



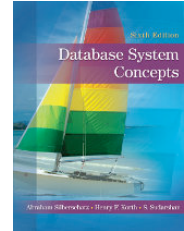
# Datenbanken: Nebenläufigkeitskontrolle durch Zeitstempel- und optimistische Verfahren

Dr. Matthias Uflacker, Stefan Halfpap, Dr. Werner Sinzig

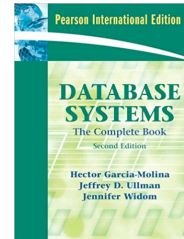
24. Juni 2019

- Einführung zu Unternehmensanwendungen (2 Vorlesungen)
- Einführung zu relationalen Datenbanken und Anfrageverarbeitung (2 Vorlesungen)
- Grundlagen des IT-gestützten Rechnungswesens und der Planung (3 Vorlesungen)
- Grundlagen von (spaltenorientierten) Hauptspeicherdatenbanken (5 Vorlesungen)
- **Trends in Hauptspeicherdatenbanken (4 Vorlesungen)**
- Klausur

- Avi Silberschatz „Database System Concepts“



- Hector Garcia-Molina „Database Systems“



- Andy Pavlo „Database Systems“ – Timestamp Ordering & Multi Version Concurrency Control

<https://15445.courses.cs.cmu.edu/fall2018/slides/18-timestampordering.pdf>

<https://15445.courses.cs.cmu.edu/fall2018/slides/19-multiversioning.pdf>

# Nebenläufigkeitskontrolle

## Motivation

---

- Bisher: Fokus auf schnelle Anfragebearbeitung
- Aber: Haupteigenschaft von relationalen DBs sind **konsistente**/korrekte Anfrageergebnisse, auch bei nebenläufigen Anfragen und trotz Auftreten von Fehlern (insbesondere wichtig für Unternehmensanwendungen, z.B. in der Buchführung)  
Durch **Nebenläufigkeitskontrolle** und Loggen + Wiederherstellung

- Konsistenz ist im DB-Kontext über Transaktionen definiert
- Transaktion: Ausführung einer Folge von (einer oder mehrerer) Operationen
  - Elementare Änderungseinheit in einer Datenbank
  - Erfüllen (abhängig vom Isolationslevel) ACID Kriterien
    - Atomicity
    - Consistency
    - Isolation
    - Durability

- Gegeben ein konsistenter Datenbankzustand: eine isoliert ausgeführte Transaktion führt zu einem konsistenten Datenbankzustand
    - Commit: erfolgreiche Ausführung aller Änderungen
    - Abort: Abbruch durch Nutzer oder DBMS; inklusive Rückgängigmachung aller Änderungen
  - Aber serielle Ausführung von Transaktionen in den meisten Fällen nicht praktikabel
    - Begrenzter Vorteil durch parallele Ausführung (Intra- vs. Inter-Query-Parallelisierung)
    - Lange laufende Transaktionen blockieren kurze
- Ziel: korrekte/serialisierbare Ausführung von nebenläufigen Transaktionen  
Serialisierbare Ausführung (Sreihenfolge) führt zu gleichem Ergebnis wie serielle Ausführung

Ziel: Serialisierbarkeit

- Wir brauchen einen Weg um Datenbankoperationen von nebenläufigen Transaktionen zu kontrollieren/steuern/synchronisieren

→ Nebenläufigkeitskontrolle

- Klassische Variante: Locks/Sperren: 2PL (2 phase locking) 2-Phasen-Sperrprotokolle

- Sperrphase (growing phase) gefolgt von Freigabephase (shrinking phase)

- Lock-Typen (exklusives vs. geteiltes)

- Deadlock-Verhinderung oder Erkennung + Wiederherstellung notwendig

- Reihenfolge von (in Konflikt stehenden) Transaktionen wird zur Ausführungszeit (durch die Ausführungszeit) bestimmt

} DBS 1

- **Hier: alternative Varianten**

# Nebenläufigkeitskontrolle

## Alternativen zu 2PL

---

- Zeitstempelverfahren (Timestamp Ordering)  
Idee: Bestimme Ausführungsreihenfolge zum Transaktionsbeginn
  - Basisvariante
  - Multiversion Concurrency Control (MVCC)
  
- Optimistische Nebenläufigkeitskontrolle  
Idee: validiere Transaktionen nach der Ausführung; Reihenfolge wird durch den Validierungszeitpunkt bestimmt



Idee:

- Weise jeder Transaktion zum Beginn einen eindeutigen, aufsteigenden Zeitstempel zu
  - Der **Zeitstempel markiert den logischen Ausführungszeitpunkt**
- bestimmt die logische Reihenfolge der Transaktionen → Serialisierbarkeit

Praktische Umsetzung:

- Transaktion T benötigt Zeitstempel  $TS(T)$ , z.B. durch Systemuhr oder logischen Zähler
- Datenelement I benötigt zwei Zeitstempel, die aktualisiert werden müssen
  - W- $TS(I)$  speichert den größten Zeitstempel einer Transaktion, die das Element erfolgreich geschrieben hat
  - R- $TS(I)$  speichert den größten Zeitstempel einer Transaktion, die das Element erfolgreich gelesen hat

# Nebenläufigkeitskontrolle

## Zeitstempelverfahren – Basisvariante – Beispiel 1

T1	T2	TS
READ(A)		1
	READ(A)	2
	WRITE(A)	3
READ(B)		4
	READ(B)	5
	WRITE(B)	6

$TS(T1) = 1$

$TS(T2) = 2$

Element l	R-TS(l)	W-TS(l)
A	$0 \rightarrow_1 1 \rightarrow_2 2$	$0 \rightarrow_3 2$
B	$0 \rightarrow_4 1 \rightarrow_5 2$	$0 \rightarrow_6 2$

### **Zeitstempelfverfahren müssen sicherstellen, dass in Konflikt stehende Operationen in der Reihenfolge ihrer Zeitstempel ausgeführt werden**

Leseoperation: Transaktion T möchte Element I lesen

- Fall 1:  $TS(T) < W-TS(I)$ 
  - I wurde bereits von einer späteren Transaktion verändert (geschrieben)
  - Leseoperation ist nicht möglich und T muss abbrechen
- Fall 2:  $TS(T) \geq W-TS(I)$ 
  - Leseoperation ist möglich und wird ausgeführt
  - $R-TS(I) = \max(R-TS(I), TS(T))$

# Nebenläufigkeitskontrolle

## Zeitstempelverfahren – Basisvariante – Beispiel 2

T1	T2	TS
READ(A)		1
	WRITE(B)	2
READ(B)		3

$TS(T1) = 1$

$TS(T2) = 2$

$TS(T1) == 1 < 2 == W-TS(B)$   
Leseoperation ist nicht möglich

Element l	R-TS(l)	W-TS(l)
A	$0 \rightarrow_1 1$	0
B	0	$0 \rightarrow_2 2$

# Nebenläufigkeitskontrolle

## Zeitstempelverfahren – Basisvariante – Protokoll

Schreiboperation: Transaktion T möchte Element I schreiben

- Fall 1:  $TS(T) < R-TS(I)$ 
  - I wurde bereits von einer späteren Transaktion gelesen
  - Schreiboperation ist nicht möglich und T muss abbrechen
- Fall 2:  $TS(T) < W-TS(I)$ 
  - I wurde von einer späteren Transaktion überschrieben
  - Variante 1: Schreiboperation ist veraltet und wird abgebrochen
  - Variante 2: Schreiboperation ist unnötig und wird ignoriert (Thomas' Write Rule)
- Fall 3: ansonsten
  - Schreiboperation ist möglich und wird ausgeführt
  - $W-TS(I) = TS(T)$

# Nebenläufigkeitskontrolle

## Zeitstempelverfahren – Basisvariante – Beispiel 3

T1	T2	TS
READ(A)		1
	READ(A)	2
WRITE(A)		3

$$TS(T1) = 1$$

$$TS(T2) = 2$$

$TS(T1) == 1 < 2 == R-TS(A)$   
Schreiboperation ist nicht möglich

Element l	R-TS(l)	W-TS(l)
A	$0 \rightarrow_1 1 \rightarrow_2 2$	0

# Nebenläufigkeitskontrolle

## Zeitstempelverfahren – Basisvariante – Beispiel 4

T1	T2	TS
READ(A)		1
	WRITE(B)	2
WRITE(B)		3

$TS(T1) = 1$

$TS(T2) = 2$

$TS(T1) == 1 < 2 == W-TS(B)$   
Schreiboperation ist veraltet, kann  
aber ignoriert werden

Element l	R-TS(l)	W-TS(l)
A	$0 \rightarrow_1 1$	0
B	0	$0 \rightarrow_2 2$

# Nebenläufigkeitskontrolle

## Zeitstempelfverfahren – Basisvariante – Protokoll

---

Abgebrochene Transaktionen

- Müssen Änderungen rückgängig machen
- Bekommen neuen Zeitstempel
- Starten von vorne

**Achtung: Basisvariante erlaubt nicht wiederherstellbare Reihenfolgen von Operationen, insbesondere durch das Lesen von uncommitteten Änderungen**

Folgende Erweiterungen sind möglich:

- Variante 1: Alle Schreiboperationen atomar am Ende der Transaktion durchführen
- Variante 2: Transaktionen warten mit dem Lesen von uncommitteten Änderungen
- Variante 3: Spätere Überprüfung nach dem Lesen uncommitteter Änderungen



# Nebenläufigkeitskontrolle

## Zeitstempelverfahren – Basisvariante – Beispiel 5

T1	T2	TS
WRITE(A)		1
	READ(A)	2
ABORT		3

$TS(T1) = 1$

$TS(T2) = 2$

T1 kann abbrechen, obwohl T2 die neue uncommittete Version von Element A bereits gelesen hat

Element l	R-TS(l)	W-TS(l)
A	$0 \rightarrow_2 2$	$0 \rightarrow_1 1$

# Nebenläufigkeitskontrolle

## Zeitstempelverfahren – Basisvariante – Beobachtungen

- **Mehraufwand** durch lokale Kopien und Aktualisierungen der Zeitstempel
- Lang laufende Transaktionen können wiederholt abrechnen  
(Mit der Dauer einer Transaktion steigt die Wahrscheinlichkeit, dass die Transaktion veränderte Elemente einer neueren Transaktionen lesen möchte)
- Im Falle vieler lesender Transaktionen ist die Anzahl der Konflikte niedrig  
→ Idee: Optimiere für den Fall ohne Konflikte – **Optimistische** Nebenläufigkeitskontrolle  
(als Gegenteil der beiden pessimistischen Verfahren: 2PL und Zeitstempelverfahren)  
Umsetzung: Nebenläufigkeitskontrolle durch Validierung (spätere Überprüfung)

2 oder 3 Phasen pro Transaktion

■ Phase 1: Lesephase

- Speichere die Menge der gelesenen und geschriebenen Elemente lokal

■ Phase 2: Validierungsphase

- Überprüfe zum Commit-Zeitpunkt, ob es Konflikte mit anderen Transaktionen gibt
- Fall 1: erfolgreiche Validierung: beginne mit der Schreibphase
- Fall 2: starte die Transaktion ansonsten neu

■ (optionale) Phase 3: Schreibphase

- Wende lokale Änderungen auf Datenbank an

Praktische Umsetzung:

- Alle Transaktionen müssen die Phasen in der gezeigten Reihenfolge durchlaufen
- Phasen nebenläufiger Transaktionen können überlappen
- Validierung muss Zeiträume der Phasen kennen

→ Speichere pro Transaktion drei Zeitstempel

- START(T) speichert Beginn der Transaktion
- VAL(T) speichert Beginn der Validierungsphase (/ Ende der Lesephase)
- FIN(T) speichert Ende der Schreibphase

**Die Reihenfolge der Transaktionen ist durch die Validierungszeitstempel VAL(T) bestimmt**

Validierungstests für Transaktion  $T_i$  mit allen vorherigen Transaktionen  $T_k$  mit  $VAL(T_k) < VAL(T_i)$  muss folgende Bedingungen erfüllen

- Fall 1:  $FIN(T_k) < START(T_i)$   
Validierung mit  $T_k$  trivial, da  $T_k$  vor Beginn von  $T_i$  beendet war
- Fall 2: ansonsten  
Keine Überlappung der geschriebenen Elemente von  $T_k$  mit den gelesenen Elementen von  $T_i$

# Nebenläufigkeitskontrolle durch Validierung – Beispiel

T1	T2	TS
READ(A)		1
	WRITE(B)	2
READ(B)		3
VALIDATE		4
	READ(A)	5
	VALIDATE	6
	WRITE	7

lokal

$VAL(T1) == FIN(T1) = 4$ ; Validierung trivial, da keine Transaktion vor T1

$VAL(T2) = 6$ ; Validierung trivial, da T1 nicht schreibt

global;  $FIN(T2) = 7$

# Nebenläufigkeitskontrolle

## Multiversion Concurrency Control

---

Bisher: Serialisierbarkeit durch Verschieben/Sperren von Operationen oder Transaktionsabbruch:

- Leseoperation muss warten, da der Wert noch nicht geschrieben oder committet ist
- Transaktion muss abbrechen, da zu lesender Wert bereits überschrieben wurde

Idee: Vermeide diese Probleme durch das Aufbewahren alter Versionen der geänderten Elemente

**DB speichert mehrere physische Versionen von einem logischen Element**, z.B. Tupel

# Nebenläufigkeitskontrolle

## Multiversion Concurrency Control

DB speichert mehrere physische Versionen von einem logischen Element, z.B. Tupel

- Änderungen eines Elements erzeugen eine neue Version  
(siehe 08\_DB\_Weitere\_Optimierungen - Insert-Only-Ansatz)
- Transaktionen lesen die neueste Version, die zum Transaktionsstart existierte

→ Leseoperationen blockieren nicht Schreiboperationen

→ Schreiboperationen blockieren nicht Leseoperationen

+ Snapshot-Isolation (und Zeitreise-Queries)

+ Möglicherweise gesetzlich vorgeschrieben

- Speicherverbrauch

- Höhere Scan-Kosten

} siehe 08\_DB\_Weitere\_Optimierungen



# Nebenläufigkeitskontrolle

## Multiversion Concurrency Control - Protokoll

---

### Anpassungen an Basisprotokoll

- Datenelement I hat (potenziell) mehrere Versionen mit je zwei Zeitstempeln
  - $W-TS(I_T)$  /  $BEGIN-TS(I_T)$  speichert den Zeitstempel der erstellenden Transaktion  $TS(T)$ 
    - wird genutzt um die passende Version für Leseoperationen zu finden
  - $END-TS(I_T)$  speichert, falls gesetzt, den Beginn-Zeitstempel der neuen Version
    - wird genutzt um die passende Version für Leseoperationen zu finden
  - $R-TS(I_T)$  funktioniert analog zum Basisprotokoll
    - wird genutzt um unmögliche Schreiboperationen abzulehnen
- Nicht mehr benötigte Versionen können gelöscht werden

# Nebenläufigkeitskontrolle

## Multiversion Concurrency Control – Beispiel

T1	T2	TS
READ(A)		1
	WRITE(B)	2
READ(B)		3

$TS(T1) = 1$

$TS(T2) = 2$ ;  $B_2$  wird erzeugt

Version  $B_0$  wird gelesen

Element l	R-TS(l)	W-TS(l)	END-TS(l)
A	$0 \rightarrow_1 1$	0	$\infty$
$B_0$	$0 \rightarrow_3 1$	0	$\infty \rightarrow_2 2$
$\rightarrow_2 B_2$	$\rightarrow_2 2$	$\rightarrow_2 2$	$\rightarrow_2 \infty$

# Nebenläufigkeitskontrolle

## Multiversion Concurrency Control - Designentscheidungen

„MVCC is more than just a concurrency control protocol.

It completely affects how the DBMS manages transactions and the database.“

Andy Pavlo „Multi Version Concurrency Control“ Vorlesung

<https://15445.courses.cs.cmu.edu/fall2018/slides/19-multiversioning.pdf>

MVCC Implementierungen erfordern mehrere Designentscheidungen:

- Nebenläufigkeitsprotokoll: es existieren auch 2PL- und optimistische Varianten
- Versionsspeicherung: ob und wie einzelne Versionen verlinkt sind  
(effizientes Finden der richtigen Version ist insbesondere für transaktionale Workloads wichtig)
- Speicherbereinigung: wann und wie werden alte Versionen erkannt und gelöscht
- Indexverwaltung: auf welche Versionen verweisen Indexe

- Bisher betrachtete Operationen: Daten lesen und aktualisieren (feste Menge von Datenelementen)
- INSERTS und DELETES erzeugen weitere Probleme, z.B. Phantom-Problem

Prädikat-Locking und wiederholter Scan zum Commit als Techniken um damit umzugehen

Aber: Serialisierbarkeit limitiert Nebenläufigkeit stark und damit Performanz

- Schwächere Konsistenzlevel als Alternativen
- Insbesondere ist das Standard-( oder sogar höchste )Isolationslevel von Datenbanken nicht immer Serialisierbarkeit

<http://www.bailis.org/blog/when-is-acid-acid-rarely/>

# Nebenläufigkeitskontrolle

## SQL-92 Isolationslevel (Überblick)

	<b>Lesen uncommitteter Änderungen</b>	<b>Nicht wiederholbares Lesen</b>	<b>Phantom</b>
<b>Serializable</b>	<b>Unmöglich</b>	<b>Unmöglich</b>	<b>Unmöglich</b>
<b>Repeatable Read</b>	<b>Unmöglich</b>	<b>Unmöglich</b>	<b>Möglich</b>
<b>Read Committed</b>	<b>Unmöglich</b>	<b>Möglich</b>	<b>Möglich</b>
<b>Read Uncommitted</b>	<b>Möglich</b>	<b>Möglich</b>	<b>Möglich</b>

- Zeitstempel- und optimistische Verfahren zur Nebenläufigkeitskontrolle sind Alternativen zu 2PL um konsistente Anfrageergebnisse zu gewährleisten
  - Zeitstempelverfahren: Zeitstempel definiert die Ausführungsreihenfolge
  - Optimistische Verfahren: Validierungszeitpunkt definiert die Ausführungsreihenfolge
- Multiversion Concurrency Control verhindert Transaktionsverzögerungen und -abbrüche durch das Speichern älterer Versionen von Datenelementen