<https://moodle.lmu.de/course/view.php?id=19485>

Transaktion: ist eine Folge von Befehlen(read, write), die die DB von einen konsistenten Zustand in einen anderen konsistenten Zustand überführt.

Hauptaufgaben der Transaktions-Verwaltung:

Synchronisation

Recovery

Bsp.: eine 200Euro Überweisung, Plan(2 Transaktionen):

-erniedrige Stand von Jad um 200Euro.

- erhöhe Stand von Meier um 200Euro.

Eigenschaften von Transaktionen:

ACID-Prinzip:

* Atomicity, Consistency, Isolation, Durability

SQL Befehle die man gelernt hat, terminieren!

Steuerung von Transaktion:

* Begin of Transaction (BOT): markiert den Anfang einer Transaktion.
* End of Transaction (EOT) : Alla Änderungen vom letzten BOT werden festgeschrieben- SQl: commit oder commit work
* Abort: markiert den Abbruch, zu BOT zurückführt- SQL: rollback oder rollback work
* Üntestüzung langer Transaktionen:
  1. Define savepoint: SQL savepoint <identifier>
  2. Backup tranaction: rollbackt to <identifier>

Ende von Transaktion:

* Commit gelingt: der neue Zustand wird dauerhaft gespeichert
* Commit scheitert: -der ursprüngliche Zustand wie zu Beginn der Trans. Bleibt erhalten (widererstellt). Ein Commit kann z.B. scheitern, wenn die Verletzung von Integritätsbedingungen erkannt wird.
* ROLLBACK benutzter widerruft Änderungen.Diagram

  Description automatically generated

Aufgaben eines DBMS (3 Kapiteln)

Wahrung eines korrekten DB-Zustands unter realen Benutzungsbedingung, d.h.:

* + 1. Synchronisation (Concurrency Control) Schutz vor Fahleren durch sich gegenseitig störenden nebenläufigen Zugriff mehrerer Benutzer
    2. Datensicherheit (Recovery): Schutz vor Verlust von Daten durch technische Fehler.
    3. Integrität(integrity): Schutz vor Verletzung der Korrektheit und Vollständigkeit von Daten durch berechtigte Benutzer.

Synchronisation:

Serielle Ausführung von Transaktionen:

* Unerwünscht, da die Leistungsfähigkeit des Systems beeinträchtigt ist.
* Folgen: niedriger Durchsatz, hohe Wartezeiten.

Mehrbenutzerbetrieb:

* Führt i.A zu einer besseren Auslastung des Systemes(z.B. Wartezeiten bei E/A- Vorgängen können zur Bearbeitung anderer Transaktionen genutzt werden.)
* Aufgabe der Synchronisation
* Gewährleistung des logischen Einbenutzerbetriebs, d.h. innerhalb einer TA ist ein Benutzer von den Aktivitäten anderer Benutzer nicht betroffen

Anomalien im Mehrbenutzerbetrieb:

Wie unterscheiden u.a. folgende Grundmuster von Anomalien:

* Verloren gegangene Änderungen(Lost Updates)
* Zugriff auf schmutzige (nicht dauerhaft gültige ) Daten (Dirty Read/ Write)
* Nicht-reproduzierbares Lesen (Non Repeatable Read)
* Phantomproblem

Table

Description automatically generated

Lost Updates:

Zwei Transaktionen T1, T2 führen je eine Änderung auf dem selben Objekt aus:

Text

Description automatically generated

Verstoß gegen Durability:

In der DB ist nur die Änderung von T1 wirksam, die Änderung von T2 ist verloren gegangen. Table, timeline

Description automatically generated with medium confidence

Dirty Read/ Write:

Zugriff auf schmutzige Daten, d.h. auf Objekte, die von einer noch nicht abgeschlossenen Transaktion geändert wurden.

Text

Description automatically generated

Table

Description automatically generated with medium confidence

Non Repeatable Read :

* Eine Transaktion….(Seite 15)

Ein Teuere Wayout:

**Dann wenn einen Nutzer für eine Relation INSERT INTO macht, soll die ganze Relation zum anderen Nutzer unerreichbar (unselectable) scheinen?**

**Nebentäti**gkeit vor den Benutzern verbergen.

Aber Transparent für den Benutzer…

Schedules:

* Allgemeinder Schedule: Ein Schedule („Historie“) für eine Menge {T1, …, Tn} von Transaktionen ist eine Folge von Aktionen, die durch Mischen der Aktionen der Transaktionen Tm entsteht wobei die Reihenfolge innerhalb der. Jeweiligen Transaktion beibehalten wird.
* Allgemeine Schedules bieten offenbar eine beliebige Verzahnung und sind daher aus Performanz-Gründen erwünscht.
* Frage: Warum darf die Reihenfolge der Aktionen innerhalb einer TA nicht verändert werden.

20….Seite 30 ….

Serieller Schedule:

Ein serieller Schedule: Aktionen der Tran. Nicht untereinander verzahnt. Sondern in Blöcken hintereinander ausgeführt.

Aus Sicht des Isolation Prinzips sind serielle Schedules erwünscht

Kompromiss zwischen Performanz und Isolation(bzw. allgem. Und seriellen Schedules)

Serialiesiebarer Schedule:

Ein allgemeiner Schedule S von {T1, …, Tn} ist serialisierbar, wenn er dieselbe Wirkung hat wie ein beliebiger serieller Schedule von {T1, …, Tn}

Nur serialisierbare Schedulers dürfen zugelassen werden!

Diagram

Description automatically generated

Konflikt-Äquivalenz:

Schreib-lese-Abhängigkeit

Idee: wenn in S1 eine Transaktion T1 z.B. einen Wert liest, den T2 geschrieben hat, dann muss das auch in S2 so sein.

Abhängigkeit:

Sei S ein Schedule. Wir sprechen von einer:

* Schreib-Lese-Abhängigkeit von Ti 🡪 Tj

Es existiert Objekt x, so dass in S wj(x) vor rj(x) kommt. Abkürzung: wri,j(x)

* Lese-schreibe-Abhängigkeit von Ti 🡪 Tj

Es existiert Objekt x, so dass in S rj(x) vor wj(x) kommt. Abkürzung: rwi,j(x).

* Schreibe-Schreib-Abhängigkeit von Ti 🡪Tj

Es existiert Objekt so dass in S wj(x) vor wj(x) kommt. Abkürzung: wwi,j(x)

2 Shcedules heißen konfliktäqualent:

* wenn sie die gleichen Transaktions- & Aktionsmengen besitzen bzw. beide Operationen.
* Und sie die gleichen Abhängigkeitsmengen besitzen (lese-schreibe- ab. Von Objekt x in S1 dann auch in S2)

Graphical user interface, text, application

Description automatically generated

Diagram

Description automatically generated