

**ETH**  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

# Architektur und Implementierung von Datenbanksystemen WS 05/06

Dr. Jens-Peter Dittrich  
 jens.dittrich@inf  
 www.inf.ethz.ch/~jensdi  
 Institut für Informationssysteme



**ETH**  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Agenda

- Transaktionskonzept
- Recovery
- ARIES

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 2

**ETH**  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Transaktionsverwaltung

**Beispiel: Überweisung**

1. Lese den Kontostand von A in die Variable a: read(A,a);
2. Reduziere den Kontostand um 50 CHF: a := a - 50;
3. Schreibe den neuen Kontostand in die Datenbasis: write(A,a);
4. Lese den Kontostand von B in die Variable b: read(B,b);
5. Erhöhe den Kontostand um 50 CHF: b := b + 50;
6. Schreibe den neuen Kontostand in die Datenbasis: write(B,b);

Was passiert, wenn nur ein Teil dieser Anweisungen ausgeführt wird?

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 3

**ETH**  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Operationen auf Transaktionsebene

- **begin of transaction (BOT)**
  - kennzeichnet den Beginn einer Transaktion
- **commit**
  - Alle Änderungen der Transaktion werden festgeschrieben
  - Änderungen werden dauerhaft in die Datenbank eingebracht
- **abort (rollback)**
  - bricht Transaktion ab
  - DB muss sicherstellen, dass Datenbasis wieder in den Zustand zurückversetzt wird, der vor der Transaktion existierte

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 4

**ETH**  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Operationen auf Transaktionsebene

- **define savepoint**
  - definiert einen Sicherungspunkt, auf den sich die noch aktive Transaktion zurücksetzen lässt
  - Änderungen bis zu diesem Sicherungspunkt werden **noch nicht** in die Datenbasis eingebracht (es kann noch ein abort geben)
- **backup transaction**
  - aktive Transaktion wird auf den jüngsten (letzten) Sicherungspunkt zurückgesetzt

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 5

**ETH**  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Abschluss einer Transaktion: erfolgreich

- **Beispiel**
  - BOT**
  - op<sub>1</sub>
  - op<sub>2</sub>
  - ...
  - op<sub>n</sub>
  - commit**
- commit macht Änderungen von op<sub>1</sub>, ..., op<sub>n</sub> persistent

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 6

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Abschluss einer Transaktion: Abbruch

- Beispiel
  - BOT
  - OP<sub>1</sub>
  - OP<sub>2</sub>
  - ...
  - OP<sub>i</sub>
  - abort
- DB muss sicherstellen, dass alle Änderungen von OP<sub>1</sub>,...,OP<sub>i</sub> rückgängig gemacht werden

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 7

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Abschluss einer Transaktion: Fehler

- Beispiel
  - BOT
  - OP<sub>1</sub>
  - OP<sub>2</sub>
  - ...
  - OP<sub>k</sub>
  - !FEHLER
- Soft- oder Hardwarefehler (Stromausfall, HD-Crash, Programmfehler, aufgedeckter Deadlock, Verletzung von Konsistenzbedingung, etc.)
- DB muss sicherstellen, dass alle Änderungen von OP<sub>1</sub>,...,OP<sub>k</sub> rückgängig gemacht werden

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 8

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Anforderungen an die Transaktionsverwaltung

- gleichzeitig (nebenläufig) ablaufende Transaktionen
- Mehrbenutzersynchronisation
- Datenbanken gegen Soft- und Hardwarefehler schützen
- Abgeschlossene Transaktionen müssen erhalten bleiben
- Nicht abgeschlossene Transaktionen müssen vollständig zurückgesetzt werden

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 9

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Eigenschaften von Transaktionen: ACID

- Atomicity (Atomarität)
  - Transaktion ist kleinste, nicht mehr weiter zerlegbare Einheit
  - Entweder werden alle Änderungen der Transaktion ausgeführt oder keine
  - Alles-oder-Nichts-Prinzip
- Consistency (Konsistenz)
  - Transaktion hinterlässt einen konsistenten Datenbestand
  - Andernfalls wird sie zurückgesetzt
  - Zwischenzustände während der TA-Bearbeitung dürfen aber inkonsistent sein
  - Endzustand muss die im Schema definierten Konsistenzbedingungen erfüllen (z.B. referentielle Integrität)

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 10

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Eigenschaften von Transaktionen: ACID

- Isolation
  - nebenläufig (parallel) ausgeführte Transaktionen dürfen sich nicht beeinflussen
  - Effekte anderer Transaktionen dürfen nicht sichtbar sein
- Durability (Dauerhaftigkeit)
  - Änderungen erfolgreich abgeschlossener (committed) Transaktionen bleiben dauerhaft in der Datenbank vorhanden
  - Dies muss auch nach einem Systemfehler (Hard- oder Softwarefehler) gewährleistet sein!
  - Änderungen erfolgreich abgeschlossener Transaktionen können nur durch kompensierende Transaktionen aufgehoben werden

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 11

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Beispiel: Transaktionsbeginn und -ende

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 12

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Wie wird ACID vom DBMS garantiert?

- A & D: Recovery (heute)
- I: Mehrbenutzersynchronisation (nächste Stunde)

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 13

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Transaktionsverwaltung in SQL

- **commit work**
  - Änderungen werden - falls keine Konsistenzverletzungen oder andere Probleme auftauchen - festgeschrieben
  - Schlüsselwort **work** ist optional
- **rollback work**
  - Alle Änderungen sollen zurückgesetzt werden
  - Anders als der **commit**-Befehl muss das DBMS die erfolgreiche Ausführung eines **rollback**-Befehls immer garantieren können.
  - Schlüsselwort **work** ist optional
  - rollback kann optional bis zum letzten savepoint erfolgen:
    - rollback**
    - to savepoint xy**

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 14

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Zustandsübergänge einer Transaktion

- **potentiell**  
TA wartet uns ist bereit in den Zustand "aktiv" zu wechseln. Der Übergang heisst "inkarnieren".
- **aktiv**  
TA rechnet und konkurriert um Betriebsmittel mit anderen TAs
- **wartend**  
TA wurde vom System (z.B. wegen zu hoher Last) angehalten. Verringert sich die Last des Systems, kann die TA wieder aktiviert werden.
- **abgeschlossen**  
TA wurde durch commit abgeschlossen, Integritätsbedingungen wurden noch nicht überprüft, Änderungen der TA noch nicht persistent
- **persistent**  
Änderungen der TA wurde in Datenbasis persistiert

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 15

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Zustandsübergänge einer Transaktion

- **gescheitert**  
TA wurde durch **abort**, Systemfehler oder durch Verletzung von Konsistenzbedingungen abgebrochen
- **wiederholbar**  
gescheiterte TAs sind u.U wiederholbar, zunächst müssen aber die Änderungen der TA zurückgesetzt werden. Danach kann die TA neugestartet werden.
- **aufgegeben**  
gescheiterte TA ist nicht wiederholbar

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 16

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Zustandsübergänge einer Transaktion

```

graph TD
    P[potentiell] -- inkarnieren --> A[aktiv]
    A -- verdrängen --> W[wartend]
    W -- einbringen --> A
    A -- abbrechen --> G[gescheitert]
    W -- abbrechen --> G
    G -- neustarten --> A
    G -- zurücksetzen --> AU[aufgegeben]
    AU -- zurücksetzen --> G
    A -- beenden --> P
    AB[abgeschlossen] -- festschreiben --> PE[persistent]
    AB -- abbrechen --> G
    G -- zurücksetzen --> AU
  
```

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 17

ETH  
 Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

## Recovery

- Mögliche Szenarien
  - Fehler im Anwendungsprogramm
  - Stromausfall
  - Feuer/Löschwasser/Erdbeben zerstört Server
  - Hardwarefehler (Festplattencrash)
  - Hund des Hausmeisters verrichtet sein Geschäft im Serverraum
  - etc.

Wie garantiert das DBMS trotzdem Atomicity and Durability?

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 18

ETH  
Hörsaal für Technische Hochschulen Zürich  
Swiss Federal Institute of Technology Zurich

## Fehlerklassifikation

1. Lokaler Fehler einer noch nicht persistenten TA
2. Fehler mit Hauptspeicherverlust
3. Fehler mit Hintergrundspeicherverlust

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 19

ETH  
Hörsaal für Technische Hochschulen Zürich  
Swiss Federal Institute of Technology Zurich

## Lokaler Fehler

- Ursachen
  - Fehler im Anwendungsprogramm
  - abort
  - systemgesteuerter Abbruch bei Deadlock
- Abhilfe
  - lokales Undo der Änderungen der TA

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 20

ETH  
Hörsaal für Technische Hochschulen Zürich  
Swiss Federal Institute of Technology Zurich

## Fehler mit Hauptspeicherverlust

- Szenario
  - Stromausfall: Alle im DB-Puffer befindlichen Seiten sind verloren
- Abhilfe
  - **Undo**: alle durch nicht abgeschlossene TA bereits eingebrachten Änderungen müssen rückgängig gemacht werden
  - **Redo**: alle noch nicht eingebrachte Änderungen durch abgeschlossene TAs müssen nachgeholt werden
- Bemerkungen
  - Wir gehen also davon aus, dass die materialisierte Datenbasis nicht zerstört wurde sich allerdings in einem inkonsistenten Zustand befindet
  - Der transaktionskonsistente Zustand wird durch **Undo** und **Redo** wiederhergestellt.

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 21

ETH  
Hörsaal für Technische Hochschulen Zürich  
Swiss Federal Institute of Technology Zurich

## Fehler mit Hintergrundspeicherverlust

- Szenarien
  - head crash
  - Feuer/Erdbeben
  - Hund des Hausmeisters...
  - etc.
- Abhilfe
  - Mehrfache Replikation der Daten und Log-Information an räumlich entfernten Orten

Wie speichern Sie Ihre Masterarbeit?

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 22

ETH  
Hörsaal für Technische Hochschulen Zürich  
Swiss Federal Institute of Technology Zurich

## Schreibstrategien

- **Schreibstrategien** (vgl. Kap 2, Folie 37)
  - **FORCE** (write-through)
    - schreibt geänderte Seiten **abgeschlossener** TAs beim commit atomar zurück
  - **NO FORCE** (write-back)
    - schreibe geänderte Seite erst bei Verdrängung aus dem DB-Puffer zurück
- **Ersetzung von Puffer-Seiten**
  - **NO STEAL**
    - das Ersetzen von **nicht-abgeschlossener** TAs modifizierten Seiten ist verboten
  - **STEAL**
    - jede Seite darf ersetzt werden

	force	¬force
¬steal	¬redo	redo
	¬undo	¬undo
steal	¬redo	redo
	undo	undo

¬force ⇔ redo  
 steal ⇔ undo

19. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 23

ETH  
Hörsaal für Technische Hochschulen Zürich  
Swiss Federal Institute of Technology Zurich

## Einbringstrategien

- Update in Place
  - Jede Seite hat genau eine Heimat auf dem Hintergrundspeicher
  - Der alte Zustand der Seite wird überschrieben
- Twin-Block-Verfahren
  - Für Jede Seite werden zwei Versionen gehalten
- Schattenspeicherkonzept
  - nur geänderte Seiten werden dupliziert
  - weniger Redundanz als beim Twin-Block-Verfahren
- Vgl. Kap 2, Folien 17f

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 24

ETH  
Hörsaal für Informatiker  
ETH National Institute of Technology Zurich

## Hier zugrunde gelegte Systemkonfiguration

- **STEAL**  
jede von nicht abgeschlossenen TAs geänderte Seite darf aus dem Puffer verdrängt werden
- **NO FORCE**  
geänderte Seiten abgeschlossener TAs sind möglicherweise noch nicht auf Platte materialisiert worden
- **UPDATE-IN-PLACE**  
Von jeder Seite gibt es nur eine Kopie auf Platte.
- **Kleine Sperrgranulate**
  - Granularität: Sätze
  - D.h. eine Seite kann gleichzeitig "dreckige" Daten (einer nicht abgeschlossenen TA) und "committed updates" enthalten
  - Dies gilt für Puffer-Seiten und Blöcke (durch STEAL dreckiger Seiten entstanden)

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 25

ETH  
Hörsaal für Informatiker  
ETH National Institute of Technology Zurich

## Protokollierung von Änderungsoperationen

- Struktur der Log-Einträge  
[LSN, TA, PageID, Redo, Undo, PrevLSN]
- LSN (Log Sequence Number)
  - eindeutige ID des Log-Eintrags
  - IDs werden monoton aufsteigend vergeben
  - chronologische Reihenfolge der Einträge kann hiermit ermittelt werden
- TA (Transaktions ID)
  - Kennung der Transaktion, die die Änderung durchgeführt hat
- PageID
  - Seite auf welcher die Änderungen gemacht wurden
  - bei mehreren Seiten: pro Seite ein Log-Eintrag

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 26

ETH  
Hörsaal für Informatiker  
ETH National Institute of Technology Zurich

## Protokollierung von Änderungsoperationen

- Struktur der Log-Einträge  
[LSN, TA, PageID, Redo, Undo, PrevLSN]
- Redo
  - Information wie die Änderung nachvollzogen werden kann
- Undo
  - Information, wie die Änderung rückgängig gemacht werden kann
- PrevLSN (Previous Log Sequence Number)
  - Verweis auf die vorhergegangene LSN
  - wird aus Effizienzgründen benötigt
  - z.B. rückwärts Durchlaufen der LSNs einer TA für lokales Zurücksetzen

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 27

ETH  
Hörsaal für Informatiker  
ETH National Institute of Technology Zurich

## Beispiel einer Log-Datei

Schritt	$T_1$	$T_2$	Log
			[LSN, TA, PageID, Redo, Undo, PrevLSN]
1.	<b>BOT</b>		[#1, $T_1$ , <b>BOT</b> , 0]
2.	$r(A, a_1)$		
3.		<b>BOT</b>	[#2, $T_2$ , <b>BOT</b> , 0]
4.		$r(C, c_2)$	
5.	$a_1 := a_1 - 50$		[#3, $T_1$ , $P_A$ , $A-50$ , $A+50$ , #1]
6.	$w(A, a_1)$		
7.		$c_2 := c_2 + 100$	
8.		$w(C, c_2)$	[#4, $T_2$ , $P_C$ , $C+100$ , $C-100$ , #2]
9.	$r(B, b_1)$		
10.	$b_1 := b_1 + 50$		
11.	$w(B, b_1)$		[#5, $T_1$ , $P_B$ , $B+50$ , $B-50$ , #3]
12.	<b>commit</b>		[#6, $T_1$ , <b>commit</b> , #5]
13.		$r(A, a_2)$	
14.		$a_2 := a_2 - 100$	
15.		$w(A, a_2)$	[#7, $T_2$ , $P_A$ , $A-100$ , $A+100$ , #4]
16.		<b>commit</b>	[#8, $T_2$ , <b>commit</b> , #7]

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 28

ETH  
Hörsaal für Informatiker  
ETH National Institute of Technology Zurich

## Logische vs. physische Protokollierung

- **Physische Protokollierung**
  - Inhalte und Zustände (Bytes) werden protokolliert
  - **before-image** enthält den Zustand vor Ausführung der Operation
  - **after-image** enthält den Zustand nach Ausführung der Operation
- **Logische Protokollierung**
  - Zustandsübergänge (Operationen) werden protokolliert (siehe letztes Beispiel)
  - Das before-Image wird durch Ausführung der Undo-Information aus dem after-image generiert.
  - Das after-image wird durch Ausführung der Redo-Information aus dem before-image generiert.

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 29

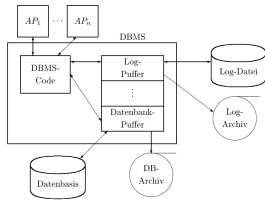
ETH  
Hörsaal für Informatiker  
ETH National Institute of Technology Zurich

## Physiologische Protokollierung

- **Physiologische Protokollierung**
  - Inhalte oder Zustände werden protokolliert
  - Log-Einträge dürfen sich nur auf eine Seite beziehen
  - ARIES benutzt dies

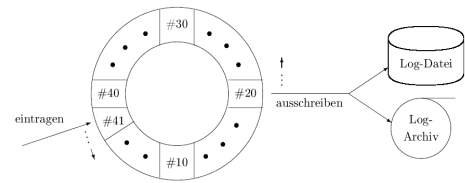
3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 30

## Schreiben der Log-Information



- Wichtig: erst wird der Log-Eintrag geschrieben, dann die Änderung ausgeführt
- Die Log-Information wird zweimal geschrieben
  - Log-Datei für schnellen Zugriff
  - Log-Archiv (notwendig bei Verlust des Hintergrundspeichers)

## Log-Ringpuffer

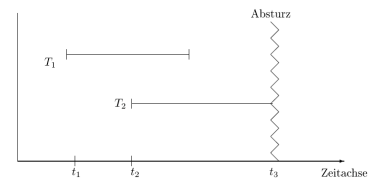


- separat gehalten vom DB-Puffer
- Log-Datei liegt üblicherweise auf separater Festplatte
  - bei vollständigem Hintergrundspeicherverlust: Log-Archiv
  - bei Hintergrundspeicherverlust ohne Verlust der Festplatte, die die Log-Datei hält: Log-Datei

## Das WAL-Prinzip

- 2 Regeln, die beide befolgt werden müssen:
  - Bevor eine Transaktion festgeschrieben (committed) wird, müssen alle zu ihr gehörenden Log-Einträge ausgeschrieben werden.
  - Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, in das temporäre und das Log-Archiv ausgeschrieben werden.
- Es werden grundsätzlich alle Log-Einträge bis zum letzten notwendigen Log-Eintrag geschrieben.

## Wiederanlauf nach einem Fehler

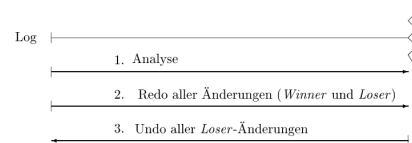


- Transaktionen der Art  $T_1$  müssen hinsichtlich ihrer Wirkung vollständig nachvollzogen werden. Transaktionen dieser Art heissen **Winner**.
- Transaktionen, die wie  $T_2$  zum Zeitpunkt des Absturzes noch aktiv waren, müssen rückgängig gemacht werden. Diese Transaktionen heissen **Loser**.

## Wiederanlauf: 3 Phasen

- Analyse** des Logs
  - Die Log-Datei wird vom letzten Checkpoint aufsteigend durchlaufen
  - commit-Einträge im Log kennzeichnen **Winner-TAs**
  - TAs ohne commit sind **Loser**
- Redo-Phase**
  - alle Änderungen werden in zeitlicher Reihenfolge in die Datenbasis eingebracht (auch die der **Loser**!)
- Undo-Phase**
  - Log-Datei wird absteigend vom Zeitpunkt des Absturzes durchlaufen
  - Winner-Einträge werden ignoriert
  - Für jeden Loser-Eintrag wird Undo ausgeführt (unabhängig von LSN)

## Wiederanlauf: schematisch



- Bemerkung: das funktioniert nur, wenn man das WAL-Prinzip eingehalten hat

ETH  
Hörsaal für Informatiker  
Swiss Federal Institute of Technology Zurich

## 1. Analyse

- Ziel: Loser-Transaktionen berechnen
- Log sequentiell durchlaufen
- Alle Einträge mit BOT in Liste TT aktiver Transaktionen aufnehmen
- Commit-Einträge im Log kennzeichnen Winner und werden aus TT entfernt
- Alle anderen TAs sind Loser
- Ergebnis: Liste TT aktiver nicht durch commit beendeter Transaktionen

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 37

ETH  
Hörsaal für Informatiker  
Swiss Federal Institute of Technology Zurich

## 2. Redo

- Ziel: Zustand der DB zum Zeitpunkt des Crash herstellen
- Alle Log-Einträge sequentiell aufsteigend durchlaufen

Für alle Log-Einträge e:  
 Lade Seite p mit ID e.PageID  
 Falls e.PageID.LSN < e.LSN:  
 $p = p + \text{after-image von } e$   
 $p.LSN = e.LSN$  ← Wozu ist dies gut?  
 Sonst  
 SKIP!

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 38

ETH  
Hörsaal für Informatiker  
Swiss Federal Institute of Technology Zurich

## 3. Undo

- Ziel: Änderungen aller Loser-TAs rückgängig machen
- Alle Log-Einträge sequentiell absteigend durchlaufen

Für alle Log-Einträge e:  
 Falls e.TA in TT: //d.h. Eintrag bezieht sich auf Loser-TA  
 Lade Seite p mit ID e.PageID  
 $p = p + \text{before-image von } e$   
 Erzeuge Kompensationseintrag  
 Sonst  
 SKIP! //d.h. Änderungen der Winner werden nicht rückgängig gemacht

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 39

ETH  
Hörsaal für Informatiker  
Swiss Federal Institute of Technology Zurich

## Fehlertoleranz des Wiederanlaufs

- Was passiert, wenn beim Wiederanlauf der Strom ausfällt?
- Lösung: man muss garantieren, dass die Redo- und Undo-Phase idempotent sind, d.h. für jede Aktion a gilt:
  - $\text{UNDO}(\text{UNDO}(\dots(\text{UNDO}(a))\dots)) = \text{UNDO}(a)$
  - $\text{REDO}(\text{REDO}(\dots(\text{REDO}(a))\dots)) = \text{REDO}(a)$
- Redo-Phase
  - LSN des Eintrags, der ausgeführt wurde, wird in die Seite eingetragen
  - bei einem erneuten Redo wird somit eine zweite Ausführung verhindert
- Undo-Phase
  - Kompensationseinträge im Log (CLR, compensation log record)

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 40

ETH  
Hörsaal für Informatiker  
Swiss Federal Institute of Technology Zurich

## Kompensationseinträge im Log

- Für jede ausgeführte Undo-Operation wird ein neuer CLR angelegt: [LSN, TA, PageID, Redo, Undo, PrevLSN, UndoNextLSN]
- Redo = Undo des ursprünglichen Log-Eintrags
- Bei einem neuen Wiederanlauf wird "Undo" in der Redo-Phase ausgeführt!
- CLR's werden in der Undo-Phase nicht ausgeführt (obwohl Loser).
- Es wird mit Hilfe von UndoNextLSN zum nächsten Log-Eintrag gesprungen.

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 41

ETH  
Hörsaal für Informatiker  
Swiss Federal Institute of Technology Zurich

## Logeinträge nach abgeschlossenem Wiederanlauf

- [#1, T1, BOT, 0]
- [#2, T2, BOT, 0]
- [#3, T1, PA, A=50, A+=50, #1]
- [#4, T2, PC, C+=100, C=100, #2]
- [#5, T1, PB, B+=50, B=50, #3]
- [#6, T1, commit, #5]
- [#7, T2, PA, A=100, A+=100, #4]
- [#7, T2, PA, A+=100, #7, #4]
- [#4, T2, PC, C=100, #7, #2]
- [#2, T2, -, -, #4, 0]

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 42

ETH  
Hörsaalgebäude, Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zürich

## Partielles Zurücksetzen einer Transaktion

Historie: 1 → 2 → 3 → 4 → 5 → ...

Log: #1 → #2 → #3 → #4 → #4' → #3' → #5 → ...

- Schritt 3 und 4 werden zurückgenommen
- notwendig für die Realisierung von Sicherheitspunkten innerhalb von TAs

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 43

ETH  
Hörsaalgebäude, Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zürich

## Sicherungspunkte (Checkpoints)

- Achtung: **savepoint** ≠ **checkpoint**
- Problem: Log-Datei wird mit zunehmender Betriebszeit des DBMS immer grösser
- Folge: Wiederanlauf dauert immer länger
- Lösung: markiere einen Sicherungspunkt im Log, über den man beim Wiederanlauf nicht hinausgehen muss

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 44

ETH  
Hörsaalgebäude, Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zürich

## Transaktionskonsistente Sicherungspunkte

Sicherungspunkt  $S_{i-1}$  | Sicherungspunkt  $S_i$  ausgeschrieben | Sicherungspunkt  $S_{i+1}$  geschrieben

Absturz

Zeitchase

- Idee: alle neuen TAs, die nach  $S_i$  eintreffen, werden suspendiert bis alle laufenden TAs committed haben und materialisiert sind
- Beispiel:
  - $T_1$  und  $T_2$  werden materialisiert
  - Alle Log-Datei kann gelöscht werden
  - $T_3$  wechselt nach aktiv
- Vorteil: Nach Schreiben des Sicherungspunktes kann die Log-Datei gelöscht und neubegonnen werden

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 45

ETH  
Hörsaalgebäude, Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zürich

## Aktionskonsistente Sicherungspunkte

Systemabsturz

Sicherungspunkt

Zeitchase

- Alle **elementaren** Änderungsoperationen werden abgeschlossen und materialisiert
- Vorteil: bei Wiederanlauf keine Redo-Information älter als Sicherungspunkt notwendig
- Aber: Undo kann über Sicherungspunkt hinausgehen!

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 46

ETH  
Hörsaalgebäude, Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zürich

## Unschärfe (Fuzzy) Sicherungspunkte

- Idee: schreibe **nicht** die modifizierten Seiten auf Platte sondern nur die Liste der Seiten-IDs in die Log-Datei
- Für jede schmutzige Seite wird die LSN, die die Seite schmutzig werden liess, mit ins Log geschrieben.
- Die kleinster dieser LSNs ergibt den Wert **minDirtyPageLSN**
- minDirtyPageLSN** bezeichnet den Start-Punkt der Redo-Phase
- Achtung:
  - keine schmutzigen Seiten werden auf den Hintergrundspeicher geschrieben werden
  - deswegen möglicherweise sehr alte Seiten im DB-Puffer (hot-spots)
  - Folge: In der Redo-Phase muss die Log-Datei (fast) komplett durchlaufen werden => kein Gewinn durch den Sicherungspunkt
  - Abhilfe: Hintergrund-Thread der schmutzige Seiten periodisch herausschreibt

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 47

ETH  
Hörsaalgebäude, Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zürich

## Wiederanlauf bei Unterschiedlichen Sicherungspunkten

Sicherungspunkt

Log

(a) **transaktionskonsistent**

Analysis →

Redo →

Undo ←

(b) **aktionskonsistent**

MinLSN

Analysis →

Redo →

Undo ←

(c) **unschärf (fuzzy)**

MinDirtyPageLSN

MinLSN

Analysis →

Redo →

Undo ←

3. Juni 2005 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 48

## ARIES

- Algorithms for Recovery and Isolation Exploiting Semantics
- State-of-the art für recovery in vielen Produkten (DB2, MS SQL Server, Cloudscape siehe [http://www.almaden.ibm.com/u/mohan/ARIES\\_Impact.html#systems](http://www.almaden.ibm.com/u/mohan/ARIES_Impact.html#systems))
- Literatur: C. Mohan, Donald J. Haderle, Bruce G. Lindsay, Hamid Pirahesh, Peter M. Schwarz: ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. ACM Trans. Database Syst. 17(1): 94-162 (1992)
- Analog zu der in dieser Stunde entwickelten Methode
- Features
  - fuzzy Sicherungspunkte
  - Physiologisches Logging
  - Feingranulares (Records) Locking

## Nächste Woche

Mehrbenutzersynchronisation

