6. Transaktionsverwaltung, Integritätssicherung und Zugriffskontrolle

Inhalt

Transaktionskonzept

Überblick Recovery und Synchronisation

Semantische Integritätsbedingungen und Trigger

Zugriffskontrolle in SQL

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)

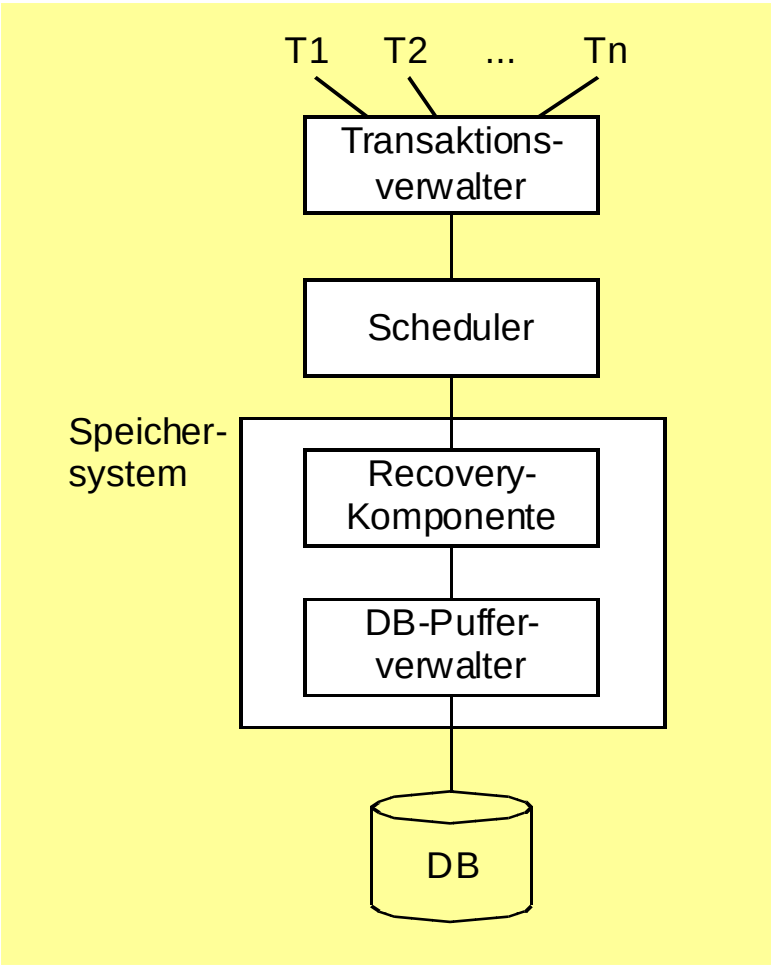
- Ablaufkontrollstruktur: Transaktion (Forts.)
  - ***ACID***-Eigenschaften von Transaktionen (Forts.)
    - **Isolation**
      - Nebenläufig (parallel, gleichzeitig) ausgeführte TA dürfen sich nicht gegenseitig beeinflussen
      - Parallele TA bzw. deren Effekte sind nicht sichtbar (logischer Einbenutzerbetrieb)
    - **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-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)

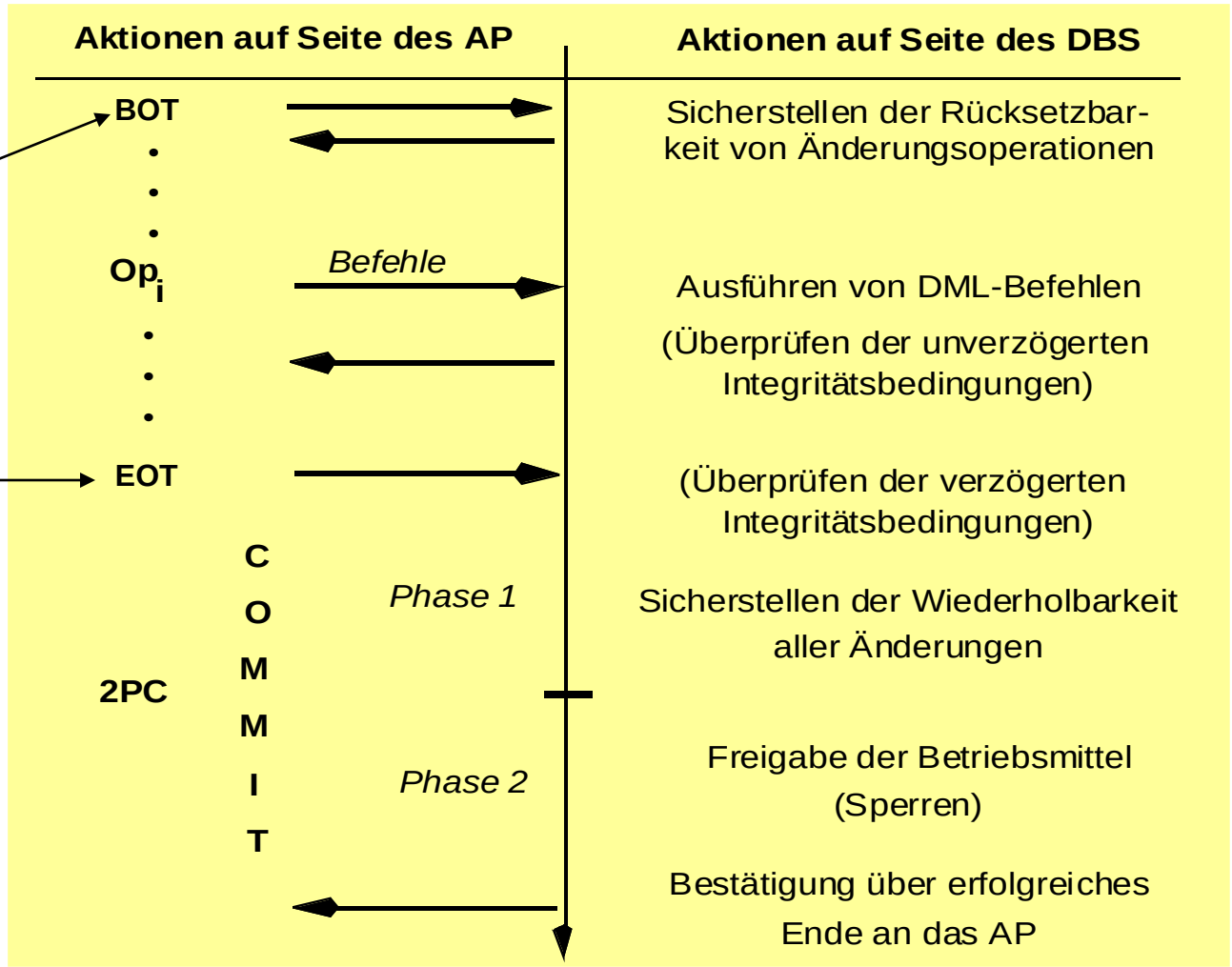


TA-Verwaltung (3)

- 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** (Synchronisation)
    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



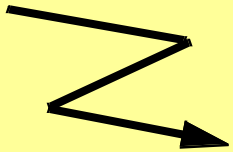
nach SQL-Standard: implizit

nach SQL-Standard:
Commit Work

‚TA-Selbstmord'
nach SQL-Standard :
Rollback Work

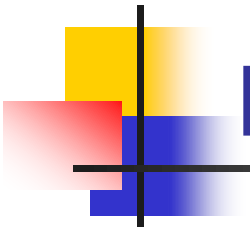
TA-Verwaltung (5)

- Transaktionsablauf (Forts.)
  - Mögliche Ausgänge einer Transaktion

BOT	**BOT**	**BOT**
DML1	**DML1**	**DML1**
DML2	**DML2**	**DML2**
DML3	**DML3**	**DML3**
• • •	• • •	 **Systemausfall, Programm-fehler usw.**
DMLn	**DMLn**	
COMMIT	**ROLLBACK**	**erzwungenes ROLLBACK**
normales Ende	**abnormales Ende**	**erzwungenes abnormales Ende**

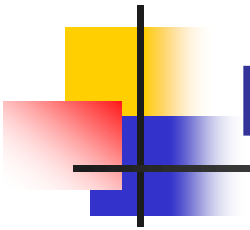
DB-Recovery (1)

- Automatische Behandlung aller ‚erwarteten' Fehler durch das DBVS
- Voraussetzung
  - Sammeln redundanter Information während des normalen Betriebs (Logging)
- Fehlermodell von zentralisierten DBVS
  - Transaktionsfehler
  - Systemfehler
  - Gerätefehler
- *„A recoverable action is 30% harder and requires 20% more code than a non-recoverable action"* (J. Gray)
- 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 (2)

- 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)

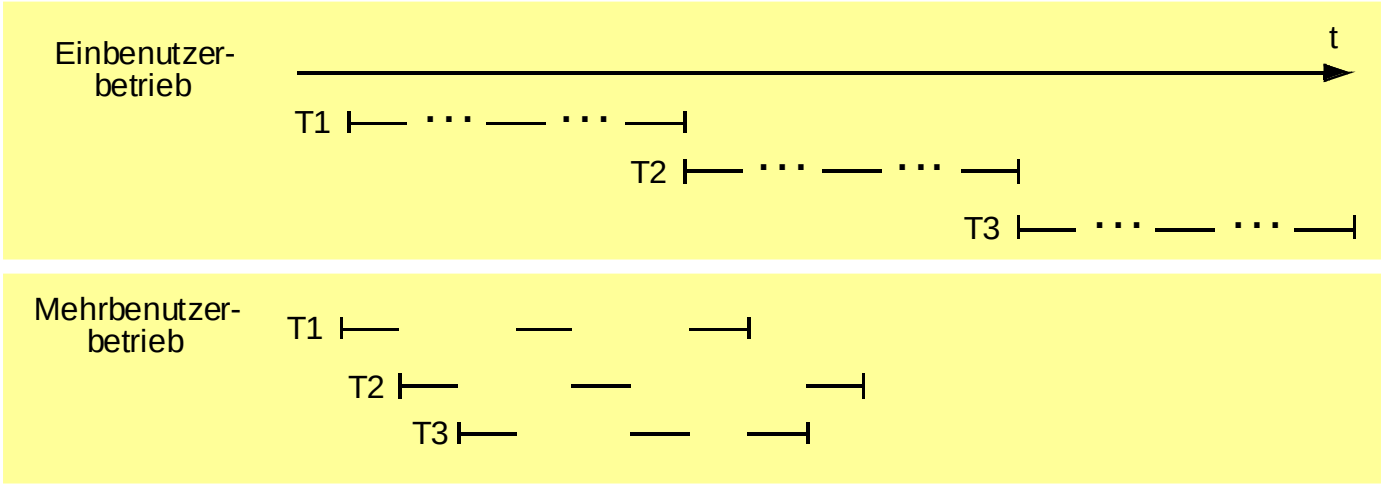


DB-Recovery (3)

- 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)

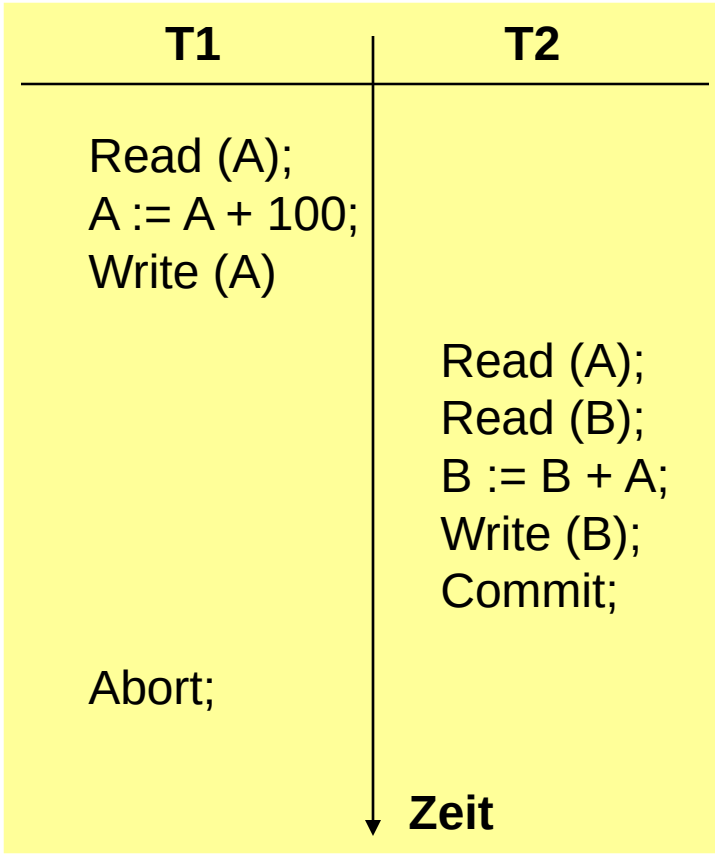
- Einbenutzer-/Mehrbenutzerbetrieb



  - 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)

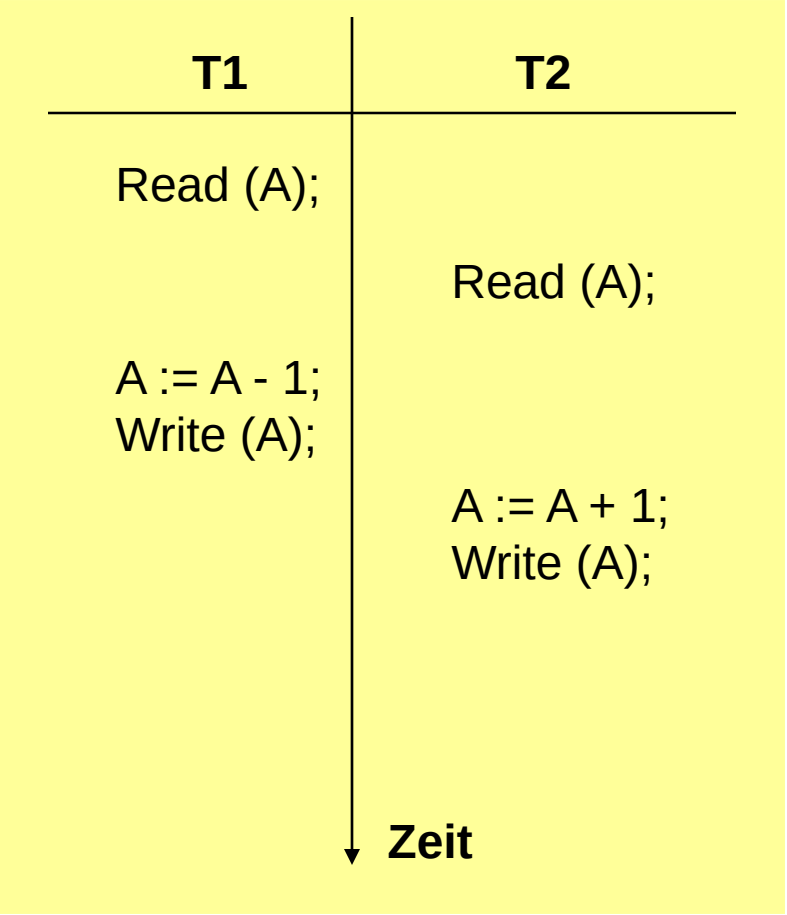
Synchronisation (2)

- 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)
		2345 39.000
SELECT Gehalt INTO :gehalt FROM Pers WHERE Pnr = 2345;		3456 48.000
summe := summe + gehalt;		
	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;`	

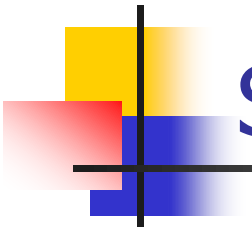
	`INSERT INTO Pers (Pnr, Anr, Gehalt)` `VALUES (4567, 17, 55.000);`

	`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

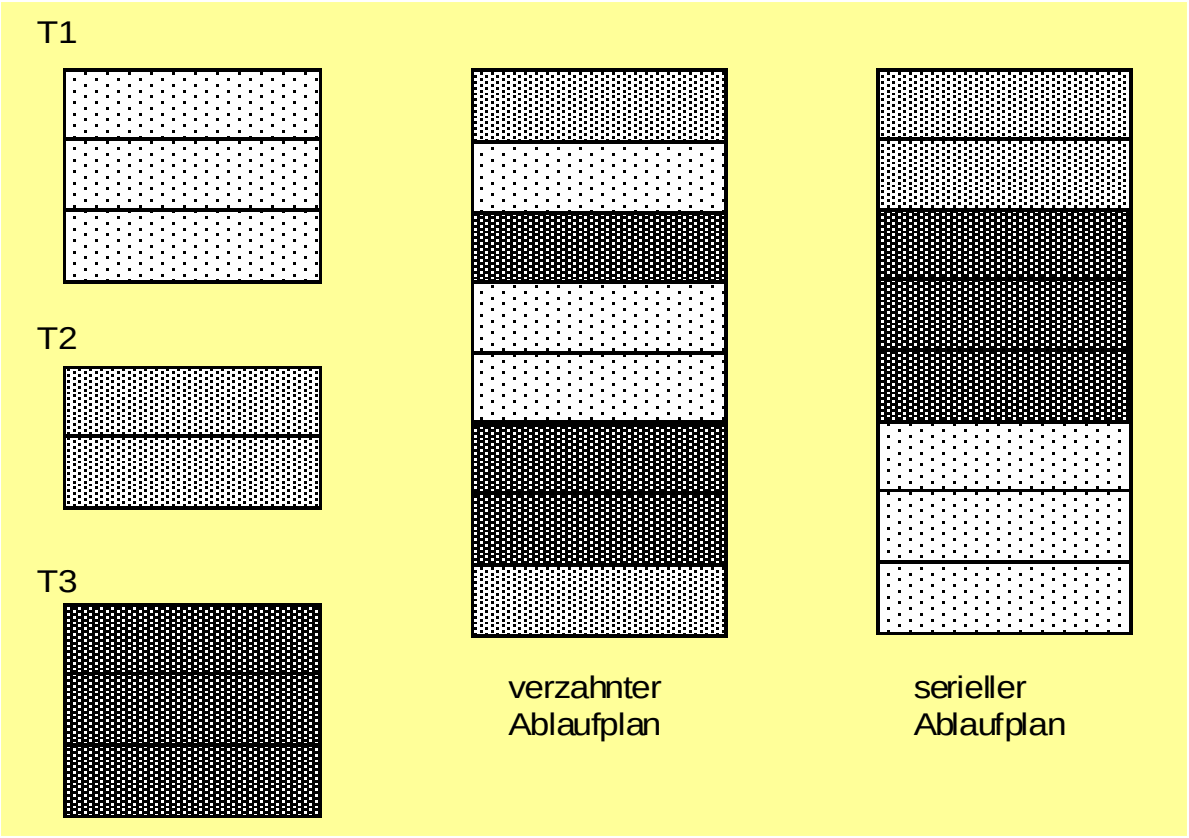


Synchronisation (6)

- 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.

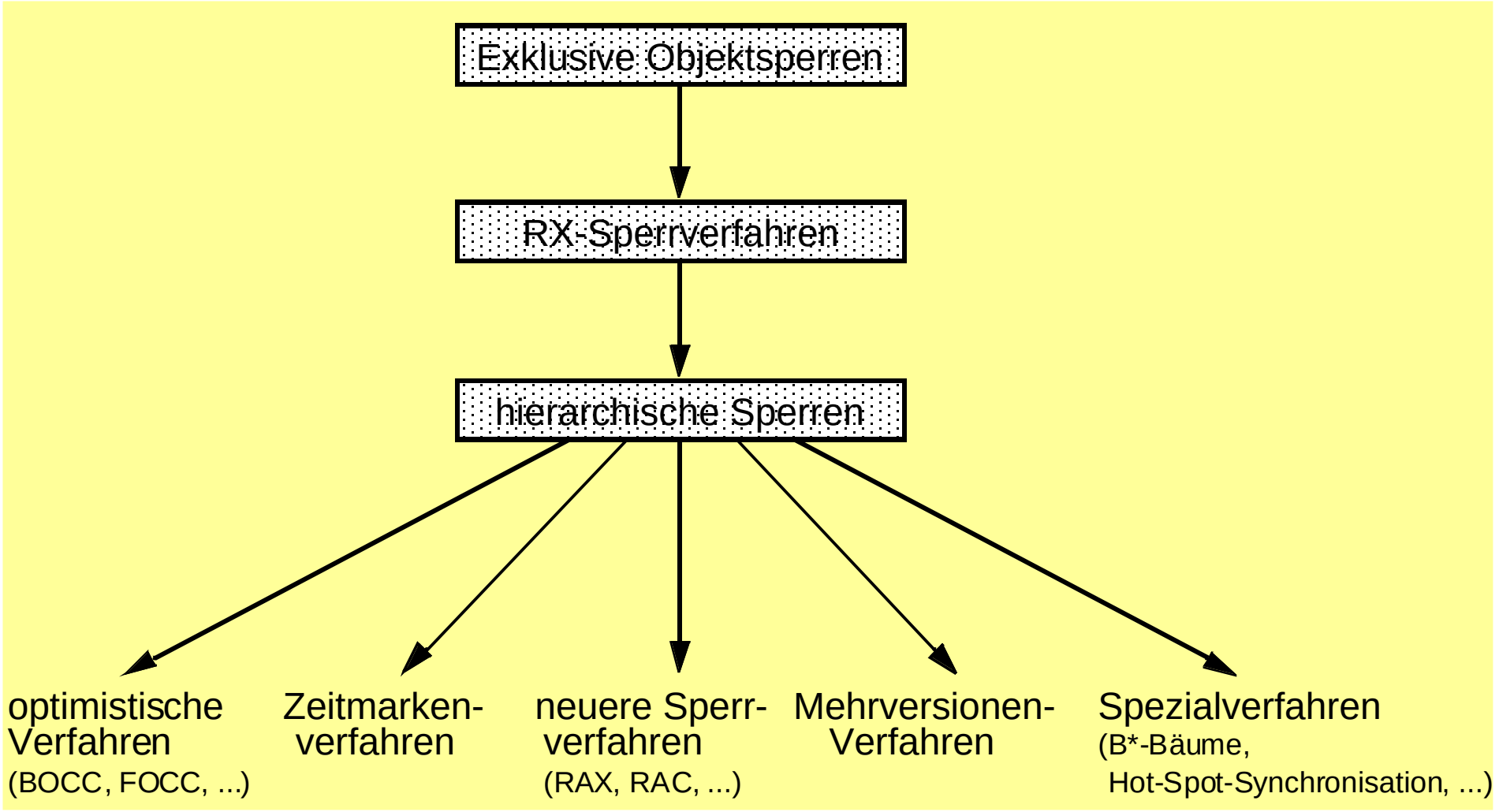
  - Hintergrund:
    - Serielle Ablaufpläne sind korrekt
    - Jeder Ablaufplan, der denselben Effekt wie ein serieller erzielt, ist akzeptierbar

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 Synchronisationsverfahren



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	

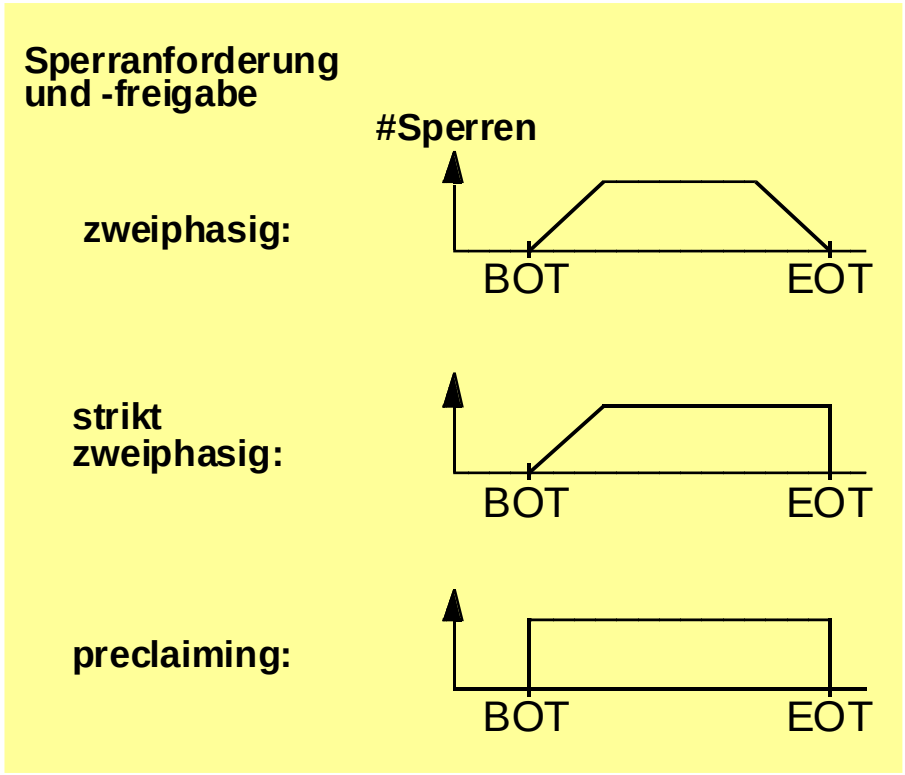
Synchronisation (13)

- 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

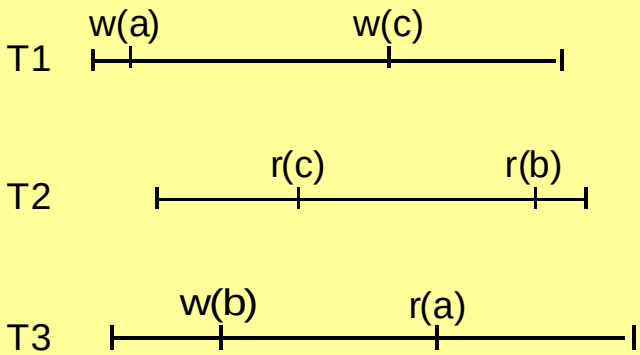
Synchronisation (14)

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

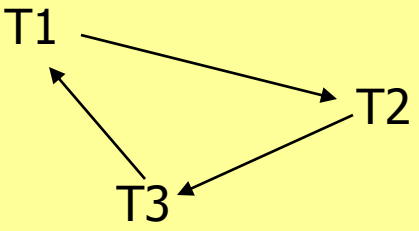


Synchronisation (15)

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



Warte-Beziehungen:



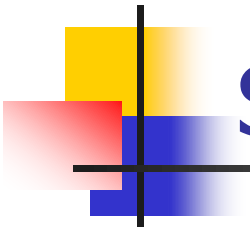
Synchronisation (16)

- 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 $\rightarrow$ X, andernfalls U $\rightarrow$ 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 (20)

- 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 (21)

- 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 (22)

- 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)

Semantische IBs (1)

- Klassifikation: Unterscheidung nach
  - Ebenen der Abbildungshierarchie eines DBS (Blöcke, Seiten, Tupel, ...)
  - Reichweite (Attribut, Relation, mehrere Relationen)
  - Zeitpunkt der Überprüfbarkeit (sofort, erst nach mehreren Operationen)
  - Art der Überprüfbarkeit (Zustand, Übergang)
  - Anlass für Überprüfung (Datenänderung, Zeitpunkt)
- Konsistenz der Transaktionsverarbeitung
  - Bei COMMIT müssen alle semantischen Integritätsbedingungen erfüllt sein.
  - Zentrale Spezifikation/Überwachung im DBS: *„system enforced integrity*"

Semantische IBs (2)

- Reichweite
  - **Art und Anzahl** der von einer Integritätsbedingung (genauer: des die Bedingung ausdrückenden Prädikats) betroffenen **Objekte**
    - **ein Attribut** (Bsp.: PNR vierstellige Zahl, NAME nur Buchstaben und Leerzeichen)
    - **mehrere Attribute eines Tupels** (Bsp.: GEHALTS-SUMME einer Abteilung muss kleiner sein als JAHRES-ETAT)
    - **mehrere Tupel derselben Relation** (Bsp.: kein GEHALT mehr als 20 % über dem Gehaltsdurchschnitt aller Angestellten derselben Abteilung, PNR ist Primärschlüssel)
    - **mehrere Tupel aus verschiedenen Relationen** (Bsp.: GEHALTS-SUMME einer Abteilung muss gleich der Summe der Attributwerte in GEHALT der zugeordneten Angestellten sein)
  - **geringere Reichweite = einfachere Überprüfung**

Semantische IBs (3)

- Zeitpunkt der Überprüfbarkeit
  - Unverzögerte Bedingungen
    - müssen immer erfüllt sein
    - können sofort nach Auftauchen des Objektes überprüft werden (typisch: solche, die sich auf ein Attribut beziehen)
  - Verzögerte Bedingungen
    - z.B. zyklische Fremdschlüsselbedingungen
    - lassen sich nur durch eine Folge von Änderungen erfüllen (typisch: mehrere Tupel, mehrere Relationen)
    - benötigen Transaktionsschutz (als zusammengehörige Änderungssequenzen)

Semantische IBs (4)

- Art der Überprüfbarkeit
  - Zustandsbedingungen
    - betreffen den zu einem bestimmten Zeitpunkt in der DB abgebildeten Objektzustand
  - Übergangsbedingungen
    - Einschränkungen der Art und Richtung von Wertänderungen einzelner oder mehrerer Attribute
    - Beispiele: GEHALT eines Angestellten darf niemals sinken, FAM-STAND darf nicht von „ledig" nach „geschieden" oder von „verheiratet" nach „ledig" geändert werden
    - sind am Zustand nicht prüfbar - entweder sofort bei Änderung oder später durch Vergleich von altem und neuem Wert (Versionen)

Semantische IBs (5)

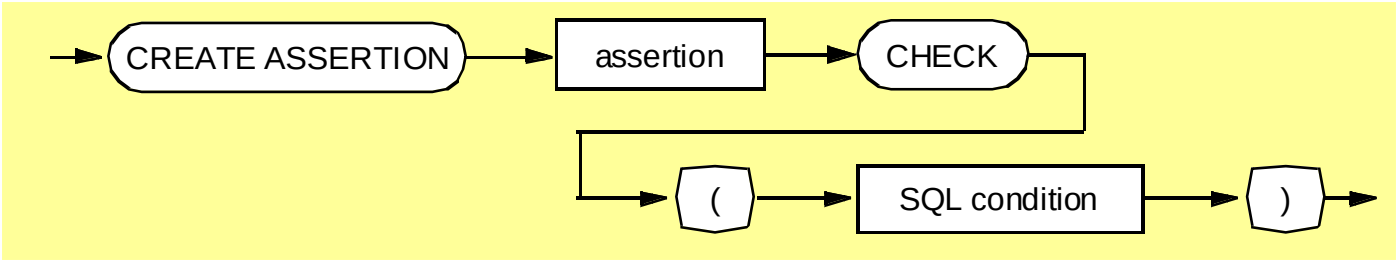
- Anlass für Überprüfung
  - Änderungsvorgang in der DB
    - alle bisherigen Beispiele implizieren Überprüfung innerhalb der TA
  - „Verspätete" Überprüfung: Änderung zunächst nur in (mobiler) Client-DB
  - Ablauf der äußeren Zeit
    - z. B. Daten über produzierte und zugelassene Fahrzeuge – Fahrzeug muss spätestens ein Jahr nach Herstellung angemeldet sein
    - nicht trivial: was ist zu tun bei Verletzung?
      kann an der Realität liegen – abstrakte Konsistenzbedingung erfüllen oder (inkonsistente) Realität getreu abbilden?

Semantische IBs (6)

- Integritätsbedingungen in SQL
  - Bereits eingeführt (siehe Datendefinition)
    - CHECK-Bedingungen bei CREATE DOMAIN, CREATE TABLE, Attributdefinition
    - UNIQUE, PRIMARY KEY, Verbot von Nullwerten
    - Fremdschlüsselbedingungen (FOREIGN-KEY-Klausel)
  - Die vorgenannten Integritätsbedingungen sind an DB-Schemaelemente gebunden
  - **Allgemeine Integritätsbedingungen**
    - beziehen sich typischerweise auf mehrere Relationen
    - lassen sich als eigenständige DB-Objekte definieren
    - erlauben die Verschiebung ihres Überprüfungszeitpunktes
    - **Assertion**

Semantische IBs (7)

- Integritätsbedingungen in SQL (Forts.)
  - Assertion-Anweisung

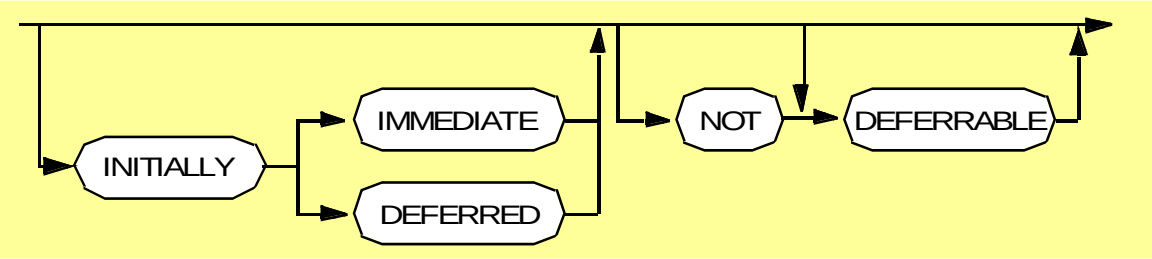


  - Beispiel: Die Relation Abt enthält ein Attribut, in dem (redundant) die Anzahl der Angestellten einer Abteilung geführt wird. Es gilt folgende Zusicherung:

```
CREATE ASSERTION A1
        CHECK (NOT EXISTS
                        (SELECT * FROM Abt A
                         WHERE A.Anzahl_Angest <>
                                (SELECT  COUNT (*) FROM  Pers P
                                 WHERE P.Anr = A.Anr)));
```

Semantische IBs (8)

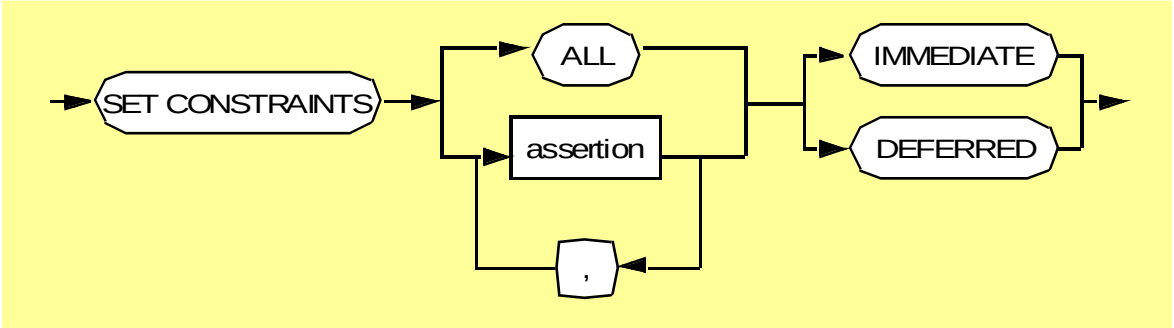
- Integritätsbedingungen in SQL (Forts.)
  - Festlegung des Überprüfungszeitpunktes



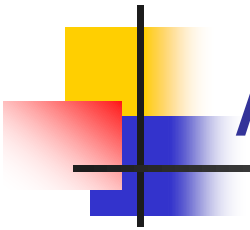
    - IMMEDIATE: am Ende der Änderungsoperation (Default)
    - DEFERRED: am Transaktionsende (COMMIT)

Semantische IBs (9)

- Integritätsbedingungen in SQL (Forts.)
  - **Überprüfung** kann durch **Constraint-Modus gesteuert** werden

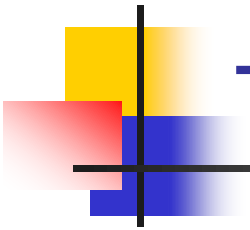


    - Zuordnung gilt für die aktuelle Transaktion
    - Bei benamten Constraints ist eine selektive Steuerung der Überprüfung möglich; so können ‚gezielt' Zeitpunkte vor COMMIT ausgewählt werden.



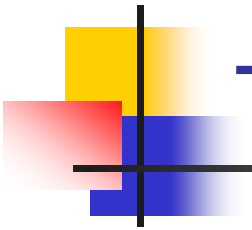
Aktives Verhalten (1)

- Bisher
  - Integritätsbedingungen beschreiben, was innerhalb der DB gültig und zulässig ist.
- Neue Idee
  - Spezifikation und Durchführung von Reaktionen bestimmte Situationen oder Ereignisse in der DB
  - „Zusammenhangsregel" (kausale, logische oder „beliebige" Verknüpfung) statt statischem Prädikat
  - Je mehr Semantik des modellierten Systems explizit repräsentiert ist, umso mehr kann das DBS „aktiv" werden!
- Oft synonyme Nutzung der Begriffe *Produktionsregel, Regel, Aktive Regel, Trigger, Alerter*



Trigger (1)

- Einsatz und Standardisierung
  - Trigger werden schon seit ~1985 in relationalen DBS eingesetzt
  - Ihre Standardisierung wurde jedoch erst in SQL:1999 vorgenommen
- Konzept nach SQL:1999
  - Wann soll ein Trigger ausgelöst werden?
    - Zeitpunkte: BEFORE / AFTER
    - auslösende Operation: INSERT / DELETE / UPDATE
  - Wie spezifiziert man (bei Übergangsbedingungen) Aktionen?
    - Bezug auf verschiedene DB-Zustände erforderlich
    - OLD/NEW erlaubt Referenz von alten/neuen Werten

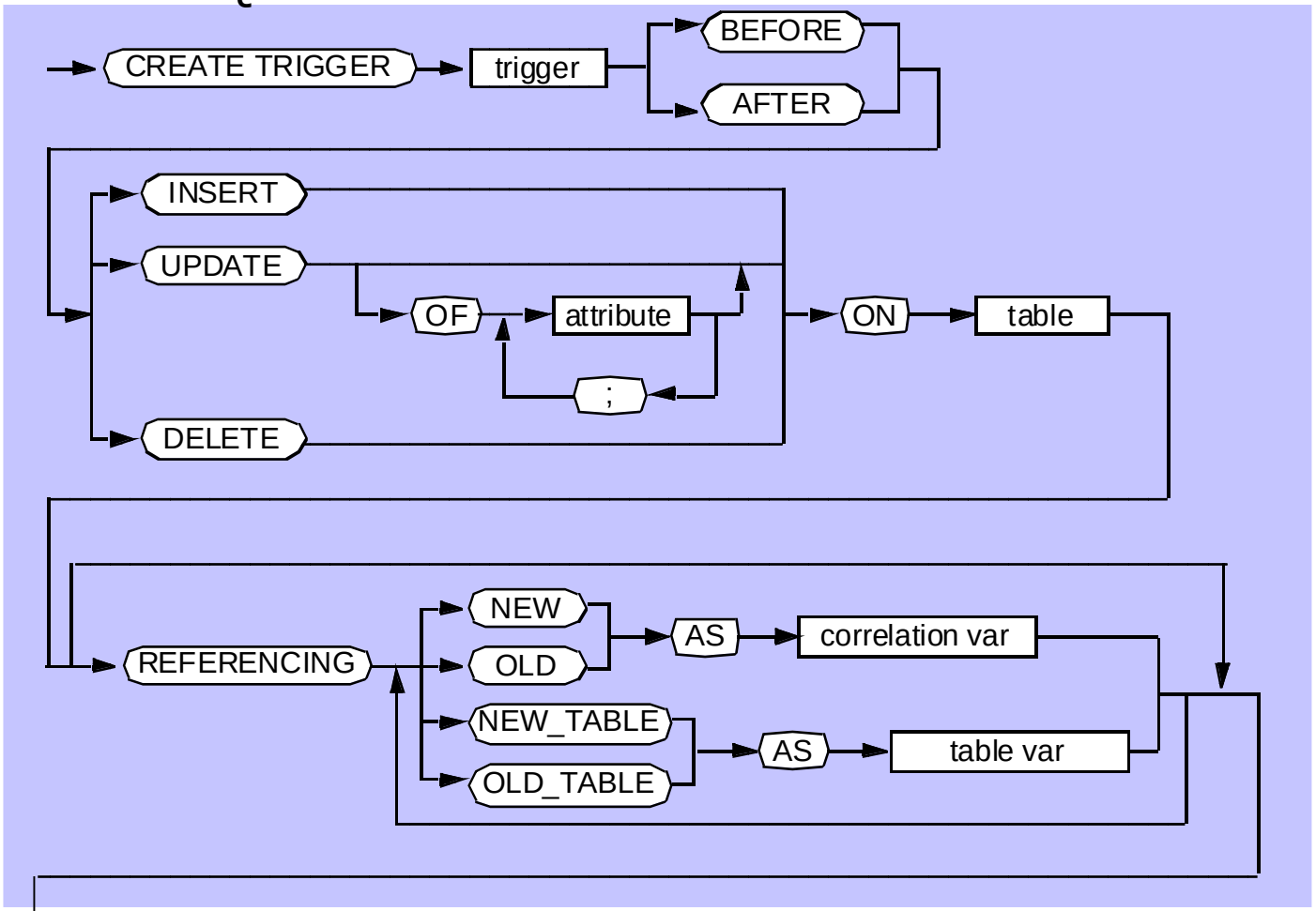


Trigger (2)

- Konzept nach SQL:1999 (Forts.)
  - Ist die Trigger-Ausführung vom DB-Zustand abhängig?
    - WHEN-Bedingung optional
  - Was soll wie verändert werden?
    - pro Tupel oder pro DB-Operation (Trigger-Granulat)
    - mit einer SQL-Anweisung oder mit einer Prozedur aus PSM-Anweisungen (persistent stored module, stored procedure)
  - Existiert das Problem der Terminierung und der Auswertungsreihenfolge?
    - mehrere Trigger-Definitionen pro Relation (Tabelle) sowie
    - mehrere Trigger-Auslösungen pro Ereignis möglich

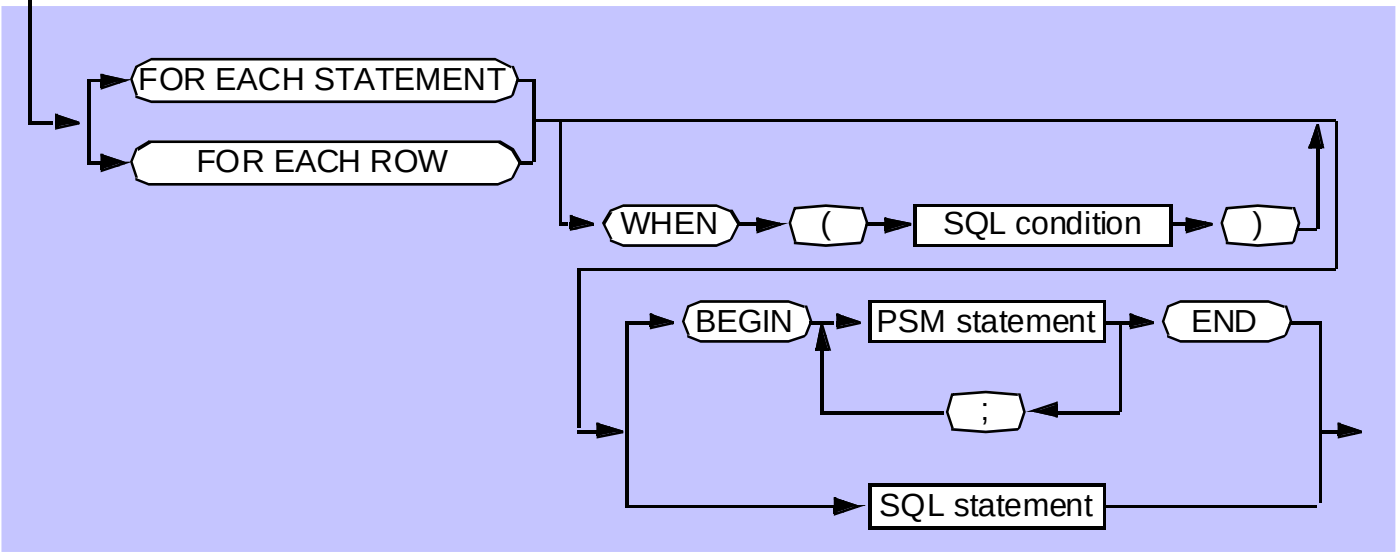
Trigger (3)

- Syntax nach SQL:1999



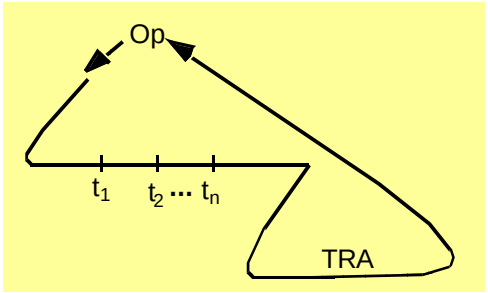
Trigger (4)

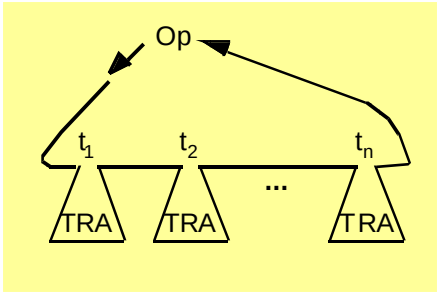
- Syntax nach SQL:1999 (Forts.)



Trigger (5)

- Übergangstabellen und -variablen
  - sie vermerken Einfügungen (bei INSERT), Löschungen (bei DELETE) und die alten und neuen Zustände (bei UPDATE).
  - Übergangstabellen (transition tables) beziehen sich auf mengenorientierte Änderungen
  - Übergangsvariablen (transition variables) beziehen sich auf tupel-weise Änderungen
- Trigger-Granulat
  - FOR EACH STATEMENT: mengenorientiertes Verarbeitungsmodell
  - FOR EACH ROW: tupelorientiertes Verarbeitungsmodell
  - TRA: Trigger-Aktion





Trigger (6)

- Einsatzbeispiel
  - Gehaltsumme in Abt soll bei Änderungen in Pers, die „Gehälter" betreffen, automatisch aktualisiert werden
  - es sind Trigger für INSERT/DELETE/UPDATE erforderlich; sie werden bei Auftreten der spezifizierten Änderungsoperationen sofort ausgeführt

Abt	Anr	Aname	Ort	Geh_Summe
	K51	PLANUNG	KAISERSLAUTERN	43500
	K53	EINKAUF	FRANKFURT	45200
	K55	VERTRIEB	FRANKFURT	80000

Pers	Pnr	Name	Alter	Gehalt	Anr	Mnr
	406	COY	47	50 000	K55	123
	123	MÜLLER	32	43 500	K51	-
	829	SCHMID	36	45 200	K53	777
	574	ABEL	28	30 000	K55	123

Trigger (7)

- Einsatzbeispiel (Forts.)

```
CREATE TRIGGER T1
AFTER INSERT ON Pers                          (* Ereignis *)
REFERENCING NEW AS NP
FOR EACH ROW
        UPDATE Abt A                          (* Aktion *)
        SET A.Geh_Summe = A.Geh_Summe + NP.Gehalt
        WHERE A.Anr = NP.Anr;
```

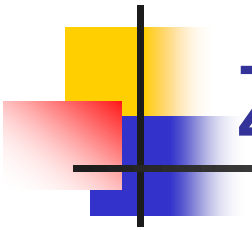
```
CREATE TRIGGER T2
AFTER UPDATE OF Gehalt ON Pers                (* Ereignis *)
REFERENCING OLD AS OP NEW AS NP
FOR EACH ROW
        UPDATE Abt A                          (* Aktion *)
        SET A.Geh_Summe = A.Geh_Summe + (NP.Gehalt - OP.Gehalt)
        WHERE A.Anr = NP.Anr;
```

Trigger (8)

- Einsatzbeispiel (Forts.)

```
CREATE TRIGGER T3
AFTER  UPDATE OF Gehalt ON Pers            (* Ereignis *)
REFERENCING OLD_TABLE AS OT NEW_TABLE AS NT
FOR EACH STATEMENT
        UPDATE Abt A                        (* Aktion *)
        SET A.Geh_Summe = A.Geh_Summe +
                (SELECT SUM (Gehalt) FROM NT WHERE Anr = A.Anr) -
                (SELECT SUM (Gehalt) FROM OT WHERE Anr = A.Anr)
        WHERE A.Anr IN (SELECT Anr FROM NT);
```

Zugriffskontrolle - Allgemeines (1)



- Zugriffskontrolle: technische Maßnahme des Datenschutzes
- Kernfrage: Wie kann ich erreichen, dass Benutzer mit unterschiedlichen Rechten gemeinsam auf Daten zugreifen können?
  - Frage nach der Zugriffskontrolle (bei Daten)
- Zugriffskontrolle (Autorisierung)
  - Vergabe von Zugriffsrechten (Lesen, Schreiben, . . .) auf DB-Objekten, Programmen usw.
  - Ziele
    - Verhinderung von zufälligen oder böswilligen Änderungen
    - möglichst weitgehende Isolation von Programmfehlern
    - Verhinderung von unberechtigtem Lesen/Kopieren

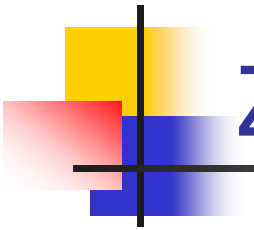
Zugriffskontrolle - Allgemeines (2)

- Autorisierungsmodell
  - Explizite Autorisierung:
    - Dieses Modell wird im Englischen als Discretionary Access Control (DAC) bezeichnet. Wegen seiner Einfachheit ist DAC weit verbreitet („discretionary" bedeutet in etwa „nach dem Ermessen des Subjekts").
    - Der Zugriff auf ein Objekt o kann nur erfolgen, wenn für den Benutzer (Subjekt s) ein Zugriffsrecht (Privileg p) vorliegt
    - Autorisierungsregel (o, s, p)
  - Schutzinformation als Zugriffsmatrix
    - *Subjekte:* Benutzer, Programme, Terminals
    - *Objekte:* Programme (Anwendungs-, Dienstprogramme), DB-Objekte (Relationen, Sichten, Attribute)
    - *Zugriffsrechte:* Lesen, Ändern, Ausführen, Erzeugen, Weitergabe von Zugriffsrechten usw., ggf. abhängig von Terminal, Uhrzeit usw.

Zugriffskontrolle - Allgemeines (3)

- Autorisierungsmodell (Forts.)
  - Autorisierung
    - zentrale Vergabe der Zugriffsrechte (DBA)
    - dezentrale Vergabe der Zugriffsrechte durch Eigentümer der Objekte
  - Objektgranulat
    - wertunabhängige oder
    - wertabhängige Objektfestlegung (Sichtkonzept)
  - Wirksamkeit der Zugriffskontrolle beruht auf drei Annahmen:
    - fehlerfreie Benutzer-Identifikation/-Authentisierung
    - erfolgreiche Abwehr von (unerwarteten) Eindringlingen (vor allem strikte Isolation der Benutzer- und DBS-Prozesse sowie Übermittlungskontrolle)
    - Schutzinformation ist hochgradig geschützt!

Zugriffskontrolle in SQL (1)

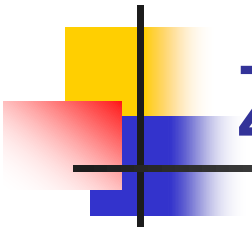


- WICHTIG: Sicht-Konzept erlaubt wertabhängigen Zugriffsschutz
- Vergabe von Rechten

```
GRANT    {privileges-commalist | ALL PRIVILEGES}
         ON accessible-object TO grantee-commalist
         [WITH GRANT OPTION]
```

  - Objekte (accessible-object)
    - Relationen bzw. Sichten
    - aber auch: Domänen, Datentypen, Routinen usw.

Zugriffskontrolle in SQL (2)



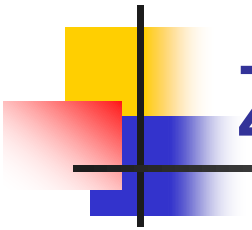
- Vergabe von Rechten (Forts.)
  - Zugriffsrechte (privileges)
    - SELECT, INSERT, UPDATE, DELETE, REFERENCES, USAGE, EXECUTE, . . .
    - Attributeinschränkung bei INSERT, UPDATE und REFERENCES möglich
    - Erzeugung einer „abhängigen" Relation erfordert REFERENCES-Recht auf von Fremdschlüsseln referenzierten Relationen.
    - USAGE erlaubt Nutzung spezieller Wertebereiche (character sets).
    - dynamische Weitergabe von Zugriffsrechten: WITH GRANT OPTION (GO: dezentrale Autorisierung)
  - Empfänger (grantee)
    - Liste von Benutzern bzw. PUBLIC
    - Liste von Rollennamen

Zugriffskontrolle in SQL (3)

- Vergabe von Rechten (Forts.)
  - Beispiele
    - **GRANT SELECT ON** Abt **TO PUBLIC**
    - **GRANT INSERT, DELETE ON** Abt
      TO Mueller, Weber **WITH GRANT OPTION**
    - **GRANT UPDATE** (Gehalt) **ON** Pers **TO** Schulz
    - **GRANT REFERENCES** (Pronr) **ON** Projekt **TO PUBLIC**
- Rücknahme von Zugriffsrechten

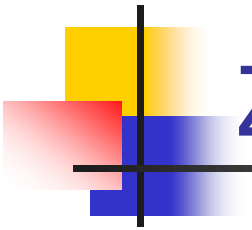
```
REVOKE   [GRANT OPTION FOR]  privileges-commalist
         ON  accessible-object FROM grantee-commalist
         {RESTRICT | CASCADE}
```

  - Beispiel: **REVOKE SELECT ON** Abt **FROM** Weber **CASCADE**



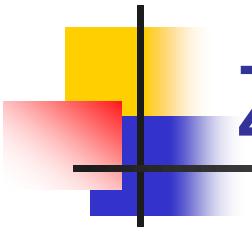
Zusammenfassung (1)

- 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



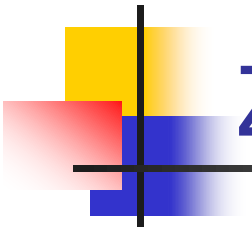
Zusammenfassung (2)

- Logging/Recovery
  - Transaktions-Recovery
  - Crash-Recovery
  - Medien-Recovery
  - Katastrophen-Recovery
- Synchronisation
  - Korrektheitskriterium Serialisierbarkeit
  - Sperrverfahren
  - Konsistenzebenen



Zusammenfassung (3)

- Semantische Integritätskontrolle
  - Relationale Invarianten, referentielle Integrität und Aktionen
  - Benutzerdefinierte Integritätsbedingungen (*assertions*)
    - zentrale Spezifikation/Überwachung im DBS wird immer wichtiger
- Aktives DB-Verhalten zur
  - Integritätssicherung
  - Wartung abgeleiteter Daten
  - Durchführung allgemeiner Aufgaben (Regeln, Alerter, Trigger)
- Triggerkonzept in SQL99 standardisiert



Zusammenfassung (4)

- Zugriffskontrolle in DBS
  - wertabhängige Festlegung der Objekte (Sichtkonzept)
  - Vielfalt an Rechten erwünscht
  - zentrale vs. dezentrale Rechtevergabe
  - Rollenkonzept: vereinfachte Verwaltung komplexer Mengen von Zugriffsrechten