Vorlesung Implementierung von Datenbanksystemen

8. Transaktionen

Prof. Dr. Klaus Meyer-Wegener

Wintersemester 2019/20

Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

- **Umgang mit Fehlern und Ausfällen**
- **Programmfehler –**
  Datenbank-Anwendungsprogramm stürzt ab:
  - Ist fehlerhaft:
    - Division durch Null, unzulässige Adressierung (Exception), ...
  - Wird von außen abgebrochen:
    - Überschreiten einer CPU-Zeit-Vorgabe
    - Sättigungszustand im System
    - Eingriff der Operator (cancel, kill)
- **Ergebnis:**
  - Hauptspeicherinhalte des Programms (Programmvariablen = Programmzustand) verschwunden
  - Daten im Puffer und auf Platte, die das Programm zu ändern begonnen hatte, in irgendeinem unbekannten Zwischenzustand (unvollständig)

Programmfehler

- **Notwendige Maßnahmen:**
  - Daten des Programms, die noch im Puffer sind, wegwerfen
  - Datenbestand auf Platte muss von Hand bereinigt werden, bevor das nächste Programm darauf zugreifen darf.
    - Programmierer müssen Programmlauf nachvollziehen und Änderungen der Daten einzeln rückgängig machen!

- **Aufwand! Zeit! Daten blockiert für andere!**

Systemfehler

- **Systemfehler:**
  Datenbankverwaltungssystem (DBVS) fällt aus oder
  Betriebssystem (BS) fällt aus oder
  Hardware-Fehler oder
  Stromausfall oder ...
  - Programm ist (meistens) unschuldig.
    - War evtl. sogar schon fertig,
      d.h. alle Änderungen vollständig ausgeführt – aber leider bisher nur im Puffer des BS oder DBVS abgelegt
- **Ergebnis:**
  - Pufferinhalte sind verlorengegangen.
  - Daten auf Platte in unbekanntem Zustand
    - Muss vor Neustart irgendwelcher Programme bereinigt werden

Systemfehler (2)

- **Notwendige Maßnahmen:**
  - *Alle* Programme, die gerade liefen oder kurz zuvor fertig geworden waren, von Hand nachvollziehen!
  - Klären, welche Programm ihre Ergebnisse vollständig auf die Platte bringen konnten (also abgehakt werden können) und welche nach einer Bereinigung wiederholt werden müssen.

- **Noch viel größerer Aufwand!**

Gerätefehler

- **Gerätefehler: Platte defekt**
  - Z.B. Headcrash; nichts mehr lesbar! Oder Diebstahl ...
- **Ergebnis:**
  - Programme, DBVS und BS können nicht mehr weitermachen.
  - Änderungen auf Platte verloren
    - Die ggf. noch im Hauptspeicher verfügbaren daher sinnlos (ohne Basis)
- **Notwendige Maßnahmen:**
  - Neue Platte formatieren
  - Sicherungskopien (Backup) einspielen
  - Alle Änderungen seit Anlegen der Sicherungskopie wiederholen
    - Also seit gestern Abend, letzter Woche, Anfang des Monats, ...
  - Alle Programme mit gleichen Eingabedaten (!) und in der gleichen Reihenfolge (!) nochmal ablaufen lassen
- **Wiederum: Aufwand, Zeit, Daten blockiert**

Erwünschte Zustände der Daten auf Platte

- **Physische Konsistenz**:
  - Korrektheit der Speicherungsstrukturen
  - Alle Verweise und Adressen (TIDs) stimmen.
  - Alle Indexe sind vollständig und stimmen mit den Primärdaten überein.
    - Kein Schlüsselwert eines neuen oder geänderten Satzes fehlt.
    - Kein Schlüsselwert eines gelöschten oder geänderten Satzes ist übriggeblieben.
- **Logische Konsistenz**:
  - Korrektheit der Dateninhalte
    - Stellen einen (möglichen) Zustand der realen Welt dar
  - Alle Bedingungen des Datenmodells (Primärschlüsseleigenschaft, Referenzielle Integrität) und alle benutzerdefinierten Integritätsbedingungen (Assertions) sind erfüllt.

Erwünschte Zustände der Daten auf Platte (2)

- **Annahmen:**
  - Alle vollständig ausgeführten **Datenbank-Operationen** (z.B. SQL-Anweisungen) hinterlassen einen **physisch konsistenten** Zustand.
    - Soll heißen: Das DBVS selbst ist nicht fehlerhaft.
  - Alle vollständig ausgeführten **Anwendungsprogramme** hinterlassen einen **logisch konsistenten** Zustand.
    - Soll heißen: Programmiererin hat alles richtig gemacht.
    - Dies wird im folgenden noch differenziert; es kann auch nach einem *Teil* eines Anwendungsprogramms schon wieder ein konsistenter Zustand erreicht sein.
- **Nach einem Fehler:**
  - Die Daten sind i.Allg. **weder physisch noch logisch konsistent.**

Ziel

- **Systemunterstützung**
  - zur **Wiederherstellung (Recovery)**
    eines logisch und physisch konsistenten Zustands der Daten
    nach einem Fehlerfall
    - Am besten automatisch
- **Der herzustellende konsistente Zustand kann sein:**
  - Entweder der vor Beginn der Änderungen eines unvollständig ausgeführten Programms herrschende
    - **Rückgängigmachen** der bereits ausgeführten Änderungen
      (**Backward Recovery**)
  - Oder der nach Abschluss aller Änderungen eines Programms erreichte
    - Komplettieren der unvollständigen Änderungen
      bzw. **Wiederholen** verlorengegangener Änderungen
      (**Forward Recovery**)

Ziel (2)

- **Voraussetzung:**
  - Geeignete **Sicherungs- und Protokollierungsmaßnahmen** im laufenden Betrieb (**Logging**)
    - Gezieltes Führen von (Sicherungs-) Kopien der Daten für die Wiederherstellung im Fehlerfall
    - Also Kosten: Speicher, E/A, CPU

8.1 Die Transaktion

- **Transaktion (TA):**
  - **Folge von DB-Operationen**,
    die, von einem logisch konsistenten Zustand ausgehend,
    die Datenbank wieder in einen logisch konsistenten Zustand überführt.
  - Muss vom Anwender definiert werden
- **Bei Fehler *vor* dem Ende der Transaktion:**
  - Alle Änderungen, die eine Transaktion bereits ausgeführt hatte,
    werden automatisch so **rückgängig gemacht**,
    dass sich die Daten wieder in dem Zustand befinden,
    in dem sie zu Beginn der Transaktion waren (Systemgarantie).
  - Der Datenbestand sieht dann aus, als ob die TA nie gestartet worden wäre.
  - Anwender kann selbst die TA neu starten (Programmlauf wiederholen).
- **Bei Fehler *nach* dem Ende der Transaktion:**
  - Alle Ergebnisse der Transaktion,
    die durch den Fehler verlorengegangen waren,
    werden automatisch **wiederhergestellt**.
  - Anwender bemerkt den Fehler nicht.

Die Transaktion (2)

- **Aufgabe des Anwenders dann nur noch: Transaktion definieren**
  - (Anfang und) Ende
    - Anfang meist implizit (Beginn des Programms, erste DB-Operation)
- **DB-Aufruf commit**
  - Anwendungsprogramm teilt dem DBVS mit,
    - dass es durch eine Reihe von DB-Änderungsoperationen wieder einen logisch konsistenten Zustand hergestellt hat oder
    - dass es nach einer Reihe von lesenden Zugriffen so weit fertig ist, dass andere die gelesenen Daten auch wieder ändern dürfen.
- **DB-Aufruf abort oder rollback**
  - Rückkehr in den Anfangszustand auf Wunsch des Anwenders

Beispiel

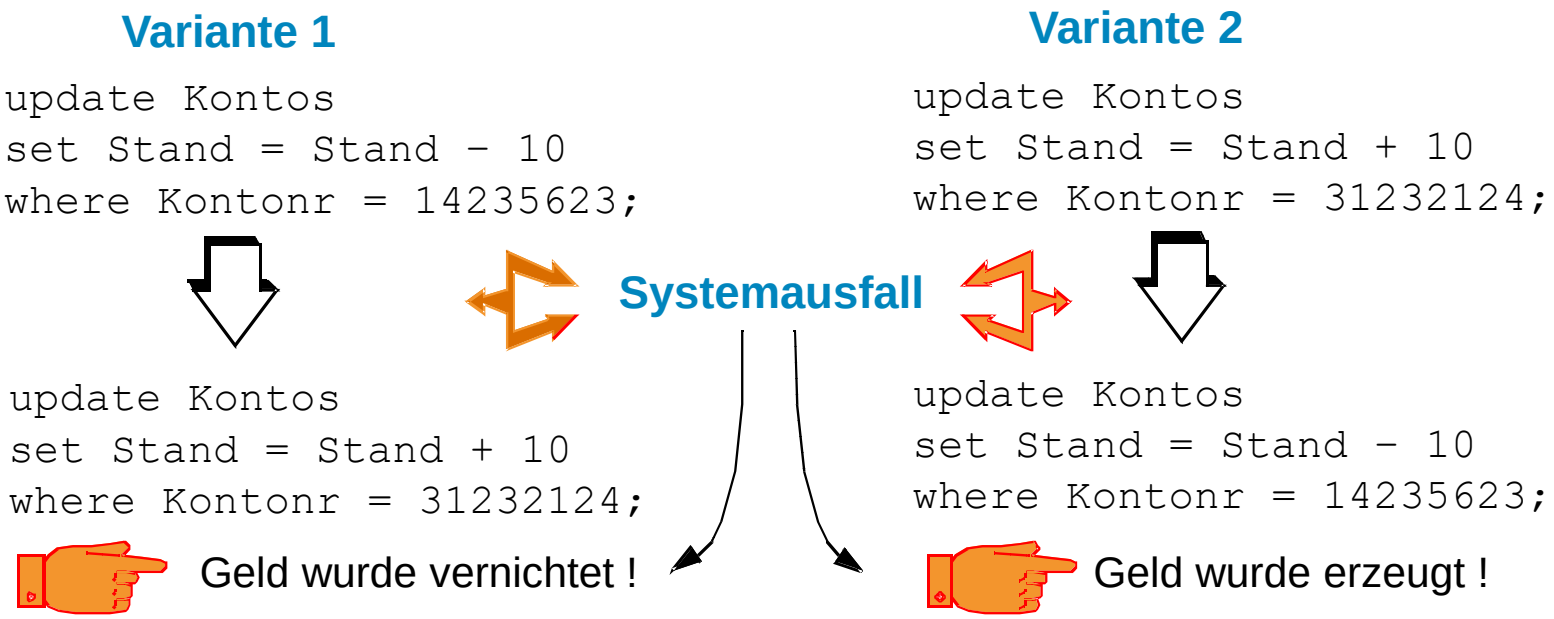
- **Programm zur Ausführung von Banküberweisungen:**

```
...
for ( ;; )
{
    scanf("%d %d %d", &von_konto, &auf_konto, &betrag);
    if ( ... )
          break;
    exec sql      update Kontos
                  set Stand = Stand - :betrag
                  where Kontonr = :von_konto;
    exec sql      update Kontos
                  set Stand = Stand + :betrag
                  where Kontonr = :auf_konto;
    exec sql commit;
}
```

  - Es darf weder Geld verschwinden noch welches aus dem Nichts entstehen!

Beispiel (2)

- **Andere Reihenfolge hilft auch nicht:**



- Einzelne DB-Operationen können die von der Anwendung geforderten Verarbeitungseinheiten nicht realisieren.

Charakterisierung von Transaktionen

- **Charakterisierung**
  - Datenbanktransaktion stellt logische **Arbeitseinheit** dar.
    - Englisch deshalb auch: Logical Unit of Work
  - Zusammenfassung von aufeinander folgenden DB-Operationen, die eine Datenbank von einem konsistenten Zustand in einen neuen konsistenten Zustand überführen (physische und logische Konsistenz)
    - DB-Operationen hier gleichzusetzen mit SQL-Befehlen
  - Innerhalb einer Transaktion können vorübergehend logisch inkonsistente Zustände auftreten.

- **Der Transaktionsmanager des DBVS garantiert**
  - entweder vollständige Ausführung einer Transaktion
  - oder Wirkungslosigkeit der gesamten Transaktion (und damit aller in ihr enthaltenen Operationen).

Atomarität

Eine Transaktion ist atomar:

- **Unteilbar:**
  - Alle zu einer Transaktion gehörenden DB-Operationen bilden eine Einheit.
- **Ununterbrechbar:**
  - Aus der Sicht der Benutzer wird die Transaktion entweder vollständig ausgeführt oder gar nicht.
- **Zusicherung des DBVS an den Anwender:**
  Alles oder nichts

Konsistenzerhaltung

Transaktion

- setzt bei ihrem Start einen **konsistenten Zustand** des Systems voraus und
- hinterlässt nach ihrem erfolgreichen Abschluss wieder einen konsistenten Zustand

- **Zusicherung der Programmierer an das DBVS**
  - Kann nicht immer überprüft werden (vgl. Banküberweisung)
  - Was geprüft werden kann (z.B. Referenzielle Integrität), wird aber geprüft.

- **Transaktion muss vollständig sein**
  - D.h. alle erforderlichen Änderungen sind enthalten
  - **So lang wie nötig**

Isolation

- **Mehrbenutzerbetrieb:**
  - Auf demselben Datenbestand arbeiten gleichzeitig mehrere Programme (Transaktionen).
  - Die müssen sich so abstimmen (bzw. durch das DBVS so "synchronisiert" werden), dass keine Inkonsistenzen entstehen.
- **Ziele:**
  - Solange eine Transaktion noch läuft, darf keine andere Transaktion von ihr bereits durchgeführte Änderungen lesen und benutzen, weil
    - die noch unvollständig und damit inkonsistent sind
    - die Transaktion noch scheitern kann
      und diese Änderungen dann wieder verschwinden
      - Die andere Transaktion hätte sonst ein "Phantom" gesehen.
  - Transaktionen laufen **isoliert** voneinander ab.

Isolation (2)

- **Ziele (Forts.):**
  - Eine Transaktion T1 darf entweder
    nur den Zustand *vor* Ausführung einer anderen Transaktion T2 sehen
    oder
    nur den Zustand *nach* Ausführung von T2.
    - Sieht sie Datenobjekt 1 vor der Änderung durch T2
      und Datenobjekt 2 nach der Änderung,
      kann dies kein konsistenter Zustand sein,
      sonst wären die beiden Änderungen ja nicht in T2 zusammengefasst worden.

Isolation (3)

- **Synchronisation der Transaktionen**
  - muss so erfolgen, dass die Wirkung auf den Datenbestand dieselbe ist wie bei einer strikt seriellen Ausführung der Transaktionen (eine nach der anderen)
  - **Fiktiver Einbenutzerbetrieb** für jede Transaktion
    - Optimal für Programmierung: nichts zu tun für gegenseitigen Ausschluss
- **Auch für rein lesende Zugriffe:**
  - Sollen nur einen konsistenten Zustand sehen
  - und keine inkonsistenten Zwischenzustände gleichzeitig arbeitender Änderungstransaktionen
- **Realisiert durch Blockierung von Daten für andere**
  - Deshalb Transaktion **so kurz wie möglich** machen

Dauerhaftigkeit

- **Eine dem Benutzer als erfolgreich abgeschlossen gemeldete Transaktion ist dauerhaft ("persistent").**

- **Zusicherung des DBVS an die Benutzer:**
  - Ihre Änderungen im Datenbestand gehen nicht durch spätere Fehler (Verlust des Puffers, Gerätefehler) verloren.
  - Sie bleiben unverändert erhalten
    (bis eine nachfolgende Transaktion sie wieder überschreibt).

Zusammenfassung: "ACID"-Eigenschaften

- **Atomarität (atomicity)**
  - Unteilbarkeit durch Transaktionsdefinition (Begin/BoT – End/EoT)
  - Alles-oder-Nichts-Prinzip
- **Konsistenzerhaltung (consistency)**
  - Eine erfolgreiche Transaktion garantiert, dass alle Konsistenzbedingungen (Integritätsbedingungen) eingehalten werden.
- **Isolation (isolation)**
  - Mehrere Transaktionen laufen voneinander isoliert ab und benutzen keine (inkonsistenten) Zwischenergebnisse anderer Transaktionen.
- **Dauerhaftigkeit (durability)**
  - Alle Ergebnisse erfolgreicher Transaktionen müssen persistent gemacht worden sein, bevor der Erfolg an die Anwendung gemeldet werden darf.

8.2 Programmierung von Transaktionen

- **DB-Zugriffe nur noch innerhalb von Transaktionen erlaubt**
  - Eine Transaktion beginnt automatisch mit der ersten ausführbaren SQL-Anweisung im Programm.
  - Die Anweisungen

    ```
    exec sql commit;
    exec sql rollback;
    ```

    beenden die Transaktion.
  - Mit der nächsten ausführbaren SQL-Anweisung beginnt automatisch eine neue.
- **Wichtiges Prinzip:**
  - Transaktionen sollten so kurz wie möglich und so lang wie nötig sein! (siehe oben)
    - Möglichst keine Interaktion mit dem Benutzer innerhalb einer Transaktion!
    - Eingabedaten (soweit möglich) *vor* dem ersten Zugriff auf die Datenbank prüfen!

Programmierung von Transaktionen (2)

- **Wenn ein Programm *nur eine* Transaktion ausführt:**
  - Scheitert das Programm, kann es (ggf. nach Korrektur) mit den gleichen Eingabedaten noch einmal ausgeführt werden.
- **Wenn ein Programm *zwei oder mehr* Transaktionen hintereinander ausführt:**
  - Scheitert das Programm, muss seine erneute Ausführung so modifiziert werden, dass bereits erfolgreich abgeschlossene Transaktionen nicht wiederholt werden!
    - Von den Eingabedaten müssen die weggelassen (übergangen) werden, die bereits zu einer erfolgreichen Änderung des Datenbestands geführt haben (vgl. Banküberweisung).
    - Der Programmierer muss den Fall der erneuten Ausführung explizit vorsehen,
      - d.h. den Programmzustand (Inhalte wichtiger Variablen) sichern und beim Wiederholen in Abhängigkeit davon Programmteile überspringen.

Beispiel

Relation zur Verwaltung eines Lagerbestands:

`Vorräte (Teilenr, Lager, Menge, StandVom)`

Einlesen und Verbuchen des Tagesumsatzes
in mehreren aufeinander folgenden Transaktionen,
damit der Datenbestand nicht zu lange blockiert ist
und nach Fehler und Rücksetzen nicht ganz von vorn begonnen werden muss:

```
#include <stdio.h>
#define EOF ...

main () {
  exec sql begin declare section;
       char datum[11], stand_vom[11], lager[21];
       unsigned int teilenr, entnahme, nachlieferung;
  exec sql end declare section;

  char letztes_datum[11] = " ";
```

Beispiel (2)


```
while (! EOF) {
   scanf ("%10s %u %20s %u %u",
            datum, &teilenr, lager, &entnahme, &nachlieferung);

   if (nach (letztes_datum, datum)) {
      printf ("Eingabe nicht nach Datum sortiert!\n");
      exit (1);
   }

   strcpy (letztes_datum, datum);

   exec sql  select stand_vom
             into :stand_vom
             from Vorrat
             where Teilenr = :teilenr
             and Lager = :lager;
```

Beispiel (3)


```
   if (nach (datum, stand_vom)) {
      exec   sql  update Vorrat
                  set Menge = Menge
                       + :nachlieferung
                       - :entnahme,
                      stand_vom = :datum
                  where Teilenr = :teilenr
                  and Lager = :lager;
   }

   exec sql commit;

} /* while */

}
```