Datenbanken

Transaktionen

Nikolaus Augsten

nikolaus.augsten@sbg.ac.at

FB Computerwissenschaften
Universität Salzburg

Wintersemester 2013/14

Literatur und Quellen

Lektüre zu den Themen "Transaktionen":

- Kapitel 9 (Transaktionen) aus Kemper und Eickler: Datenbanksysteme: Eine Einführung. 8. Auflage, Oldenbourg Verlag, 2011.
  7. Auflage: `http://www.oldenbourg-link.com/isbn/9783486592771`

Literaturquellen

- Elmasri and Navathe: Fundamentals of Database Systems. Fourth Edition, Pearson Addison Wesley, 2004.
- Silberschatz, Korth, and Sudarashan: Database System Concepts, McGraw Hill, 2006.

Danksagung Einige Folien nach einer Vorlage von:

- Sven Helmer, Freie Universität Bozen, Italien

Inhalt

Inhalt

Was ist eine Transaktion?

- Eine Transaktion ist eine Programmeinheit, die auf Daten zugreift und diese möglicherweise verändert.
- Beispiel: überweise \$50 von Konto A nach Konto B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Transaktionsverwaltung beschäftigt sich mit zwei Hauptproblemen:
  1. gleichzeitige Ausführung mehrere Transaktionen
  2. Fehler verschiedener Art (z.B. Hardware Fehler, Systemabsturz)

ACID Eigenschaften

- Datenbanksysteme müssen ACID für Transaktionen garantieren:
  - Atomicity: entweder alle Operationen einer Transaktion werden ausgeführt oder gar keine
  - Consistency: die Ausführung einer isolierten Transaktion erhält die Datenbank in konsistentem Zustand
  - Isolation: obwohl mehrere Transaktionen gleichzeitig ausgeführt werden ist es für jede einzelne Transaktion so, als wäre sie alleine
  - Durability: Nach erfolgreicher Beendigung einer Transaktion müssen deren Veränderungen in der Datenbank dauerhaft erhalten bleiben, auch bei Systemabsturz oder anderen Fehlern.

Atomicity

- Beispiel: überweise \$50 von Konto A nach Konto B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Was, wenn Fehler (Hardware od. Software) nach Schritt 3 auftritt?
  - Geld geht verloren
  - Datenbank ist inkonsistent
- Atomicity:
  - entweder alle Operationen oder gar keine
  - Änderungen von teilweise ausgeführten Transaktionen werden nicht in die Datenbank geschrieben

Consistency

- Beispiel: überweise \$50 von Konto A nach Konto B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Konsistenzbedingung Beispiel: Summe $A + B$ muss unverändert bleiben
- Konsistenzbedingungen allgemein:
  - explizite Integritätsbedingungen (z.B. Fremdschlüsselbedingung)
  - implizite Integritätsbedingungen (z.B. Summe aller Kontostände einer Bank muss gleich der Summe der Kontostände aller Filialen sein)
- Transaktion:
  - muss eine konsistente Datenbank vorfinden
  - währen der Transaktion sind inkonsistente Zustände erlaubt
  - nach Ende der Transaktion muss Datenbank wieder konsistent sein

Isolation – Einführendes Beispiel

- Beispiel: überweise \$50 von Konto A nach Konto B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Angenommen, es gibt eine zweite Transaktion T_2:
  - $T_2 : R(A), R(B), print(A + B)$
  - T_2 wird zwischen den Schritten 3 und 4 ausgeführt
  - T_2 sieht einen inkonsistenten Datenbankzustand und liefert das falsche Ergebnis

Isolation

- Triviale Isolation: alle Transaktionen laufen seriell (nacheinander)
- Isolation für nebenläufige (concurrent) Transaktionen: Für jedes Paar T_i und T_j von Transaktionen scheint es für T_i als wäre T_j bereits beendet bevor T_i gestartet ist oder hätte noch nicht begonnen, wenn T_i endet.
- Schedule: (Historie)
  - gibt die chronologische Ordnung einer Sequenz von Befehlen aus verschiedenen Transaktionen an
  - Äquivalente Schedules resultieren immer in identischen Datenbankinstanzen wenn sie mit identischen Instanzen starten
- Serialisierbarer Schedule:
  - äquivalent einem seriellen Schedule
  - serialisierbarer Schedule von $T1$ und $T2$ ist entweder zu $T1, T2$ oder $T2, T1$ äquivalent

Durability

- Wenn eine Transaktion endet, macht sie ein commit.
- Beispiel: Transaktion macht commit zu früh:
  - Transaktion schreibt A und macht ein commit
  - A wird in den Festplattenbuffer geschrieben
  - dann stürzt das System ab
  - der Wert von A geht verloren
- Durability: Nachdem eine Transaktion mit commit abgeschlossen hat, bleiben deren Änderungen auch im Falle eines Systemfehlers erhalten.
- Commit darf erst abgeschlossen werden, wenn alle Änderungen persistent gespeichert sind:
  - entweder in eine Log Datei oder direkt in die Datenbank
  - Datenbank muss im Falle eines Absturzes wiederhergestellt werden (recovery)

Unverwünschte Phänomene nebenläufiger Transaktionen

- Dirty read
  - Transaktion liest Daten, die von nebenläufiger uncommitted Transaktion geschrieben wurden
  - Problem: die Leseoperation gibt einen Wert der nie in der Datenbank war, da die schreibende Transaktion abgebrochen wurde
- Non-repeatable read
  - aufeinanderfolgende Leseoperationen auf denselben Dateneintrag ergeben verschiedene Werte innerhalb einer Transaktion (aufgrund von Änderung durch andere Transaktionen)
  - z.B. nebenläufige Transaktionen T_1: $x = R(A)$, $y = R(A)$, $z = y - x$ und T_2: $W(A = 2 * A)$, dann kann z entweder 0 oder den Anfangswert von A haben (sollte 0 sein!)
- Phantom read
  - dieselbe Anfrage innerhalb einer Transaktion gibt verschiedene Tupel zurück, wenn sie mehrmals ausgeführt wird
  - z.B. "Q: SELECT * FROM Konten WHERE *Guthaben* $>$ 1000" ergibt 2 Tupel beim ersten Aufruf, dann wird ein neues Konto mit *Guthaben* $>$ 1000 durch eine andere Transaktion eingefügt; beim zweite Aufruf gibt Q drei Tupel zurück.

Isolation Levels (SQL Standard)

- Read uncommitted: dirty, non-repeatable, phantom
  - Schreiboperationen überschreiben keine "uncommitted" Daten
  - Leseoperationen können Daten lesen, die nicht "committed" sind
- Read committed: non-repeatable, phantom
  - Leseoperationen können nur "committed" Daten lesen
  - cursor stability: innerhalb einer `SELECT` Anfrage sind Leseoperationen "repeatable"
- Repeatable read: phantom
  - phantom reads sind möglich
- Serializable:
  - keine der Phänomene sind möglich

Isolation Levels (SQL Standard)

- "Serializable" in SQL ist nicht identisch mit Serialisierbarkeit nach unserer Definition.
  - Oracle und ältere PostgreSQL Versionen erlauben nicht-serialisierbare Schedules, die jedoch der Definition von SQL Serializable genügen.
  - Jeder serialierbare Schedule ist jedoch "Serializable" nach SQL.
- Viele Systeme implementieren nur zwei Levels:
  - Read committed (meist sogar "cursor stability")
  - Serializable
- Es muss ein Kompromiss eingegangen werden:
  - read committed ist schneller, aber Ergebnisse könnten falsch sein
  - serializable garantiert exakte Ergebnisse, ist aber langsamer

Wann sollte schwächerer Level genommen werden?

- Anfrage braucht keine exakten Antworten (z.B. statistische Anfragen)
  - Beispiel: Zähle alle Konten mit Guthaben$>$ \$1000.
  - read committed ist genug
- Transaktionen mit menschlicher Interaktion
  - Beispiel: Flug Reservierung
  - Kosten für Serialisierbarkeit zu hoch, da Transaktionen zu lange dauern

Beispiel: Flug Reservierung

- Reservierung umfasst drei Schritte:
  1. rufe verfügbare Sitzplätze ab
  2. Kunde entscheidet sich für Sitzplatz
  3. reserviere Sitzplatz
- Einzelne Transaktion:
  - Sitzplätze sind gesperrt (können weder gelesen noch geschrieben werden) während Kunde sich entscheidet
  - alle anderen Benutzer müssen warten
- Zwei Transaktionen: (1) Liste holen, (2) Reservieren
  - Sitz ist möglicherweise inzwischen schon weg, wenn Kunde versucht zu reservieren
  - ist leichter zu tolerieren als das System zu blockieren

Transaktionen in SQL/1

- Parameter für Transaktion in SQL setzten:

  set transaction *level, access mode*

- level ist einer der Isolation Level:
  - **read uncommitted**
  - **read committed**
  - **repeatable read**
  - **serializable**
- access mode kann sein:
  - **read only**: nur Leseoperation in Transaktion
  - **read write**
- Read only vs. read write:
  - read only Performance deutlich erhöhen, das es zwischen read-only Transaktionen keine Konflikte gibt
  - sobald eine Transaktion schreibt, kann es (auch mit read-only Transaktionen) zu Konflikten kommen

Transaktionen in SQL/2

- Eine Transaktion beginnen:

  start transaction

- Eine Transaktion abbrechen:

  rollback [work]

  - Beispiel: Überweisung muss abgebrochen werden da zu wenig Geld auf Konto.

- Eine (erfolgreiche) Transaktion beenden:

  commit [work]

  - **commit** Befehl kann von Datenbank auch mit *rollback* beantwortet werden, falls ein *commit* nicht möglich ist
  - Beispiel: Konflikt mit anderen Transaktionen, die vorher *commit* gesandt haben

- Auch die Datenbank kann Transaktionen abbrechen, z.B., wenn sich zwei Transaktionen gegenseitig blockieren (Deadlock)

Zusammenfassung

- Transaktion: Programmeinheit, die als Ganzes ausgeführt werden soll.
- ACID soll für Transaktionen garantiert werden:
  - Atomicity: alles oder nichts
  - Consistency: konsitente Zustände sehen und hinterlassen
  - Isolation: nebenläufige Transaktionen stören sich nicht
  - Durability: auch im Fehlerfall Konsistenz und keine verlorenen Transaktionen
- SQL bietet Transaktionen mit unterschiedlichen Garantien.
- Trade-Off zwischen Korrektheitsgarantie und Effizienz

Inhalt

Datenbanken Pflichtprogramm

- Datenbanken 1: Grundlagen und Theorie
  - ER-Modell
  - Relationales Modell
  - SQL
  - Relationale Entwurfstheorie
  - Transaktionen

  ➥ 2. Semester 

- Datenbanken 2: Implementierung - wie baue ich ein DBMS?
  - Speichermedien und Dateiorganisation
  - Indexstrukturen (Hash, B^+-Baum)
  - Algorithmen für Selektion, Join, Sortierung
  - Anfrageoptimierung

  ➥ Papier und Bleistift

  ➥ 3. Semester (Wintersemester 2014/15)

Modul: Verarbeitung großer Datenmengen

- Tuning von Datenbanksystemen: schnelle Systeme in der Praxis
  - schnelle SQL Anfragen schreiben
  - Indices effizient einsetzten
  - parallele Zugriffe optimieren
  - Fehlertoleranz vs. Effizienz
  - Speicher-Backend tunen

➥ Hands-on im Proseminar

- 6 Projektblätter in Gruppen ausarbeiten
- meist Experimente an konkreten Systemen

➥ 3. Semester (Wintersemester 2014/15)

- parallel zu Datenbanken 2
- ideale Ergänzung (Theorie und Praxis)

Ankündigungen

- Tutorium: findet heute zum letzten mal statt
- Prüfungsbeispiele: alle Prüfungsfragen seit 2013 sind online
- StudienassistentIn gesucht
  - Oktober bis Jänner (evtl. bis Juni verlängerbar)
  - 10 Stunden pro Woche
  - ca. 2.170 EUR brutto (4 Monate)

  ➥ Aufgaben
  - Hilfe bei Latex/TikZ Grafiken
  - Gestaltung Website der DB-Gruppe
  - ...

  ➥ Bitte bis 27. Juni melden: nikolaus.augsten@sbg.ac.at

Wir freuen uns auf Ihre Bewerbung!

The End

Alles Gute für die Prüfung!

Viele Grüße aus Utah.
Augsten