9. Transaktionen, Datenschutz, Datensicherheit

Transaktionen:

- Folge von DB-Operationen, die bzgl. Integritätsüberwachung als eine Einheit angesehen werden

- DB muß nur vor und nach Transaktionsausführung in konsistentem Zustand sein

- müssen ACID-Eigenschaften erfüllen:
  - A: Atomarität; Transaktion wird ganz oder gar nicht ausgeführt
  - C: Konsistenz; DB muß nach Transaktion in zulässigem Zustand sein
  - I: Isolation; evtl. parallel ablaufende Transaktionen sind isoliert, können sich nicht gegenseitig beeinflussen
  - D: Dauerhaftigkeit; Wirkung einer erfolgreich beendeten Transaktion ist dauerhaft

Transaktionen

- Beispiel:

  Kontobuchung eines Betrags s von Kontonr. x nach Kontonr. y:
  IB: Summe der Kontostände vor und nach Umbuchung muß unverändert sein;
  während der Ausführung: nach Abbuchung von s von x ist IB nicht erfüllt, erst wieder nach Zubuchung auf y -> während laufender Transaktion kann IB nicht erfüllt sein

- in der Praxis:
  - Mehrbenutzerbetrieb
  - nebenläufige, konkurrierende Prozesse
  - -> atomarer, völlig isolierter Ablauf von Transaktionen nicht möglich
  - Hardwarefehler, Laufzeitfehler,...

- Forderung: Ergebnis einer Transaktion soll so aussehen, als sei sie nach dem ACID-Prinzip abgelaufen

Transaktionen

Abhilfe: Sperren (Locks)

- von Datenelementen:
  - ganze Relation
  - einzelne Tupel
  - einzelne Attributwerte
  - einzelne Seiten
- je größer Datenelement, umso geringer Verwaltungsaufwand, aber auch umso geringer Parallelität
- lock: Transaktion sperrt Datenelement vor Bearbeitung; ist Datenelement schon (von anderer Transaktion) gesperrt: warten, bis Sperre aufgehoben, dann sperren
- unlock: Entsperren des Datenelements nach Bearbeitung

Transaktionen

Probleme: Verklemmungen

- Behinderungen: eine Transaktion versucht, ein Datenelement zu sperren; dieses ist ständig durch andere Transaktionen abwechselnd gesperrt

- Deadlock:

  T1: lock A; ...; lock B;...;unlock A;...unlock B;
  T2: lock B; ...; lock A;...;unlock B;...unlock A;

- DBMS muß Verklemmungen vorbeugen, erkennen, behandeln
  - z.B. Behinderungen durch Warteschlangen, in die sich Transaktionen einreihen müssen
  - z.B. durch Forderung, daß alle Locks einer Transaktions gleichzeitig geschehen müssen
  - z.B. durch lineare Ordnung auf Datenelementen (lock A immer vor lock B)
  - z.B. bei Deadlocks durch Stoppen von Transaktionen, später erneutes Anlaufen

Transaktionen

Serialisierbarkeit:

- Ausführung konkurrierender Transaktionen ist **korrekt**, wenn Ergebnis dem einer seriellen Abarbeitung der Transaktionen (in bel. Reihenfolge) entspricht

- Schedule: Plan, der eine geordnete Folge von lock-, read-, write- und unlock-Aktionen für eine Menge von Transaktionen enthält

- serieller Schedule: alle Aktionen jeder Transaktion folgen unmittelbar aufeinander

- Schedule serialisierbar: Ergebnis eines Schedules ist äquivalent zu dem eines seriellen Schedules

- Beispiel:

S1:		S2:		S3	
T1	T2	T1	T2	T1	T2
read A		read A		read A	
A=A-10			read B	A=A-10	
write A		A=A-10			read B
read B			B=B-20	write A	
B=B+10		write A			B=B-20
write B			write B	read B	
	read B	read B			write B
	B=B-20		read C	B=B+10	
	write B	B=B+10			read C
	read C		C=C+20	write B	
	C=C+20	write B			C=C+20
	write C		write C		write C

S1: seriell, S2: nicht seriell aber serialisierbar, S3: nicht serialisierbar

Transaktionen

verschiedene Transaktionsmodelle: welche garantieren Serialisierbarkeit?

- einfaches lock-unlock-Modell:
  - Sperren setzen:
    lock A - mit implizitem read A
    unlock A - mit implizitem write A
    dazwischen Funktion(en), die den Wert von A verändern
  - Test auf Serialisierbarkeit: mittels Vorgängergraph
    Algorithmus: gegeben T1,..., Tn mit Schedule; bilde gerichteten Graphen, dessen Knoten den Ti entsprechen; suche Paare im Schedule der Art: Ti: unlock A.....Tj: lock A (nächstes nach unlock); jedes solche Paar bildet Kante von Ti nach Tj;
    hat Graph schließlich keinen Kreis: Schedule ist serialisierbar
  - Beispiel:

T1:	T2:	T3
lock A		
	lock B	
	lock C	
	unlock B	
lock B		
unlock A		
	lock A	
	unlock C	
	unlock A	
		lock A
		lock C
unlock B		
		unlock C
		unlock A

Transaktionen

Transaktionsmodelle

- 2-Phasen-Sperr-Protokoll:
  - alle Sperren (1. Phase) kommen vor allen Freigaben (2. Phase)
  - erfüllen alle Transaktionen jeweils obige Bedingung: Schedule immer serialiserbar
  - Deadlock-Problem aber damit nicht gelöst
- read-write-Modell: Unterscheidung zwischen read-only- und read-write-Zugriff
- read-only-write-only-Modell: berücksichtigt auch write-only-Zugriff
- optimistische Methoden: sperren keine Datenelemente; lassen bei jeder Lese-/Schreiboperation prüfen, ob serielle Ordnung verletzt ist

Transaktionen

Transaktionsabbruch:

- bisher: Annahme alle Transaktionen beenden Arbeit
- Problem: Transaktion kann vorher sterben (Deadlock, ^C, Division durch 0...)
- Unterscheidung:
  - laufende Transaktion (vor COMMIT): kann noch abgebrochen werden, Änderungen nicht dauerhaft gespeichert
  - abgeschlossene Transaktion (nach COMMIT): darf wegen Eigenschaft D nicht zurückgesetzt werden
  - -> kritischer Befehl ist COMMIT
- Beispiel:

	T1:	T2:
1	lock A	
2	read A	
3	A=A-1	
4	write A	
5	lock B	
6	unlock A	
7		lock A
8		read A
9		A=A*2
10	read B	
11		write A
12		COMMIT
13		unlock A
14	B=B/A (A sei 0)	

Transaktionen

Transaktionsabbruch:

- Was müßte passieren?
  - lock B (T1) lösen
  - neuen Wert von A (Zeile 3,4; T1) durch alten Wert ersetzen
  - T2 hat dann aber falschen Wert gelesen -> T2 insgesamt zurücksetzten
  - falls zwischen Zeile 13,14 andere Transaktionen A gelesen hätten: diese auch zurücksetzen (cascading rollback)
- aber: cascading rollback unerwünscht (widerspricht Transaktionsdefinition)
- -> strenges 2-Phasen-Sperr-Protokoll:
  - Transaktion darf nicht in DB schreiben, bevor sie COMMIT erreicht hat
  - vor COMMIT werden keine unlocks ausgelöst
  - -> (neuer) Wert in DB kann nur von COMMITed Transaktion geschrieben werden, Transaktion ist sicher dauerhaft

Transaktionen

Transaktionen in Oracle (in sqlplus, PL/SQL):

i.A.: Oracle sperrt alle benötigten Daten automatisch; explizite Angaben:

- `COMMIT [WORK]`: dauerhafte Speicherung der bisherigen Änderungen, Lösen aller Sperren, Löschen alles Savepoints der Transaktion, Start einer neuen Transaktion;
  vor u. nach jedem DDL-Befehl: automatisches `COMMIT`

- `ROLLBACK [WORK]`: Rücksetzen aller Änderungen seit dem letzten Transaktionsbeginn, Beenden der Transaktion, Löschen aller Savepoints der Transaktion, Lösen aller Sperren;

- `ROLLBACK [WORK] TO [SAVEPOINT] spname`: Rollback der aktuellen Transaktion bis zum Savepoint spname, Löschen aller Savepoints nach spname, Lösen aller Sperren seit dem Savepoint spname

- `SAVEPOINT spname`: Setzen eines Savepoints namens spname

- `SET TRANSACTION READ ONLY | READ WRITE | ISOLATION LEVEL SERIALIZABLE | ISOLATION LEVEL READ COMMITTED | USE ROLLBACK SEGMENT rb_seg_name`:
  setzt den Modus der aktuellen Transaktion (gilt nur bis zum nächsten `COMMIT`, `ROLLBACK`, `DDL`-Kommando), Voreinstellung: `READ WRITE`

- `LOCK TABLE relname1 [,relname2,..] IN modus MODE [NOWAIT]`:
  Sperren von Relation(en) oder View(s); überschreibt

automatische Sperren; erlaubte Modi:
ROW SHARE, SHARE UPDATE: erlaubt gleichzeitigen lesenden Zugriff, verbietet Updates; kein anderer Benutzer kann EXCLUSIVE-Modus auf Relation setzen
ROW EXCLUSIVE: erlaubt gleichzeitigen lesenden Zugriff, verhindert SHARE-Sperren durch andere; automatisch gewählter Modus bei UPDATE, INSERT, DELETE
SHARE: erlaubt gleichzeitigen lesenden Zugriff, verbietet Updates; kann von verschiedenen Benutzern auf gleicher Relation gegeben werden
EXCLUSIVE:erlaubt lesenden Zugriff, verbietet jede andere Aktivität anderer
-> keine Relation kann gegen Lesen anderer gesperrt werden!

- SELECT ...FOR UPDATE (in PL/SQL): expl. Sperren einer Relation/von Tupeln, Sperre gilt bis COMMT/ROLLBACK

Transaktionen im Embedded SQL: wie oben mit Präfix EXEC SQL

Datenschutz, Datensicherheit

Datenschutz: Schutz der Daten gegen unbefugtes Lesen, Modifizieren, Löschen

div. Techniken:

- Benutzer-Identifikation (Paßwort,..)
- Physikal. Schutzmaßnahmen, Kryptographie
- Benutzerrechte:
  - Rechte, Anfragen an bestimmte Relationen anderer zu stellen (`GRANT SELECT ON relname TO jutta;`)
  - Rechte, bestimmte 'Änderungsoperationen' durchzuführen (`GRANT UPDATE ON relname TO public; GRANT CREATE TRIGGER TO gremeyer;`)
  - speziell: Möglichkeit, erhaltenes Recht an andere weiterzugeben (`GRANT SELECT ON relname TO gremeyer WITH GRANT OPTION;`)
  - Rückname von vergebenen Rechten (`REVOKE recht ON relname FROM benutzer [RESTRICT|CASCADE]`)
- Rechte nicht auf Basisrelationen, sondern nur auf Ausschnitten (Views) -> Sichten im Rahmen von Datenschutz wichtig

Datenschutz, Datensicherheit

Datensicherheit in Datenbanken:

- Wiederherstellung der Daten bei Systemfehlern: Recovery

- 2 Speicherarten:
  instabiler: Hauptspeicher, Cache; Inhalt geht bei Systemfehler verloren
  stabiler: Platte, Diskette, CD,..; Inhalt geht bei Fehler der Speichermediums verloren

- Maßnahmen gegen Datenverlust
  - vom instabilen Speicher: Log-Buch, Journal, Wiederherstellungsprotokolle, Einsatz von Schatten-Speichern
  - vom stabilen Speicher: Backups, Archive, komplette Spiegelungen

Datenschutz, Datensicherheit

Log-Buch:
enthält vollständige Historie mit allen Änderungen auf der Datenbank; z.B. Eintrag (T,begin) bei Start einer Transaktion T, (T,A,a) für T schreibt für Attribut A Wert a,... (T,A,a',a) für T schreibt für A bei altem Wert a' neuen Wert a, (T,commit), (T,savepoint s),...

Problem: Log-Buch in instabilem Speicher: kann verloren gehen -> Log-Seiten müssen sofort auf stabilen Speicher geschrieben werden

Journal:
neben Log-Daten auch Daten über Benutzer, Zugriffsstatistiken, etc

Recovery: Algorithmus, wird nach Systemfehler gestartet, untersucht Log-Buch, setzt DB in letztmöglichen konsistenten Zustand zurück;
Algorithmus: Log-Buch rückwärts durchlaufen bis zu letzten savepoint- oder commit-Einträgen; ab dort vorwärts alle (T,A,a)-Einträge finden, a für A in DB schreiben; Transaktionen ohne (T,commit) oder mit (T,abort) ignorieren, dafür nur Warnung ausgeben ('T not committed')