB zugreifen (z.B. ERP-System, Web Shop, …)
  - Kann zu **Konkurrenzsituationen** führen

  →**Gefahr für Konsistenz der Datenbank**

- Was ist eine Transaktion?
  - **Zusammenfassung von Operationen** auf DB als **eine Arbeitseinheit**
  - Überführt DB von konsistenten in **konsistenten Zustand**
  - Zwei grundlegende Anforderungen
    - **Recovery**: Behebung von (unvermeidbaren) Fehlersituationen
    - **Synchronisation**: Parallelisierung von Transaktionen



Kemper, Eickler (2013)

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Agenda

1. **ACID-Eigenschaften**
2. **Typische Probleme**
3. **Lösungsansätze**

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Eigenschaften einer Transaktionen: ACID

- **Atomicity** (Abgeschlossenheit)
- **Consistency** (Konsistenzerhaltung)
- **Isolation** (Isoliertheit)
- **Durability** (Dauerhaftigkeit)

Abgeschlossenheit und Konsistenzerhaltung

- Atomicity
  - Transaktion ist **unteilbar**
  - D.h. eine Transaktion wird entweder **ganz** oder **gar nicht** ausgeführt
  - erfolgreiche Transaktionen werden mit `commit` persistiert
  - Andernfalls gelten sie als abgebrochen (`abort`)
  - Änderungen der Transaktion an der Datenbank müssen rückgängig gemacht werden (`rollback`)
- Consistency
  - Transaktion versetzt Datenbank in einen **konsistenten** Zustand, sofern sie vor der Transaktion konsistent war
  - Inkonsistente Zustände werden **komplett** zurückgesetzt
  - Zwischenstände dürfen inkonsistent sein
  - Konsistent: Fachliche Korrektheit (**Integrität**)
  - Beispiel
    - Doppische Buchführung: Keine Buchung ohne Gegenbuchung
    - Redundante Daten müssen konsistent gehalten werden

Kemper, Eickler (2013)

Isoliertheit und Dauerhaftigkeit

- Isolation
  - Nicht vollständig ausgeführte Transaktionen haben **keinen Einfluss auf parallele Transaktionen und sind nicht sichtbar**
  - Jede Transaktion wird behandelt, als wäre sie die einzige auf der Datenbank
  - Ergebnisse werden erst nach `commit` sichtbar
- Durability
  - Effekt einer Transaktion **bleibt wirksam**, sobald sie durch `commit` bestätigt wurde
  - Bezieht sich weniger auf fachliche Integrität sondern auf **Sicherung der Daten** vor
    - Systemausfällen
    - Hardwarefehlern
    - weiteren externen Risiken

Kemper, Eickler (2013)

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Agenda

Notation für Transaktionen

- Variablen
  - Datenbankvariablen: Großbuchstaben, z.B. **`A`**, **`B`**
  - Transaktionsvariablen: Kleinbuchstaben entsprechend Transaktionsnr. und DB-Variable: z.B. **`a1`**, **`b2`**
- Operationen
  - Lesen von DB-Variablen: **`read`**, z.B. **`read a1←A`**
  - Schreiben von DB-Variablen: **`write`**, z.B. **`write b2→B`**
  - Bestätigen, Zurücksetzen von Transaktionen: **`commit`**, **`rollback`**

Notation

Tabelle	
DBVar	Wert
A	50
B	20

- Datenbankvariablen: z.B. **A**, **B**
- Transaktionsvariablen: z.B. **a1**, **b2**
- Lesen : **read**, z. B. **read a1←A**
- Schreiben : **write**, z. B. **write b2→B**
- Bestätigen, Zurücksetzen : **commit**, **rollback**

`A = 50, B = 20`

Transaktion 1	Transaktion 2

Spalte für Variablenwerte

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Agenda

Lost Update – Verlorene Aktualisierung (write-write-conflict)

- Beschreibung:
  - Transaktion 1 schreibt einen Wert
  - Transaktion 2 überschreibt diesen Wert ohne die Aktualisierung zu berücksichtigen
  - Beide **Transaktionen lesen konsistente Datenbankzustände**
- Problem:
  - Das **Update** durch Transaktion 1 **geht verloren**
  - Datenbank wird möglicherweise **inkonsistent**
  - Schwierig zu entdecken, da Datenbank nicht zwingend inkonsistent

Lost Update

Ziel: Beide Transaktionen wollen A um 10 erhöhen

A = 20

Transaktion 1	Transaktion 2	
read a1←A		a1 = 20
	read a2←A	a2 = 20
a1 = a1 + 10		a1 = 30
write a1→A		A = 30
commit		
	a2 = a2 + 10	a2 = 30
	write a2→A	A = 30
	commit	

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Agenda

Dirty Read (Schreib-Lese-Konflikt, write-read-conflict)

- Beschreibung:
  - Transaktion 1 liest einen von Transaktion 2 veränderten Wert (und arbeitet damit weiter)
  - Transaktion 2 wurde aber noch nicht via `commit` bestätigt
- Problem:
  - Transaktion 2 kann durch `rollback` zurückgerollt werden
  - Transaktion 1 arbeitet dann auf einem **ungültigen Wert**
  - **Verstoß** gegen Eigenschaft der **Isoliertheit**

Dirty Read

A = 100

Transaktion 1	Transaktion 2	
	read a2←A	a2 = 100
	a2 = a2 – 10	a2 = 90
	write a2→A	A = 90
read a1←A		a1 = 90
a1 = a1 + 50		a1 = 140
write a1→A		A = 140
commit		
	rollback	

Dirty Read – Inkonsistenzen

Ziel: Umbuchung von einem Euro

Integritätsbedingung: A + B = 0

`A = 0, B = 0`

Transaktion 1	Transaktion 2	
	read a2←A	a2 = 0
	a2 = a2 – 1	a2 = -1
	write a2→A	A = -1
read a1←A		a1 = -1
read b1←B		b1 = 0
	read b2←B	b2 = 0
	b2 = b + 1	b2 = 1
	write b2→B	B = 1
	commit	

Verstoß gegen Integritätsbedingung

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Agenda

Non-repateable Read (nicht-wiederholbares Lesen, read-write-conflict)

- Beschreibung
  - Transaktion 1 liest ein Datum mehrfach
  - Transaktion 2 ändert dies währenddessen
  - Beide **Transaktionen lesen konsistente Datenbankzustände**
- Problem
  - Transaktion 1 operiert auf **veralteten Daten**
  - Datenbank wird **möglicherweise inkonsistent**

Non-repeatable Read: Beispiel

- Beispiel Lagerhaltung

- Variablen
  - L: Lagerbestand
  - R: Reservierung
  - B: Bestellmenge
  - E: Entnahme

- Konsistenzregel
  - L-R >= 0
  - Bestellungen und Entnahmen dürfen nur durchgeführt werden, wenn die Bedingung erfüllt bleibt

L = 200, R = 100, B = 100, E = 50

Transaktion 1	Transaktion 2	
read l1←L		l1 = 200
read r1←R		r1 = 100
read b1←B		b1 = 100
l1 = l1 – r1 - b1		l1 = 0
	read l2←L	l2 = 200
	read r2←R	r2 = 100
	read e2←E	e2 = 50
	l2 = l2 – r2 – e2	l2 = 50
	read l2←L	l2 = 200
	l2 = l2 - 50	l2 = 150
	write l2→L	L = 150
	commit	
read l1←L		l1 = 150
l1 = l1 – b1		l1 = 50
write l1→L		L = 50
commit		

Verfügbarkeitsprüfung

2. Lesen mit anderem Ergebnis (aber Schreiben ohne Prüfung)

Geplant 100 (Reservierungen!)

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Agenda

Phantom

- Beschreibung
  - Entsteht bei **Operationen auf mehreren Tupeln**
  - Ähnlich dem non-repeatable Read
  - Bezieht sich auf **Menge von Daten** nicht auf ein Datum

- Problem
  - Operation auf Basis von **veralteten** Daten
  - Datenbank wird möglicherweise **inkonsistent**

Phantom (Inconsistent Read)

Ziel: Berechnung des durchschnittlichen Umsatz pro Kunde

Transaktion 1	Transaktion 2	
`read a1←count(Kunde)`		`a1 = 110`
	`insert into Kunde`	
	`commit`	
`read u1←sum(Umsatz)`		`u1 = 220`
`du1 = u1 / a1`		`du1 = 2`
`write du1→DU`		`DU = 2`
`commit`		

Korrekt jetzt: `a1 = 111`

Korrekter Wert: 1,982

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Agenda

1. **ACID-Eigenschaften**
2. **Typische Probleme**
3. **Lösungsansätze**

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Ansatz zur Problemlösung

- Möglicher Ansatz für DBMS
  - Ausführung von Transaktionen seriell (der Reihe nach) führt immer zu konsistenten Datenbankzuständen
  - Daher **nur serielle** Transaktionen zulassen

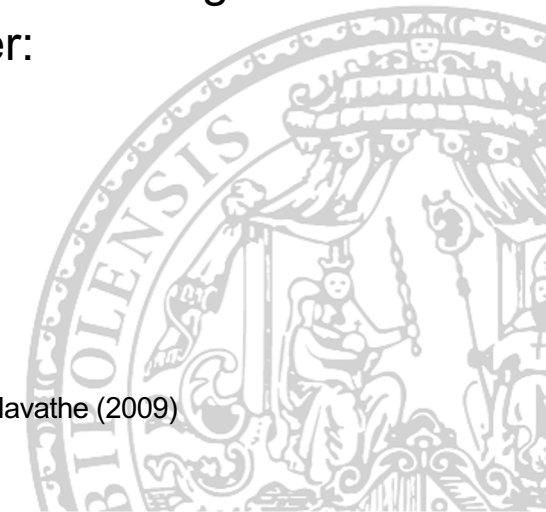
- Problem
  - **Geringe Effizienz**, da Transaktionen blockiert werden, die nicht auf gemeinsamen Daten arbeiten

- Daher sollten **möglichst viele Transaktionen parallel** ausgeführt werden, ohne dass es zu Fehlern kommt
  - Serialisierbarkeit als **Gütekriterium** für die Synchronisierung von Transaktionen

Serialisierbarkeit

- Ein System von parallelen Transaktionen ist dann korrekt **synchronisiert**, wenn es **serialisierbar** ist
- Ein Ausführungsplan ist dann seriell, wenn alle Operationen des Plans **vollständig hintereinander** ausgeführt werden könn(t)en
- D.h. es ist/ wäre immer nur **eine Transaktion aktiv**, Transaktionen überlappen nicht

- Verfahren zur Gewährleistung von Serialisierbarkeit
  - **Verifizierende** Verfahren
    - Testen zu bestimmten Zeitpunkten, ob Serialisierbarkeit noch gegeben ist
    - Wenn nicht, wird eine geeignete Transaktion zurückgesetzt
  - **Präventive** Verfahren
    - Verhindern die Entstehung von nicht-serialisierbaren Transaktionsfolgen
    - Bekanntester Vertreter: **Sperrverfahren**



Elmasri, Navathe (2009)

Lese- und Schreibsperren

- Instrument zur **Synchronisation paralleler Transaktionen**
- Jede **Transaktion sperrt** benötigte Objekte
- Arten von Sperren
  - **Lesesperre** (`rlock`):
    - Erlaubt Lesen eines Datenbankobjektes
    - Beliebig viele Lesesperren auf einem Datenbankobjekt
  - **Schreibsperre** (`wlock`):
    - Erlaubt Lesen und Schreiben eines Datenbankobjektes
    - Exklusive Sperre: Keine weiteren Lese- oder Schreibsperren erlaubt
- Verwendung von Sperren allein **garantiert keine Serialisierbarkeit**

→ Spezielle Protokolle notwendig

A = 1, B = 1

Transaktion 1	Transaktion 2	
read a1←A		a1 = 1
a1 = a1 + 1		a1 = 2
write a1→A		A = 2
read b1←B		b1 = 1
b1 = a1		b1 = 2
write b1→B		B = 2
commit		
	read b2←B	b2 = 2
	b2 = b2 + 2	b2 = 4
	write b2→B	B = 4
	read a2←A	a2 = 2
	a2 = b2	a2 = 4
	write a2→A	A = 4
	commit	

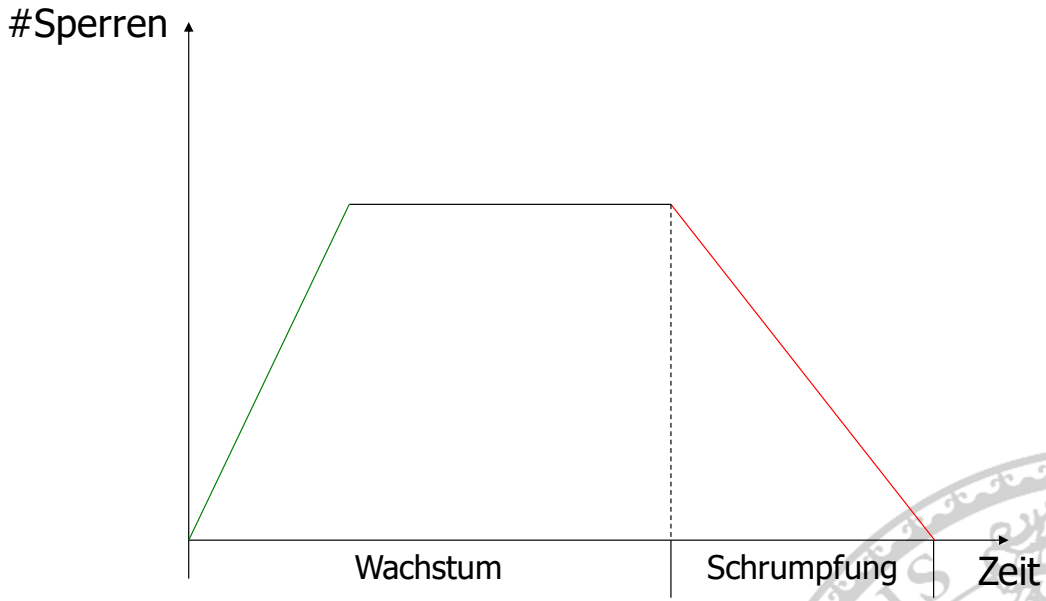
Jede serielle Ausführung ergibt A = 4, B = 4

Transaktion 1	Transaktion 2	
wlock A		
	wlock B	
read a1←A		a1 = 1
	read b2←B	b2 = 1
a1 = a1 + 1		a1 = 2
	b2 = b2 + 2	b2 = 3
write a1→A		A = 2
	write b2→B	B = 3
unlock A		
	unlock B	
wlock B		
	wlock A	
read b1←B		b1 = 3
	read a2←A	a2 = 2
b1 = a1		b1 = 2
	a2 = b2	a2 = 3
write b1→B		B = 2
	write a2→A	A = 3
unlock B; commit		
	unlock A; commit	

Verstoß gegen Kriterium der Serialisierbarkeit

Zwei-Phasen-Sperrprotokoll

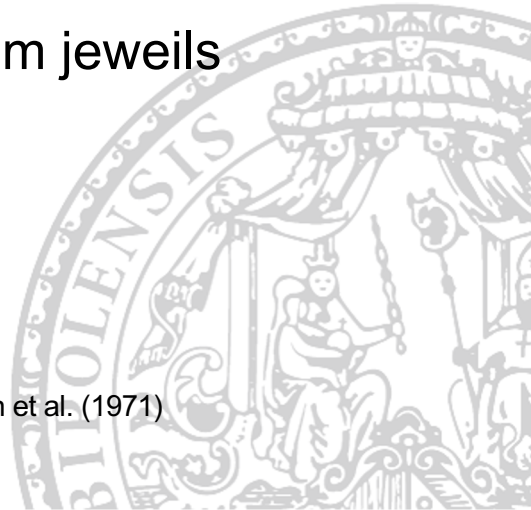
- **Garantiert die Serialisierbarkeit** von Transaktionen
- Anfordern und Freigeben von Sperren in getrennten Phasen
  - 1. Phase: Anforderung der Sperren
  - 2. Phase: Freigabe der Sperren
- Hat eine Transaktion eine Sperre freigegeben, kann sie keine neuen mehr anfordern



Deadlock

- Auch: Verklemmung
- Zyklische Wartesituation zwischen Prozessen

- Kriterien
  - **Mutual Exclusion:** Der Zugriff auf die Resourcen ist exklusiv
  - **Wait for:** Der Zugriff auf Resourcen wird aufrechterhalten während neue angefordert werden
  - **No Preemption:** Resourcen können nicht entzogen werden bis die Aufgabe abgeschlossen wurde
  - **Circular Wait:** Mindestens zwei Prozesse fordern Resourcen an, die vom jeweils anderen Prozess benötigt werden



Coffman et al. (1971)

Zwei-Phasen-Sperrprotokoll: Beispiel (I)

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Transaktion 1	Transaktion 2	
rlock A		
	rlock B	
read a1←A		a1 = 1
	read b2←B	b2 = 1
a1 = a1 + 1		a1 = 2
	b2 = b2 + 2	b2 = 3
wlock A		
	wlock B	
write a1→A		A = 2
	write b2→B	B = 3
rlock B: Muss warten, da B gesperrt		
	rlock A: Muss warten, da A gesperrt	

Deadlock

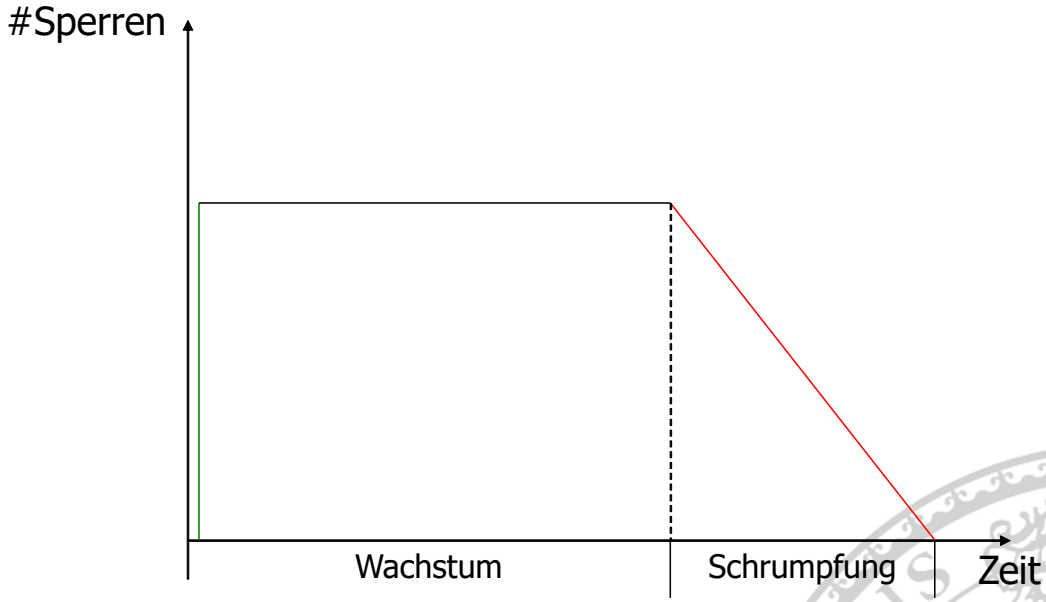
DBMS erkennt Deadlock und setzt eine Transaktion zurück

Zwei-Phasen-Sperrprotokoll - Beispiel (II)

Transaktion 1	Transaktion 2	
	Deadlock	
	rollback: B wird frei und auf 1 zurückgesetzt	B = 1
rlock B (kann jetzt gesetzt werden)		
read b1←B		b1 = 1
b1 = a1		b1 = 2
wlock B		
	rlock B: muss warten	b2 = 1
write b1→B		B = 2
unlock B		
	read b2←B	b2 = 2
unlock A, commit		
	b2 + 2	b2 = 4
	wlock B	B = 4
	write b2→B	B = 4
	rlock A	
	read a2←A	a2 = 2
	a2 = b2	a2 = 4
	wlock A	
	write a2→A	A = 4
	unlock A, unlock B, commit	

2-Phasen-Sperrprotokoll - Preclaiming

- Transaktionen müssen **zu Beginn** von anderen Operationen alle **Sperren anfordern**
- Vorteil
  - **Kein Deadlock** mehr für Transaktion, wenn alle Sperren erhalten
  - **Rollback einfach**, da noch keine Operationen durchgeführt
- Nachteil
  - Alle notwendigen **Sperren** müssen zu Beginn der Transaktion **bekannt** sein
  - **Eingeschränkte Parallelität**, da Sperren länger als notwendig gehalten werden

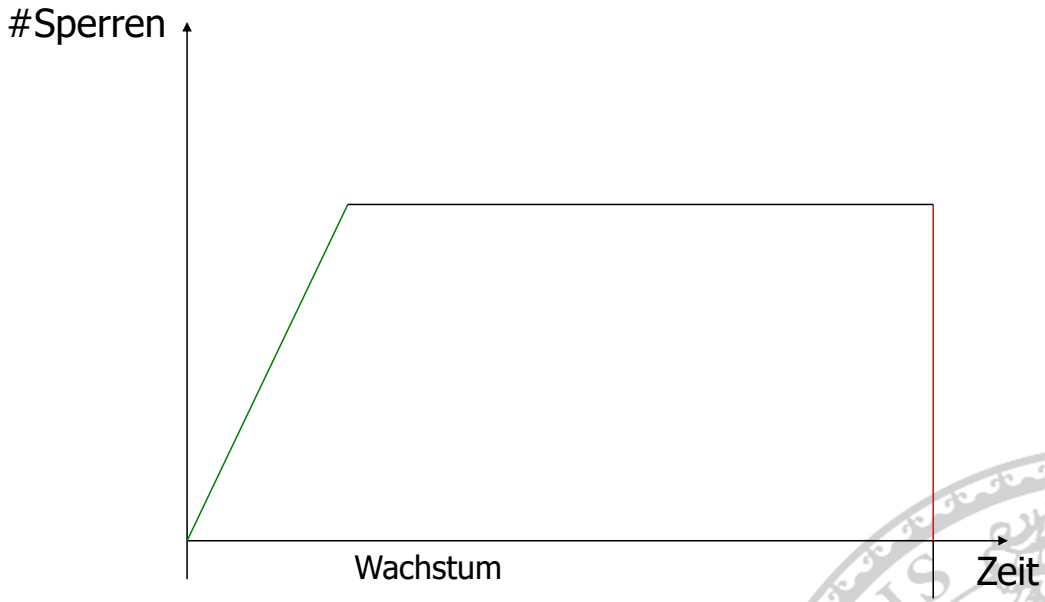


2-Phasen-Sperrprotokoll – Preclaiming: Beispiel

Transaktion 1	Transaktion 2	
wlock A		
wlock B		
	wlock B: Muss warten	
read a1←A		a1 = 1
a1 = a1 + 1		a1 = 2
write a1→A		A = 2
unlock A		
Read b1←B		b2 = 1
b1 = a1		b2 = 2
write b1→B		B = 2
unlock B		
commit		
	read b2←B	b2 = 2
	b2 = b2 + 2	b2 = 4
…	…	…

2-Phasen-Sperrprotokoll - Sperren bis EOT

- Sperren werden **bis zum Ende** der Transaktion gehalten
- Vorteil
  - Muss eine Transaktion zurückgesetzt werden, hat **keine** andere Transaktion bearbeitete **Werte gelesen**
- Nachteil
  - **Kein Deadlock-Schutz**
  - **Eingeschränkte Parallelität**, da Sperren länger als notwendig gehalten werden



Kemper, Eickler (2013)

2-Phasen-Sperrprotokoll – EOT: Beispiel

Transaktion 1	Transaktion 2	
wlock A		
wlock B		
	rlock B: Muss warten	
read a1←A		a1 = 1
a1 = a1 + 1		a1 = 2
write a1→A		A = 2
Read b1←B		b2 = 1
b1 = a1		b2 = 2
write b1→B		B = 2
unlock A		
unlock B		
commit		
	read b2←B	b2 = 2
	b2 = b2 + 2	b2 = 4
…	…	…

A wird auch erst am Ende entsperrt!