Transaktionsverwaltung

Inhalt

Transaktionskonzept
Überblick Recovery
Überblick Synchronisation
Überblick TA-Verwaltung in verteilten Systemen
2-Phasen-Commit-Protokoll

TA-Konzept (1)

- Gefährdung der DB-Konsistenz

	Korrektheit der Abbildungshierarchie	*Übereinstimmung zwischen DB und Miniwelt*
Durch das Anwendungs-programm	Mehrbenutzeranomalien **Synchronisation**	Unzulässige Änderungen **Integritätsüberwachung des DBVS** **TA-orientierte Verarbeitung**
Durch das DBVS und die Betriebsumgebung	Fehler auf den Externspeichern **Fehlertolerante Implementierung** **Archivkopien (Backup)**	Undefinierter DB-Zustand nach einem Systemausfall **TA-orientierte Fehlerbehandlung**

TA-Konzept (2)

- Ablaufkontrollstruktur: Transaktion (TA)
  - *Eine Transaktion ist eine ununterbrechbare Folge von DML-Befehlen, die die Datenbank von einem logisch konsistenten in einen (neuen) logisch konsistenten Zustand überführt.*
  - Beispiel eines TA-Programms:

```
BOT
UPDATE Konto
...
UPDATE Schalter
...
UPDATE Zweigstelle
...
INSERT INTO Ablage (...)
COMMIT
```


TA-Konzept (3)

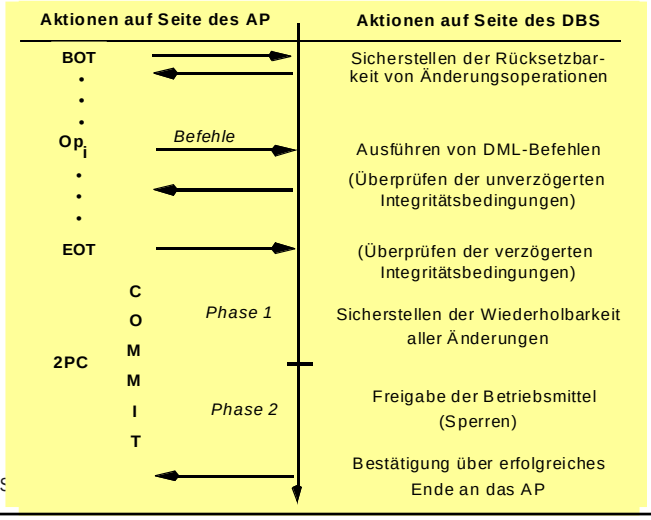
- Ablaufkontrollstruktur: Transaktion (Forts.)
  - *ACID*-Eigenschaften von Transaktionen
    - **Atomicity (Atomarität)**
      - TA ist kleinste, nicht mehr weiter zerlegbare Einheit
      - Entweder werden alle Änderungen der TA festgeschrieben oder gar keine („alles-oder-nichts“-Prinzip)
    - **Consistency**
      - TA hinterlässt einen konsistenten DB-Zustand, sonst wird sie komplett (siehe Atomarität) zurückgesetzt
      - Zwischenzustände während der TA-Bearbeitung dürfen inkonsistent sein
      - Endzustand muss alle definierten Integritätsbedingungen erfüllen

TA-Konzept (4)

- Ablaufkontrollstruktur: Transaktion (Forts.)
  - ***ACID***-Eigenschaften von Transaktionen (Forts.)
    - **Isolation**
      - Nebenläufig (parallel, gleichzeitig) ausgeführte TA dürfen sich nicht gegenseitig beeinflussen
      - Parallele TA bzw. deren Effekte sind nicht sichtbar (logischer Einbenutzerbetrieb)
    - **Durability (Dauerhaftigkeit)**
      - Wirkung erfolgreich abgeschlossener TA bleibt dauerhaft in der DB
      - TA-Verwaltung muss sicherstellen, das dies auch nach einem Systemfehler (HW- oder System-SW) gewährleistet ist
      - Wirkung einer erfolgreich abgeschlossenen TA kann nur durch eine sog. kompensierende TA aufgehoben werden

TA-Konzept (5)

- Schnittstelle zwischen Anwendungsprogramm (AP) und DBS

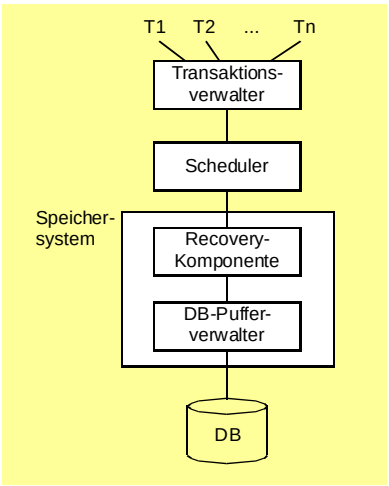


TA-Konzept (6) / TA-Verwaltung (1)

- Wesentliche Abstraktionen (aus Sicht der DB-Anwendung) zur Gewährleistung einer ‚fehlerfreien Sicht' auf die Datenbank im logischen Einbenutzerbetrieb
  - Alle Auswirkungen auftretender Fehler bleiben der Anwendung verborgen (*failure transparency*)
  - Es sind keine anwendungsseitigen Vorkehrungen zu treffen, um Effekte der Nebenläufigkeit beim DB-Zugriff auszuschließen (*concurrency transparency*)
- TA-Verwaltung
  - koordiniert alle DBS-seitigen Maßnahmen, um ACID zu garantieren
  - besitzt zwei wesentliche Komponenten
    - Synchronisation
    - Logging und Recovery
  - kann zentralisiert oder verteilt (z.B. bei VDBS) realisiert sein
  - soll Transaktionsschutz für heterogene Komponenten bieten

TA-Verwaltung (2)

- Abstraktes Architekturmodell (für das Read/Write-Modell auf Seitenbasis)

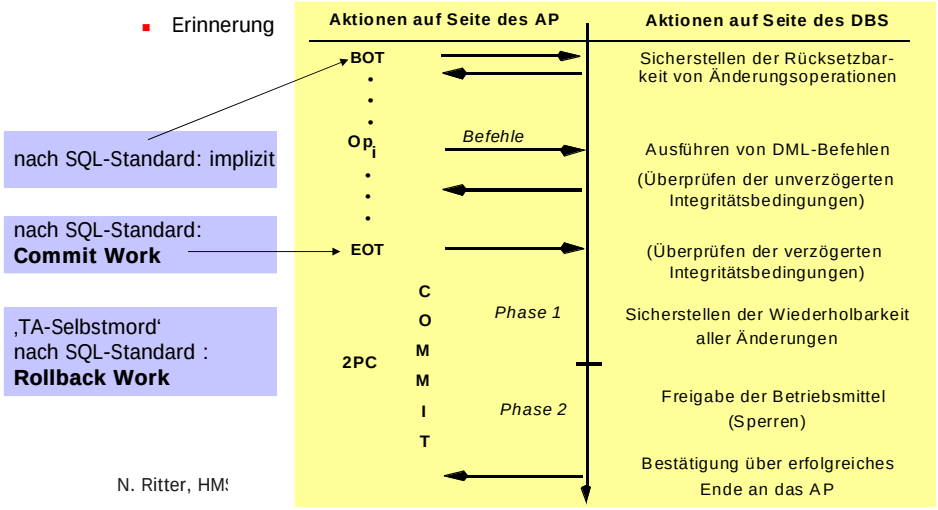


TA-Verwaltung (3)

- Komponenten (vgl. Architekturmodell vorangegangene Folie)
  - **Transaktionsverwalter**
    - Verteilung der DB-Operationen in VDBS und Weiterreichen an den Scheduler
    - zeitweise Deaktivierung von TA (bei Überlast)
    - Koordination der Abort- und Commit-Behandlung
  - **Scheduler** (Synchonisation)
    kontrolliert die Abwicklung der um DB-Daten konkurrierenden TA
  - **Recovery**-Komponente
    sorgt für die Rücksetzbarkeit/Wiederholbarkeit der Effekte von TA
  - DB-Pufferverwalter
    stellt DB-Seiten bereit und gewährleistet persistente Seitenänderungen

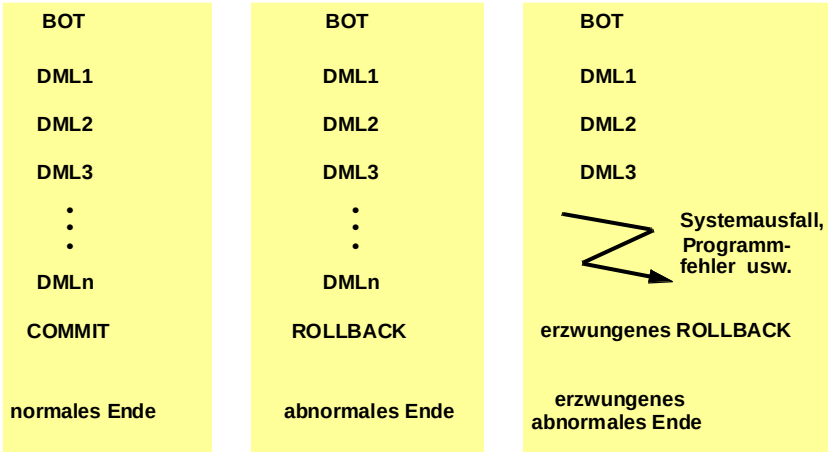
TA-Verwaltung (4)

- Transaktionsablauf
  - Erinnerung



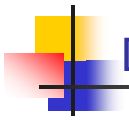
TA-Verwaltung (5)

- Transaktionsablauf (Forts.)
  - Mögliche Ausgänge einer Transaktion



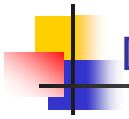
DB-Recovery (1)

- Automatische Behandlung aller ‚erwarteten' Fehler durch das DBVS
- Voraussetzung
  - Sammeln redundanter Information während des normalen Betriebs (Logging)
- Fehlermodell von zentralisierten DBVS
  - Transaktionsfehler
  - Systemfehler
  - Gerätefehler
- Probleme
  - Fehlererkennung
  - Fehlereingrenzung
  - Abschätzung des Schadens
  - Durchführung der Recovery
  - „A recoverable action is 30% harder and requires 20% more code than a non-recoverable action" (J. Gray)



DB-Recovery (2)

- TA-Paradigma verlangt
  - Alles-oder-Nichts-Eigenschaft von TAs
  - Dauerhaftigkeit erfolgreicher Änderungen
- Zielzustand nach erfolgreicher Recovery: jüngster transaktionskonsistenter DB-Zustand
  - Durch die Recovery ist der jüngste Zustand vor Erkennen des Fehlers wiederherzustellen, der allen semantischen Integritätsbedingungen entspricht, der also ein möglichst aktuelles, exaktes Bild der Miniwelt darstellt



DB-Recovery (3)

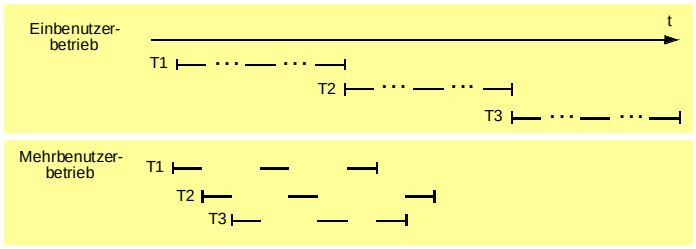
- Recovery-Arten
  1. ***Transaktions-Recovery***
     - Zurücksetzen einzelner (noch nicht abgeschlossener) Transaktionen im laufenden Betrieb (Transaktionsfehler, Deadlock)
     - Arten
       - Vollständiges Zurücksetzen auf Transaktionsbeginn (TA-UNDO)
       - Partielles Zurücksetzen auf Rücksetzpunkt (Savepoint) innerhalb der Transaktion
  2. ***Crash-Recovery*** nach Systemfehler
     - Wiederherstellen des jüngsten transaktionskonsistenten DB-Zustands
     - Notwendige Aktionen
       - (partielles) REDO für erfolgreiche Transaktionen (Wiederholung verlorengegangener Änderungen)
       - UNDO aller durch Ausfall unterbrochenen Transaktionen (Entfernen der Änderungen aus der permanenten DB)

DB-Recovery (4)

- Recovery-Arten (Forts.)
  3. ***Medien-Recovery*** nach Gerätefehler
     - Spiegelplatten bzw.
     - Vollständiges Wiederholen (REDO) aller Änderungen (erfolgreich abgeschlossener Transaktionen) auf einer Archivkopie
  4. ***Katastrophen-Recovery***
     - Nutzung einer aktuellen DB-Kopie in einem ‚entfernten' System oder
     - Stark verzögerte Fortsetzung der DB-Verarbeitung mit repariertem/neuem System auf der Basis gesicherter Archivkopien (Datenverlust)

Synchronisation (1)

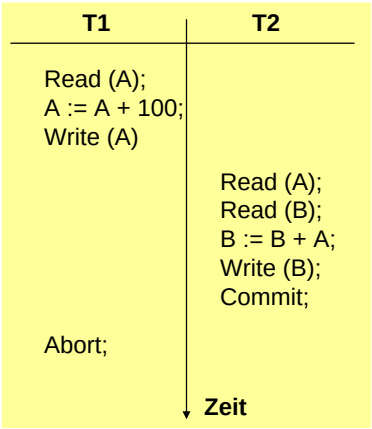
- Einbenutzer-/Mehrbenutzerbetrieb



  - CPU-Nutzung während TA-Unterbrechungen
    - E/A
    - Denkzeiten bei Mehrschritt-TA
    - Kommunikationsvorgänge in verteilten Systemen
  - bei langen TAs zu große Wartezeiten für andere TA (Scheduling-Fairness)

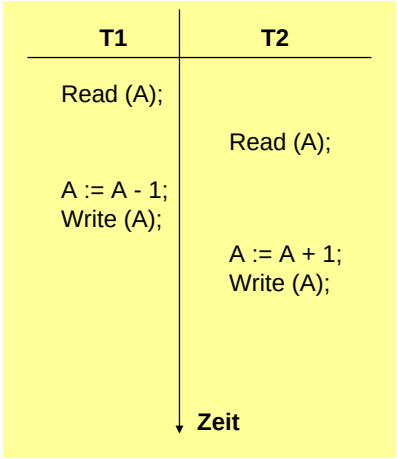
Synchronisation (2)

- Anomalien im unkontrollierten Mehrbenutzerbetrieb
  1. Abhängigkeit von nicht-freigegebenen Änderungen (Dirty-Read)
     - Geänderte, aber noch nicht freigegebene Daten werden als „schmutzig“ bezeichnet (dirty data), da die TA ihre Änderungen bis Commit (einseitig) zurücknehmen kann
     - Schmutzige Daten dürfen von anderen TAs nicht in „kritischen” Operationen benutzt werden



Synchronisation (3)

- Anomalien im unkontrollierten Mehrbenutzerbetrieb
  2. Verlorengegangene Änderung (Lost Update)
     - ist in jedem Fall auszuschließen



Synchronisation (4)

- Anomalien im unkontrollierten Mehrbenutzerbetrieb
  3. Inkonsistente Analyse (Non-repeatable Read)

Lesetransaktion (Gehaltssumme berechnen)	Änderungstransaktion	DB-Inhalt (Pnr, Gehalt)
SELECT Gehalt INTO :gehalt FROM Pers WHERE Pnr = 2345; summe := summe + gehalt;		2345 39.000 3456 48.000
	UPDATE Pers SET Gehalt = Gehalt + 1000 WHERE Pnr = 2345;	2345 40.000
	UPDATE Pers SET Gehalt = Gehalt + 2000 WHERE Pnr = 3456;	3456 50.000
SELECT Gehalt INTO :gehalt FROM Pers WHERE Pnr = 3456; summe := summe + gehalt;		

Zeit

Synchronisation (5)

- Anomalien im unkontrollierten Mehrbenutzerbetrieb
  4. Phantom-Problem

Lesetransaktion (Gehaltssumme überprüfen)	Änderungstransaktion (Einfügen eines neuen Angestellten)
SELECT SUM (Gehalt) INTO :summe FROM Pers WHERE Anr = 17;	
	INSERT INTO Pers (Pnr, Anr, Gehalt) VALUES (4567, 17, 55.000);
	UPDATE Abt SET Gehaltssumme = Gehaltssumme + 55.000 WHERE Anr = 17;
SELECT Gehaltssumme INTO :gsumme FROM Abt WHERE Anr = 17;	
IF gsumme <> summe THEN <Fehlerbehandlung>;	

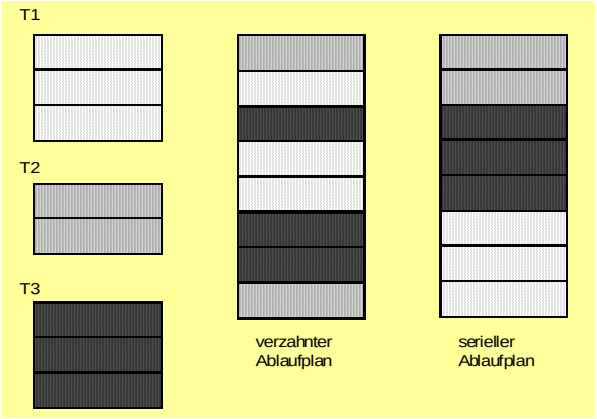
Zeit

Synchronisation (6)

- Korrektheit – Vorüberlegungen
  - einzelne TA T
    - wenn T allein auf einer konsistenten DB ausgeführt wird, dann terminiert T (irgendwann) und hinterlässt die DB in einem konsistenten Zustand
    - während der TA-Verarbeitung gibt es keine Konsistenzgarantien!
  - mehrere TAs
    - wenn Transaktionen seriell ausgeführt werden, dann bleibt die Konsistenz der DB erhalten
    - Ziel der Synchronisation: logischer Einbenutzerbetrieb, d.h. Vermeidung aller Mehrbenutzeranomalien

Synchronisation (7)

- Korrektheit – Vorüberlegungen (Forts.)
  - mehrere TAs (Forts.)



Synchronisation (8)

- Formales Korrektheitskriterium: *Serialisierbarkeit*

  Die parallele Ausführung einer Menge von TA ist serialisierbar,
  wenn es eine serielle Ausführung derselben TA-Menge gibt,
  die *den gleichen DB-Zustand*
  und *die gleichen Ausgabewerte*
  wie die ursprüngliche Ausführung erzielt.

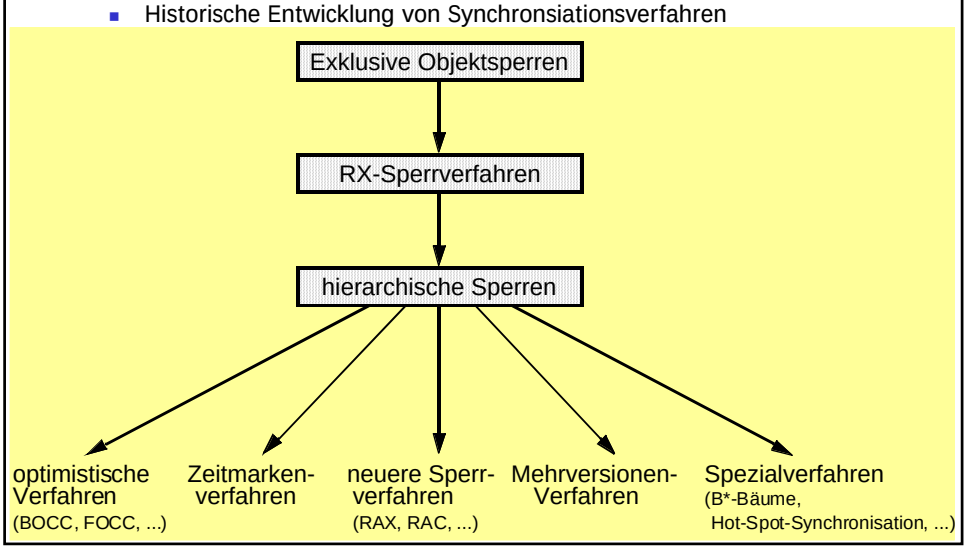
  - Hintergrund:
    - Serielle Ablaufpläne sind korrekt
    - Jeder Ablaufplan, der denselben Effekt wie ein serieller erzielt, ist akzeptierbar

Synchronisation (9)

- Einbettung des DB-Schedulers
  - als Komponente der Transaktionsverwaltung zuständig für I von ACID
  - kontrolliert die beim TA-Ablauf auftretenden Konfliktoperationen (Read/Write, Write/Read, Write/Write) und garantiert insbesondere, dass nur „serialisierbare“ TA erfolgreich beendet werden
  - nicht serialisierbare TAs müssen verhindert werden; dazu ist Kooperation mit der Recovery-Komponente erforderlich (Rücksetzen von TA).
- Zur Realisierung der Synchronisation gibt es viele Verfahren
  - Pessimistisch
  - Optimistisch
  - Versionsverfahren
  - Zeitmarkenverfahren
  - etc.
- Sperrbasierte (pessimistische) Verfahren
  - bei einer Konfliktoperation blockieren sie den Zugriff auf das Objekt
  - universell einsetzbar
  - es gibt viele Varianten

Synchronisation (10)

- Historische Entwicklung von Synchronsiationsverfahren



Synchronisation (11)

- RX-Sperrverfahren
  - Sperrmodi
    - Sperrmodus des Objektes: NL (no lock), R (read), X (exclusive)
    - Sperranforderung einer Transaktion: R, X
  - Kompatibilitätsmatrix

	NL	S	X
S	+	+	-
X	+	-	-

    - Falls Sperre nicht gewährt werden kann, muss die anfordernde TA warten, bis das Objekt freigegeben wird (Commit/Abort der die Sperre besitzenden TA)
    - Wartebeziehungen werden in einem Wait-for-Graph verwaltet

Synchronisation (12)

- RX-Sperrverfahren (Forts.)
  - Beispiel: Ablauf von Transaktionen (aus Sicht des Schedulers; an der SQL-Schnittstelle ist die Sperranforderung und –freigabe nicht sichtbar)

T1	T2	a	b	Bemerkung
		NL	NL	
lock(a, X)		X		
...				
	lock(b, R)		R	
	...			
lock(b, R)			R	
	lock(a, R)	X		T2 wartet
...				
unlock(a)		NL → R		T2 wecken
...	...			
unlock(b)			R	

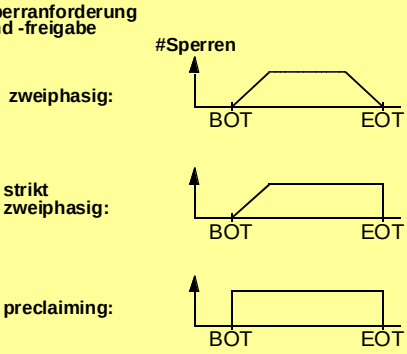
Synchronisation (13)

- RX-Sperrverfahren (Forts.)
  - Einhaltung folgender Regeln gewährleistet Serialisierbarkeit:
    1. Vor jedem Objektzugriff muss Sperre mit ausreichendem Modus angefordert werden
    2. Gesetzte Sperren anderer TA sind zu beachten
    3. Eine TA darf nicht mehrere Sperren für ein Objekt anfordern
    4. Zweiphasigkeit:
       - Anfordern von Sperren erfolgt in einer Wachstumsphase
       - Freigabe der Sperren in Schrumpfungsphase
       - Sperrfreigabe kann erst beginnen, wenn alle benötigten Sperren gehalten werden
    5. Spätestens bei Commit sind alle Sperren freizugeben

Eswaran, K.P. et al.: The notions of consistency and predicate locks in a data base system, Comm. ACM 19:11, 1976, pp. 624-63

Synchronisation (14)

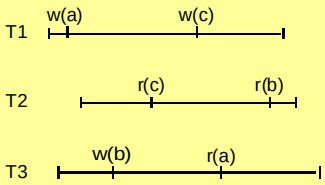
- RX-Sperrverfahren (Forts.)
  - Formen der Zweiphasigkeit
    - Praktischer Einsatz erfordert **striktes 2PL**
      - Gibt alle Sperren erst bei Commit frei
      - Verhindert kaskadierendes Rücksetzen



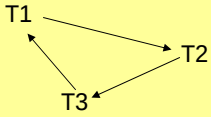
    - Strikte 2PL-Protokolle (verhindern kaskadierendes Rücksetzen)
      - **S2PL** (strict 2PL) gibt alle X-Sperren erst bei Commit frei
      - **SS2PL** (strong 2PL) gibt alle Sperren (R und X) erst bei Commit frei

Synchronisation (15)

- RX-Sperrverfahren (Forts.)
  - Deadlocks/Verklemmungen
    - Auftreten von Verklemmungen ist **inhärent** und kann bei pessimistischen Methoden (blockierende Verfahren) nicht vermieden werden
  - Beispiel einer nicht-serialisierbaren Historie, die zu einer Verklemmung führt



Warte-Beziehungen:



Synchronisation (16)

- RX-Sperrverfahren (Forts.)
  - Allgemeine Forderungen
    - Wahl des gemäß der Operation schwächst möglichen Sperrmodus
    - Möglichkeit der Sperrkonversion (upgrading), falls stärkerer Sperrmodus erforderlich
    - Anwendung: viele Objekte sind zu lesen, aber nur wenige zu aktualisieren
- Erweiterung: RUX
  - Ziel: Verhinderung von Konversions-Deadlocks
  - U-Sperre für Lesen mit Änderungsabsicht (Prüfmodus)
  - bei Änderung Konversion U → X, andernfalls U → R (downgrading)
  - Symmetrische Variante

	R	U	X
R	+	+	-
U	+	-	-
X	-	-	-

Unsymmetrische Variante (z.B. in IBM DB2)

	R	U	X
R	+	-	-
U	+	-	-
X	-	-	-

Synchronisation (17)

- Konsistenzebenen
  - Motivation
    - Serialisierbare Abläufe
      - gewährleisten „automatisch“ Korrektheit des Mehrbenutzerbetriebs
      - erzwingen u. U. lange Blockierungszeiten paralleler Transaktionen
    - „Schwächere“ Konsistenzebene
      - bei der Synchronisation von Leseoperationen
      - erlaubt höhere Parallelitätsgrade und Reduktion von Blockierungen, erfordert aber Programmierdisziplin!
      - Inkaufnahme von Anomalien reduziert die TA-Behinderungen

Synchronisation (18)

- Konsistenzebenen (Forts.) – in SQL
  - SQL erlaubt Wahl zwischen vier Konsistenzebenen (Isolation Level)
  - Konsistenzebenen sind durch die Anomalien bestimmt, die jeweils in Kauf genommen werden
  - Abgeschwächte Konsistenzanforderungen betreffen nur Leseoperationen!
  - Lost Update muss generell vermieden werden, d. h., Write/Write-Abhängigkeiten müssen stets beachtet werden

Konsistenz-ebene	Anomalie		
	Dirty Read	Non-Repeatable Read	Phantome
Read Uncommitted	+	+	+
Read Committed	-	+	+
Repeatable Read	-	-	+
Serializable	-	-	-

Synchronisation (19)

- Konsistenzebenen (Forts.) – in SQL (Standard)
  - SQL-Anweisung

```
SET TRANSACTION [mode] [ISOLATION LEVEL level]
```

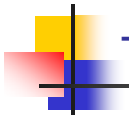
    - Transaktionsmodus
      - **READ WRITE** (Default)
      - **READ ONLY**
    - Beispiel

```
SET TRANSACTION READ ONLY,
        ISOLATION LEVEL READ COMMITTED
```

    - Ebene **READ UNCOMMITTED** und Modus **READ WRITE** sind unverträglich, da anderenfalls Schreibvorgänge auf Basis von schmutzigen Daten entstehen könnten

Synchronisation (20)

- Konsistenzebenen (Forts.) – in kommerziellen Systemen
  - Kommerzielle DBS empfehlen meist Konsistenzebene 2
  - Wahlangebot
    - Einige DBS (DB2, Tandem NonStop SQL, ...) bieten Wahlmöglichkeit zwischen
      - 'repeatable read' (Ebene 3) und
      - 'cursor stability' (Ebene 2)
    - Einige DBS bieten auch 'BROWSE'-Funktion, d. h. Lesen ohne Setzen von Sperren (Ebene 1)

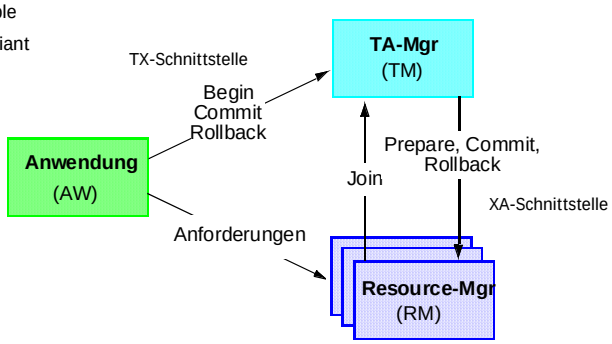


TAV in verteilten Systemen (1)

- Verallgemeinerung: Einsatz kooperierender Ressourcen-Manager (RM)
  - RM sind Systemkomponenten, die Transaktionsschutz für ihre gemeinsam nutzbaren Betriebsmittel (BM) bieten
  - RM gestatten die externe Koordination von BM-Aktualisierungen durch spezielle Commit-Protokolle
  - Gewährleistung von ACID für DB-Daten und auch für andere BM (persistente Warteschlangen, Nachrichten, Objekte von persistenten Programmiersprachen)
  - Ziel: TA-orientierte Verarbeitung in heterogenen Systemen
    - Die gesamte verteilte Verarbeitung in einer ‚Kontrollsphäre' ist eine ACID-Transaktion
      - Alle Komponenten werden durch die TA-Dienste integriert
      - Für die Kooperation ist eine Grundmenge von Protokollen erforderlich

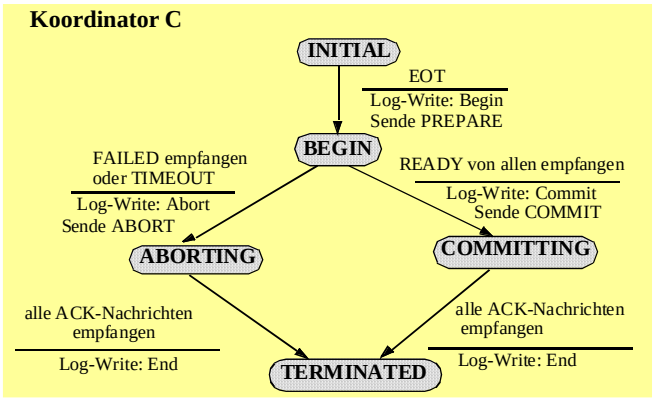
TAV in verteilten Systemen (3)

- Verteilte Transaktionsverwaltung nach X/OPEN DTP
  - unabhängige TA-Mgr
  - Resource Manager
    - recoverable
    - XA-compliant



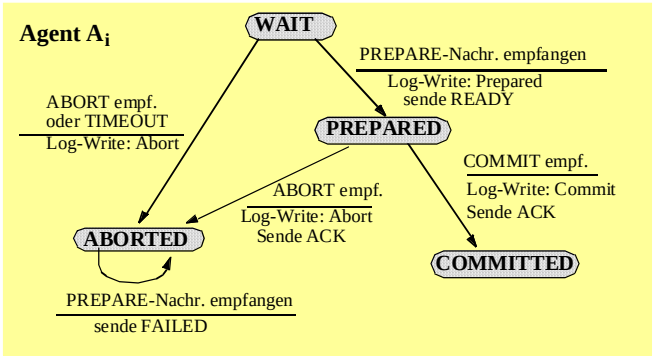
TAV in verteilten Systemen (6)

- Zweiphasen-Commit
  - Prepare-Phase, Commit/Abort-Phase
  - erfordert Folge von Zustandsübergängen, die sicher (Log) vermerkt werden müssen



TAV in verteilten Systemen (7)

- Zweiphasen-Commit (Forts.)
  - Zustandsübergänge (Forts.)



Zusammenfassung (1)

- Transaktionsparadigma (ACID)
  - Verarbeitungsklammer für die Einhaltung von semantischen Integritätsbedingungen
  - Verdeckung von (erwarteten) Fehlerfällen (*failure isolation*)
    - Logging/Recovery
  - Verdeckung der Nebenläufigkeit (*concurrency isolation*)
    - Synchronisation
  - im SQL-Standard
    - Operationen: COMMIT WORK, ROLLBACK WORK
    - Beginn einer Transaktion implizit

Zusammenfassung (2)

- Logging/Recovery
  - Transaktions-Recovery
  - Crash-Recovery
  - Medien-Recovery
  - Katastrophen-Recovery
- Synchronisation
  - Korrektheitskriterium Serialisierbarkeit
  - Sperrverfahren
  - Konsistenzebenen
- TAV in verteilten Systemen
  - Transaktionsschutz auch für verteilte Abläufe
  - X-OPEN / DTP
  - Zweiphasen-Commit-Protokoll
  - Einsatz in allen Systemen