BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Transaktionsmanagement

Medieninformatik Bachelor
Modul 9:
Datenbanksysteme

Prof. Dr. Petra Sauer, Prof. Dr. habil. Alexander Löser

Sie erinnern sich: Ihre Aufgaben

- **Aufgabe 1 Anfragen & Modellierung“**
  Denken Sie mal darüber nach, welche Anfragen Sie an die AOL Daten stellen möchten. Bitte Sie bitte ein logisches und physisches Schema zur Beantwortung dieser Anfragen.
- **Aufgabe 2 „SQL und Anfrageausführung“**
  Bitte formulieren Sie für Ihre Analyseideen aus 1.) die SQL Anfragen. Sie verstehen auch Möglichkeiten der Anfrageausführung bzw. Optimierung.
- **Aufgabe 3 „Datenintegration“**
  Zur Ausführung der Ausführung fehlen Ihnen noch externe Daten, z.B. aus dem Internet Archive, DMOZ oder Freebase.org. Bitte ergänzen Sie Ihr Schema und die Datenbasis.
- **Aufgabe 4 „Analyse, Erkenntnisgewinn und Wert“**
  Stellen Sie in 5 Minuten die wichtigsten Erkenntnisse aus den Daten vor. Bewerten Sie den Erkenntnisgewinn, z.B. gegenüber Ihren Kommilitonen oder der Literatur! Welche Erkenntnisse hätten einen kommerziellen Wert?

Die Themen

- Was sind Datenbanken?
  - Motivation, Historie, Datenunabhängigkeit, Einsatzgebiete
- Datenbankentwurf im ER-Modell & Relationaler Datenbankentwurf
  - Entities, Relationships, Kardinalitäten, Diagramme
  - Relationales Modell, ER -> Relational, Normalformen, Transformationseigenschaften
- Relationale Algebra & SQL
  - Kriterien für Anfragesprachen, Operatoren, Transformationen
  - SQL DDL, SQL DML, SELECT … FROM … WHERE …
- Datenintegration & Transaktionsverwaltung
  - JDBC, Cursor, ETL
  - Mehrbenutzerbetrieb, Serialisierbarkeit, Sperrprotokolle, Fehlerbehandlung, Isolationsebenen in SQL
- Ausblick
  - Map/Reduce, HDFS, Hive …
  - Wert von Daten

Gliederung

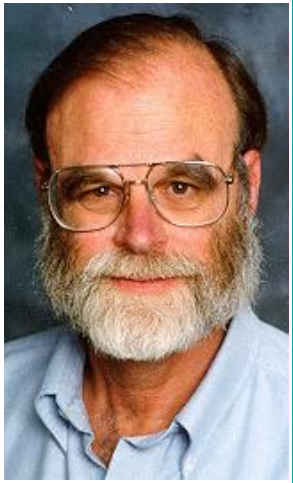
- **Einführung und Begriffe**
- Transaktionsverwaltung
- Scheduling von Transaktionen
- Sperrverfahren
- Recovery

Transaktionen: Beispielszenarien

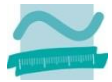
- Platzreservierung für Flüge gleichzeitig aus vielen Reisebüros
  - Platz könnte mehrfach verkauft werden, wenn mehrere Reisebüros den Platz als verfügbar identifizieren.
- Überschneidende Konto-Operationen einer Bank
- Statistische Datenbankoperationen
  - Ergebnisse sind verfälscht, wenn während der Berechnung Daten geändert werden.

Transaktionen – Historie

- In alten DBMS kein Formalismus über Transaktionen
  - Nur Tricks
- Erster Formalismus in den 80ern
  - *System R* unter Jim Gray
  - Erste (ineffiziente) Implementierung
  - ACM Turing Award
    - *For seminal contributions to database and transaction processing research and technical leadership in system implementation from research prototypes to commercial products.*
- ARIES Project (IBM Research)
  - Alle Details
  - Effiziente Implementierungen
  - C. Mohan
- Transaktionen auch in verteilten Anwendungen und Services



Gliederung



- Einführung und Begriffe
- **Transaktionsverwaltung**
- Scheduling von Transaktionen
- Sperrverfahren
- Recovery

ACID – Prinzip

Transaktion = ununterbrochene Folge von Aktionen, welche die DB von einemkonsistenten Zustand in einen neuen konsistenten Zustand überführt.

ACID-Eigenschaften einer Transaktion:

- **Atomarität**
- **Konsistenzerhaltung**
- **Isolation**
- **Dauerhaftigkeit**

ACID befreit DB-Anwendungsentwicklung von allen Aspekten der Nebenläufigkeit und des Fehlerfalls

→ Logischer Einbenutzerbetrieb

ACID – Prinzip

- **Atomarität**: alle Aktionen einer Transaktion werden vollständig oder gar nicht ausgeführt.

  → **Recovery**

- **Konsistenzerhaltung** (consistency): Transaktion hinterlässt eine konsistente DB (alle definierten Integritätsbedingungen werden eingehalten)
  - Zwischenzustände können inkonsistent sein
  - Ist Endzustand nicht konsistent, muss TA zurückgesetzt werden

  → **Integritätssicherung**

ACID – Prinzip

- **Isolation**: Transaktion läuft isoliert von anderen Transaktionen ab (die Transaktion nutzt nur „gesicherte“ Daten der DB), parallel laufende Transaktionen haben keinen Einfluss

  → **Mehrbenutzersynchronisation, Scheduling**

- **Dauerhaftigkeit** (Durability): wurde Transaktion erfolgreich beendet, so muss das DBMS sichern, dass alle Änderungen dauerhaft in der DB gespeichert werden (selbst bei nachträglichem Auftreten von Fehlern)

  → **Recovery**

Gliederung

- Einführung und Begriffe
- Transaktionsverwaltung
- **Scheduling von Transaktionen**
- Sperrverfahren
- Recovery

Schedule und vereinfachtes Modell

Schedule: „Ablaufplan“ für Transaktion, bestehend aus Abfolge von Transaktionsoperationen

Repräsentation von Datenbankänderungen einer Transaktion

- read(A,x): weise den Wert des DB-Objektes A der Variablen x zu
- write(x, A): speichere den Wert der Variablen x im DB-Objekt A

Beispiel einer Transaktion T:

- read(A, x); x := x 200; write(x, A);
- read(B, y); y := y + 100; write(y, B);

Ausführungsvarianten für zwei Transaktionen T1, T2:

- Serieller Ablaufplan, etwa T1 vor T2
- „gemischter Ablaufplan“, etwa abwechselnd Schritte von T1 und T2

Notation

- Beginn einer Transaktion: Begin-of-Transaction-Kommando **BOT** (in SQL implizit!)
- **commit**: die Transaktion soll erfolgreich beendet werden
- **abort**: die Transaktion soll abgebrochen werden

Transaktionen in Oracle SQL

Transaction Begins

```
UPDATE savings_accounts
     SET balance = balance - 500
     WHERE account = 3209;
```

Decrement Savings Account

```
UPDATE checking_accounts
     SET balance = balance + 500
     WHERE account = 3208;
```

Increment Checking Account

```
INSERT INTO journal VALUES
     (journal_seq.NEXTVAL, '1B'
     3209, 3208, 500);
```

Record in Transaction Journal

```
COMMIT WORK;
```

End Transaction

Transaction Ends

In Oracle beginnt eine Transaktion mit dem ersten ausgeführten SQL Statement das Daten manipuliert oder mit SET TRANSAKTION ….

Eine Transaktion endet in Oracle 11g entweder explizit durch COMMIT or ROLLBACK oder implizit beim nächsten DDL statement.

http://docs.oracle.com/cd/B28359_01/server.111/b28318/transact.htm#i974
http://docs.oracle.com/cd/E28271_01/server.1111/e25789/consist.htm

Probleme im Mehrbenutzerbetrieb

- Inkonsistentes Lesen: *Nonrepeatable Read*
- Abhängigkeiten von nicht freigegebenen Daten: *Dirty Read*
- Berechnungen auf unvollständigen Daten: *Phantom-Problem*
- Verlorengegangenes Ändern: *Lost Update*

Nonrepeatable Read (RWR Konflikt)

- Nicht-wiederholbares Lesen
- Beispiel:
  - Zusicherung: x = A + B + C
    am Ende der Transaktion T_1
  - x, y, z seien lokale Variablen

T_1	T_2
`read(A,x)`	
	`read(A,y)`
	`y:=y/2`
	`write(y,A)`
	`read(C,z)`
	`z:=z+y`
	`write(z,C)`
	`commit`
`read(B,y)`	
`x:=x+y`	
`read(C,z)`	
`x:=x+z`	
`commit`	

Problem: A hat sich im Laufe der Transaktion geändert.
x = A + B + C gilt nicht mehr

Beispiel nach Kai-Uwe Sattler (TU Ilmenau)

Das Phantom-Problem (RWR Konflikt)

T_1	T_2

`SELECT COUNT(*)`
`INTO X`
`FROM Mitarbeiter`	

	`INSERT INTO Mitarbeiter` `VALUES (‚Meier', 50000, …)`
	`commit`
`UPDATE Mitarbeiter` `SET Gehalt = Gehalt +10000/X`	
`commit`	

Problem: Meier geht nicht in die Mitarbeiterzählung ein. Meier ist das Phantom.

Das Phantomproblem ist ein Sonderfall des non-repeatable read.

Beispiel nach Kai-Uwe Sattler (TU Ilmenau)

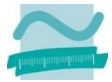
Dirty Read (WR Konflikt)

T_1	T_2
`read(A,x)`	
`x:=x+100`	
`write(x,A)`	
	`read(A,x)`
	`read(B,y)`
	`y:=y+x`
	`write(y,B)`
	`commit`
`abort`	

Problem: T_2 liest den veränderten A-Wert, diese Änderung ist aber nicht endgültig, sondern sogar ungültig.

Folie nach Kai-Uwe Sattler (TU Ilmenau)

Lost Update (WW Konflikt)



T_1	T_2	A
`read(A,x)`		10
	`read(A,x)`	10
`x:=x+1`		10
	`x:=x+1`	10
`write(x,A)`		11
	`write(x,A)`	11

Problem: Die Erhöhung von T_1 wird nicht berücksichtigt.

Folie nach Kai-Uwe Sattler (TU Ilmenau)

Isolationskonzepte

Vermeidung / in Kauf nehmen der Anomalien durch Wahl der Konsistenzebene (Isolationskonzept)

Anomalie \ Isolationsstufe	Dirty Read	Non-Repeatable Read	Phantome
Read Uncommitted	+	+	+
Read Committed	-	+	+
Repeatable Read	-	-	+
Serializable (Standard in SQL 92)	-	-	-

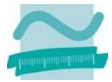
Was sind *serialisierbare* Schedules?

Create your Own Exam: Transaktionen

- Bitte erstellen Sie eine Multiple Choice Aufgabe zum Thema Transaktionen
  - Formulieren Sie eine Frage und 3 Antworten (A, B, C)
  - Davon sollte mindestens eine Antwort richtig und mindestens eine Antwort falsch sein
- Geben Sie die Aufgabe an Ihren rechten Nachbarn. Diskutieren Sie gemeinsam und markieren Sie die richtigen Lösungen
- Geben Sie am Ende der Vorlesung Ihre Aufgabe bei mir ab

5 min

Gliederung



- Einführung und Begriffe
- Transaktionsverwaltung
- Scheduling von Transaktionen
- **Sperrverfahren**
- Recovery

Scheduling von Transaktionen

Eine verschränkte Ausführung mehrerer Transaktionen heißt serialisierbar, wenn ihr Effekt identisch zum Effekt einer (beliebig gewählten) seriellen Ausführung dieser Transaktionen ist.

- **Serieller Schedule**
  - alle Aktionen unterschiedlicher Transaktionen stehen streng hintereinander, keine Überlappung
- **Äquivalente Schedules**
  - Der Effekt der Ausführung des einen Schedules ist identisch mit dem Effekt der Ausführung eines anderen Schedules
- **Serialisierbarer Schedule (Wichtigster Schedule)**
  - Ein Schedule, der äquivalent zu einem seriellen Schedule ist. Jeder Schedule, der denselben Effekt erzielt, wie ein serieller Schedule, ist akzeptabel.

Weitere Begriffe: Konfliktserialisierbarkeit, Viewserialisierbarkeit

Beispiele für Schedules

Serieller Schedule

Schritt	T_1	T_2
1.	**BOT**	
2.	read(*A*)	
3.	write(*A*)	
4.	read(*B*)	
5.	write(*B*)	
6.	**commit**	
7.		**BOT**
8.		read(*C*)
9.		write(*C*)
10.		read(*A*)
11.		write(*A*)
12.		**commit**

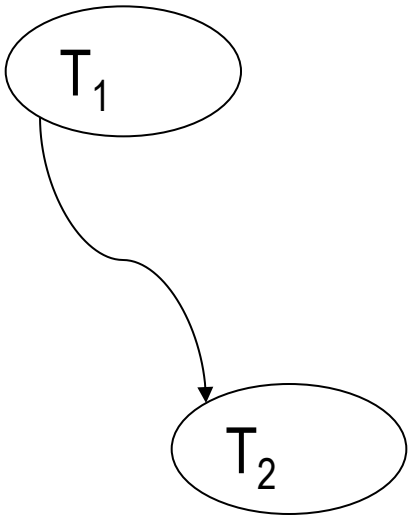
Serialisierbarer Schedule

Schritt	T_1	T_2
1.	**BOT**	
2.	read(*A*)	
3.		**BOT**
4.		read(*C*)
5.	write(*A*)	
6.		write(*C*)
7.	read(*B*)	
8.	write(*B*)	
9.	**commit**	
10.		read(*A*)
11.		write(*A*)
12.		**commit**

Exkurs: Schedules und Konfliktgraphen

Schritt	T_1	T_2
1.	**BOT**	
2.	read(*A*)	
3.		**BOT**
4.		read(*C*)
5.	write(*A*)	
6.		write(*C*)
7.	read(*B*)	
8.	write(*B*)	
9.	**commit**	
10.		read(*A*)
11.		write(*A*)
12.		**commit**

Zyklenfreier Konfliktgraph



Serialisierbarer Schedule

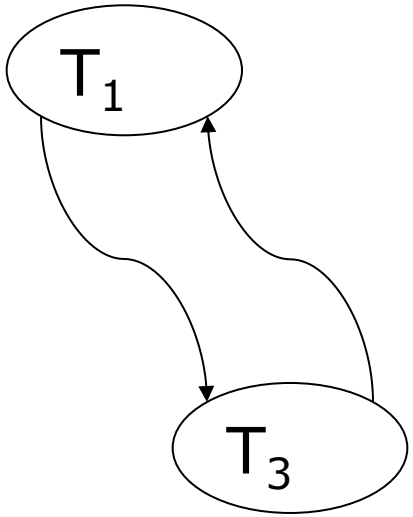
Beispiel: Alfons Kemper (TU München)

Konfligierende Schedules

Nicht serialisierbar („konfligierend“)

Schritt	T_1	T_3
1.	**BOT**	
2.	read(*A*)	
3.	**write(*A*)**	
4.		**BOT**
5.		**read(*A*)**
6.		write(*A*)
7.		read(*B*)
8.		**write(*B*)**
9.		**commit**
10.	**read(*B*)**	
11.	write(*B*)	
12.	**commit**	

Konfliktgraph mit Zyklen



Beispiel: Alfons Kemper (TU München)

Sperrenbasierte Synchronisation

Idee von Sperrverfahren:

- Konflikte von TA vermeiden durch Verzögerung

Fundamentalsatz des Sperrens (Eswaran, CACM 11/76, S. 624-663)

- Auf ein Objekt kann nur zugegriffen werden, wenn es **vor dem Zugriff** mit einer Sperre belegt wurde. Zugriffs- und Sperrart müssen passend sein (w(x) -> x-lock, r(x) -> s-lock)
- Höherstufen von Sperren ist möglich (s-lock -> x-lock)
- Sperren anderer Transaktionen beachten! (Sperranforderung, die mit gesetzter Sperre unverträglich ist, muss auf Freigabe der Sperre warten)
- eine Transaktion fordert keine Sperre an, die sie besitzt bzw. besaß (kein wiederholtes Sperren nach Freigabe)
- Spätestens bei **Transaktionsende** gibt TA alle Sperren zurück

Typen von DML-Sperren - Klassifikation:

- **Nach den gesperrten Objekten**:
  - Datensatzsperren
  - Tabellensperren
- **Nach der Exklusivität der Sperre**:
  - **Exklusive** Sperre (**x-Sperre**)

    Kein weiterer schreibender oder reservierender Zugriff neben der Transaktion!

    Nutzen: für Transaktionen, die Daten verändern
  - **Geteilte** bzw. teilbare Sperre (share lock – **s-Sperre**)

    mehrere s-locks gleichzeitig durch verschiedene Transaktionen möglich

Achtung: Alle DML-Operationen haben x-locks zur Folge!
Achtung: oftmals Kombination von Datensatz- und Tabellensperren (automatisch mit DS-Sperren gesetzt)

SX-Kompatibilitätsmatrix

- **Annahmen:**
  - **Transaktionen** T_1 und T_2
  - T_1 besitzt Sperre auf Objekt a
  - T_2 fordert Sperre auf Objekt a
  - **+ Sperren sind kompatibel**
  - **- Sperren sind inkompatibel**

T_1 \ T_2	T_2 fordert S-Sperre für Objekt a	T_2 fordert X-Sperre für Objekt a
Objekt a gesperrt mit S-Sperre	+	-
Objekt a gesperrt mit X-Sperre	-	-

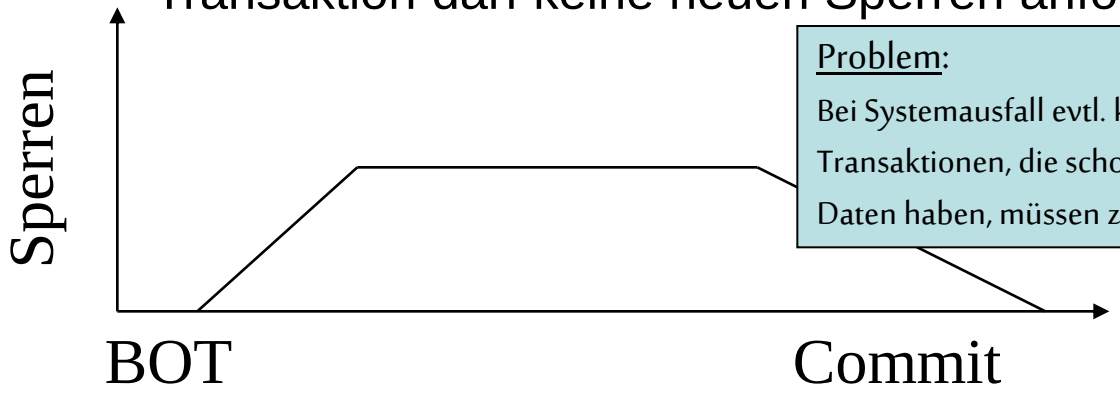
Sperrverfahren

2 – Phasen – Sperrverfahren:

→ Protokoll zur Erzeugung von Schedules

- 1. Phase: **Wachstumsphase**
  - Transaktion fordert Sperren an
  - Transaktion gibt keine Sperren frei

- 2. Phase: **Schrumpfungsphase**
  - Transaktion gibt Sperren frei
  - Transaktion darf keine neuen Sperren anfordern



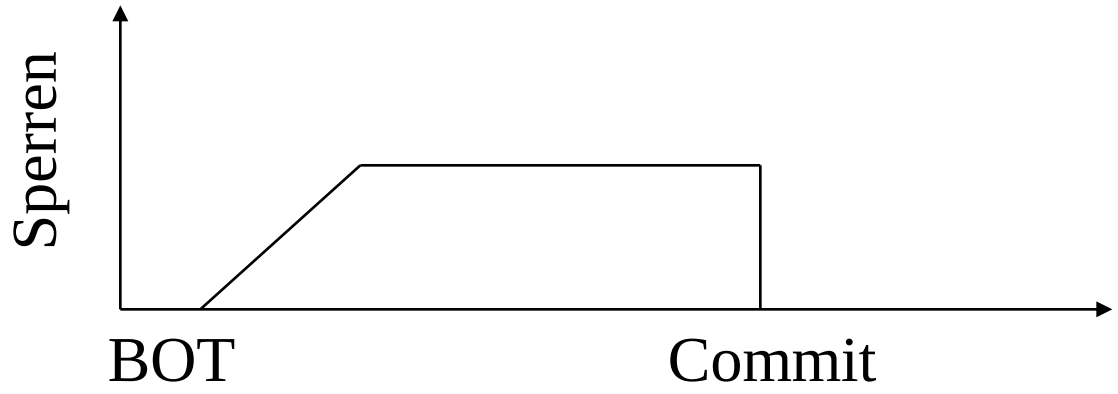
Problem:
Bei Systemausfall evtl. kaskadierender Abbruch:
Transaktionen, die schon freigegebene
Daten haben, müssen zurückgesetzt werden

Striktes 2 – Phasen – Sperrverfahren:

→Protokoll zur Erzeugung von serialisierbaren Schedules

→ in den DBMS i.d.R. umgesetzt

- Idee: Vermeidung von kaskadierendem abort
- Vorgehen:
  - Jede Transaktion fordert s-Sperre (*shared lock*) vor dem Lesen an, x-Sperre (*exclusive lock*) vor dem Schreiben
  - Alle Sperren werden bei Beendigung der Transaktion zu einem Zeitpunkt freigegeben



Zurücksetzen einer Transaktion

- Kaskadierender Abort:
  - Wenn eine Transaktion *Ti* zurückgesetzt wird (abort), müssen alle ihre Aktionen rückgängig gemacht werden (undo).
  - Außerdem muß eine Transaktion *Tj* zurückgesetzt werden, das ein Objekt liest, das zuletzt von *Ti* geschrieben wurde
- Lösung durch Sperrenfreigabe nur zur Commit-Zeit
  - Wenn *Ti* ein Objekt schreibt, kann *Tj* dieses nur lesen, nachdem *Ti* commit gemacht hat
- Undo aller Aktionen beim Abort:
  - Führen von Log-Daten einer Transaktion: Aufzeichnung aller Write-Aktionen
  - Auch nutzbar für die Wiederherstellung der Daten bei System-Crah (Recovery): alle Transaktionen, die zum Zeitpunkt des System-Crashs aktiv waren, werden beim Wiederanlauf zurückgesetzt

Gliederung

- Einführung und Begriffe
- Transaktionsverwaltung
- Scheduling von Transaktionen
- Sperrverfahren
- **Recovery**

Fehlerklassen von DBS

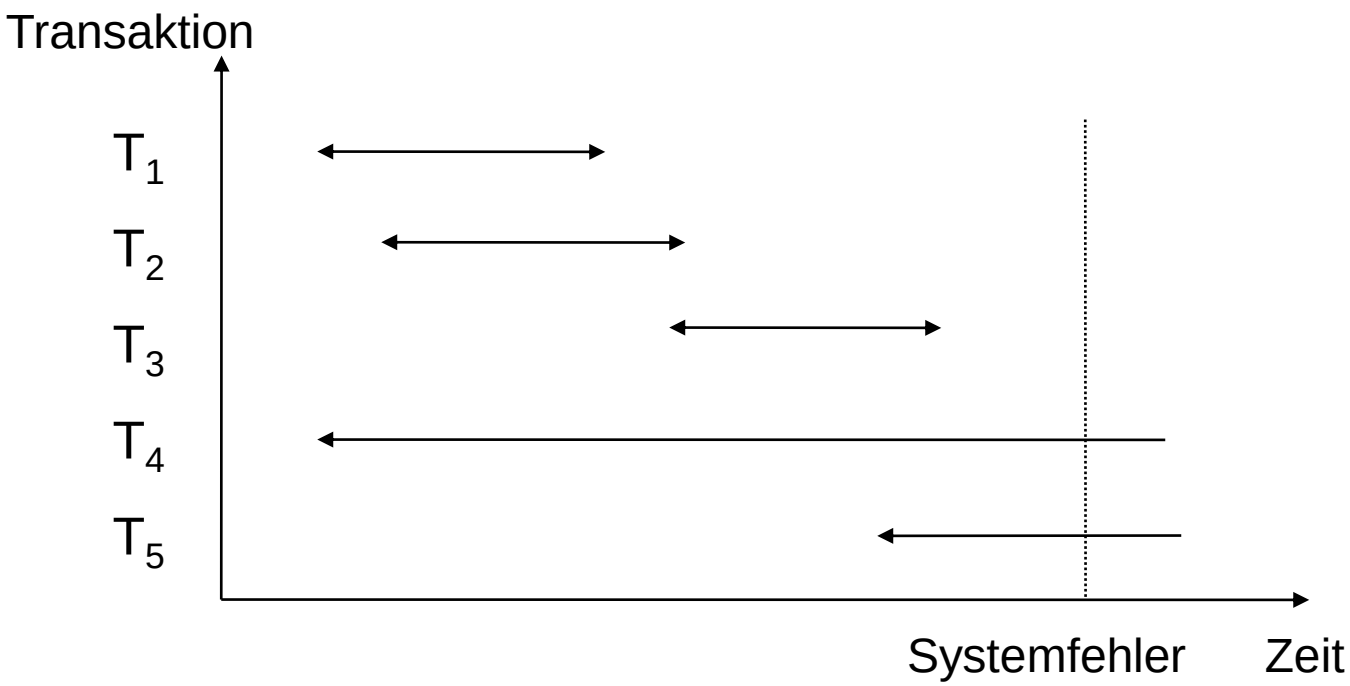
- **Transaktionsfehler**
  TA erreicht nicht **commit**
  alle Effekte der TA müssen rückgängig gemacht werden
  Auftreten: Programmfehler, Opfer-TA bei Deadlock etc.

- **Systemfehler**
  Fehler im DBMS, im BS, in HW
  Stromausfall etc.
  Hauptspeicherinhalte sind verloren gegangen

- **Mediafehler**
  Headcrash etc.
  Teile des „stabilen“ Speichers sind verloren gegangen
  (nachfolgend nicht betrachtet)

Hoffentlich gibt es ein aktuelles, vollständiges **Backup**!

Transaktionen: Verhalten bei Systemabsturz



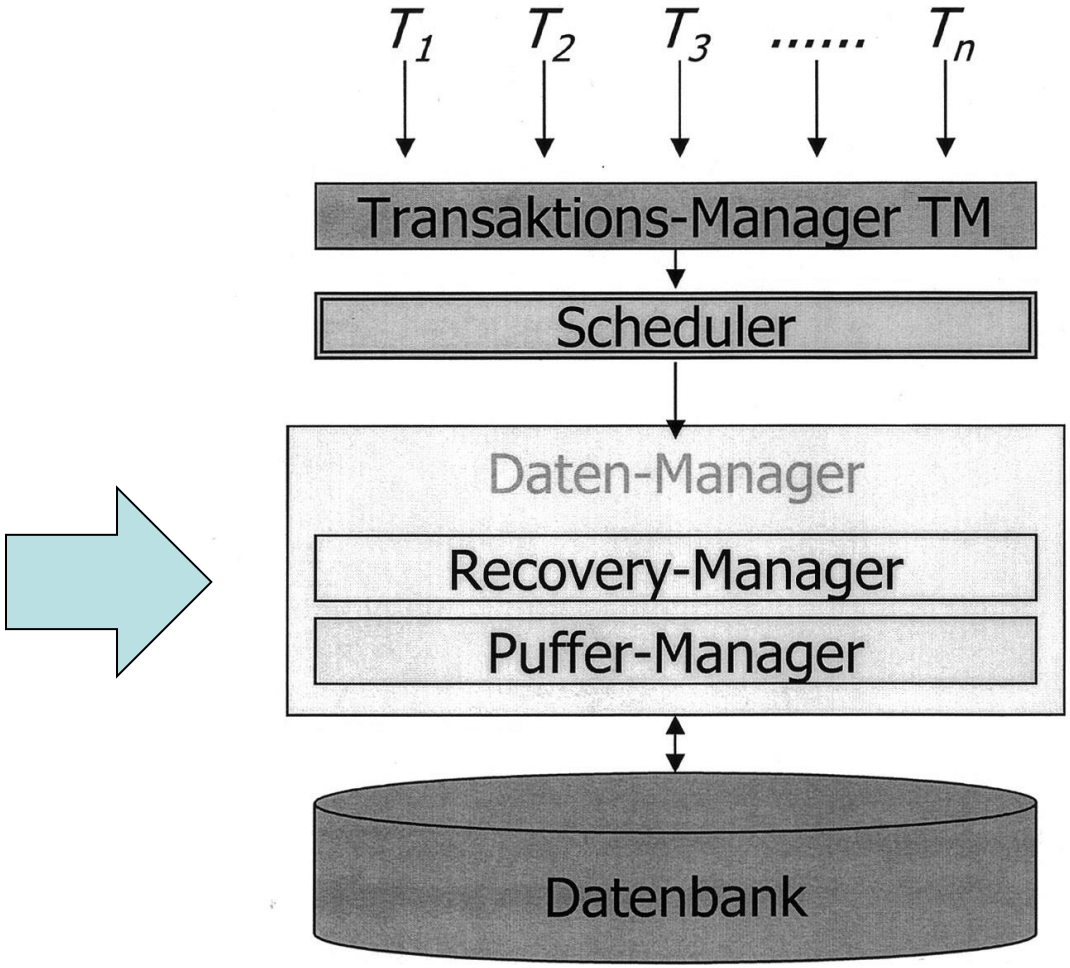
- Inhalt des flüchtigen Speichers zum Zeitpunkt T-Fehler ist unbrauchbar !

Zustände der Transaktionen:

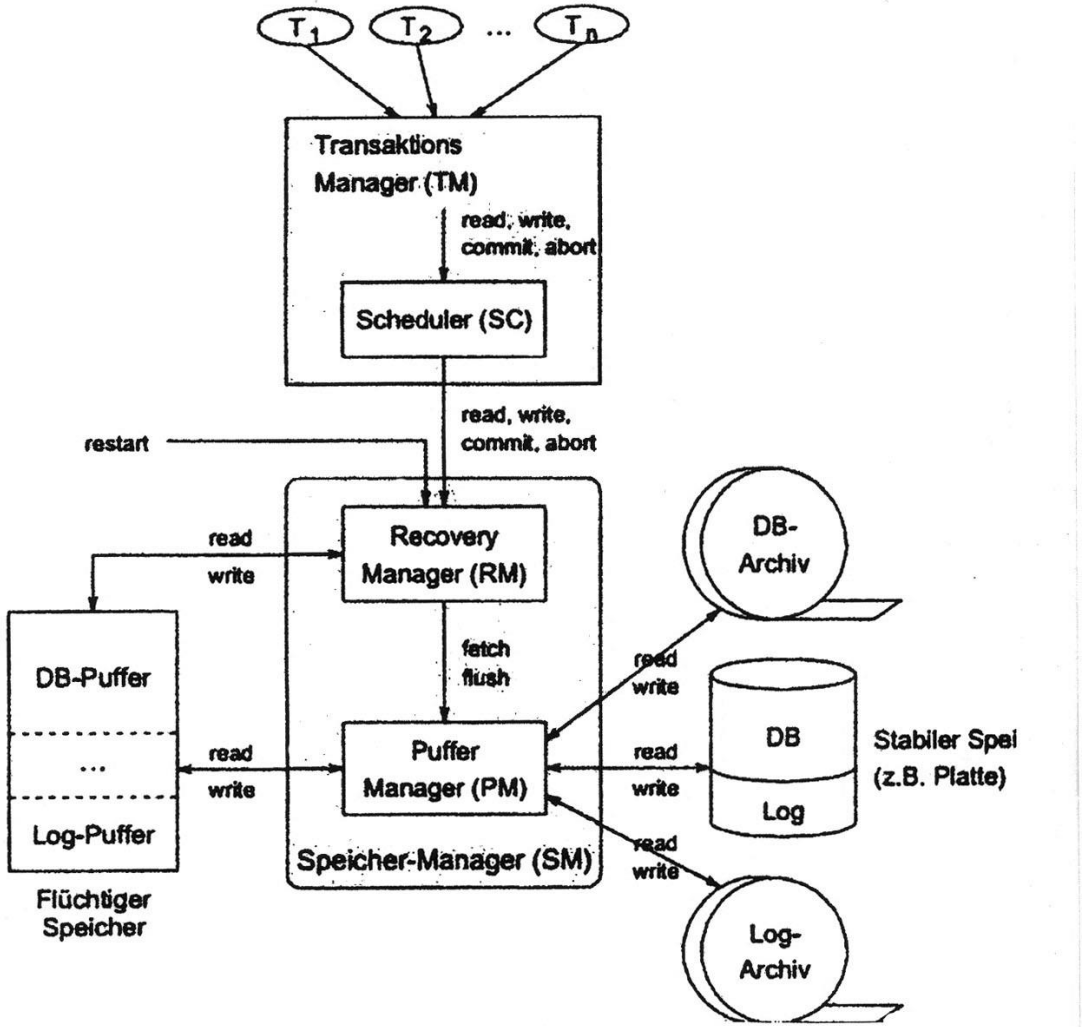
- zum Fehlerzeitpunkt noch aktive Transaktionen (T2 und T4)
- bereits vor dem Fehlerzeitpunkt beendete Transaktionen (T1, T3 und T5)

Erforderlich: Zurücksetzen der betroffenen Transaktionen.

Beteiligte Systemkomponenten:



Recovery-Manager



Recovery und Log-Datei

- Logisches Logging: Speicherung von Änderungsaktionen mit Parametern
  - Log-Sätze zur Beschreibung von Änderungen: alter und neuer Wert
  - Log-Satz muß auf Platte gespeichert werden vor der geänderten Seite
  - Transaktionsbeginn, Transaktions-Commit sowie Transaktions-Rollback
- Logs sind oft *dupliziert* und *archiviert* auf stabilem Speicher

Wiederherstellung nach einem Crash

- 3 Phasen im Recovery-Algorithmus (*Aries*-Ansatz)
  - Analyse-Lauf
    - Sequentielles Lesen (Scan) der Log-Datei vom letzten Checkpoint bis zu ihrem Ende
    - Bestimme die zum Checkpoint-Zeitpunkt laufenden Transaktionen
    - Bestimme davon Gewinner (TA mit Commit-Eintrag im Log) und Verlierer (TA mit Rollback-Satz bzw. ohne Commit-Satz)
    - Ermittle die Seiten, die nach dem Checkpoint geändert wurden (dirty pages)
  - Redo-Lauf
    - Wiederholung der Änderungen, welche noch nicht in den betroffenen Seite vorliegen (Lese Log von hinten nach vorne)
    - Somit sichergestellt, daß alle protokollierten Updates wirksam und auf Platte geschrieben sind
  - Undo-Lauf
    - Zurücksetzen der Verlierer-Transaktionen, die während des Crash aktiv waren (durch Schreiben des *before value*, der im Log-Satz steht)
    - Lesen des Logs von vorne nach hinten bis zum Beginn der ältesten Transaktion, die beim letzten Checkpoint aktiv war

Zusammenfassung

- Synchronisation (Concurrency Control) und Recovery gehören zu den wichtigsten Funktionen des DBMS t werden
- Benutzer brauchen sich (fast) nicht um Nebenläufigkeit zu kümmern
  - System erzeugt automatisch Anforderungen und Freigaben von Sperren (*Lock Management*)
  - System plant Aktionen unterschiedlicher Transaktionen in einer Weise, um sicherzustellen, daß die resultierende Ausführung äquivalent zu irgendeiner seriellen Transaktionsausführung ist (*Scheduling*)
- *Write-ahead-Logging* (WAL) wird genutzt, um Aktionen von zurückgesetzten Transaktionen rückgängig zu machen und das System nach einem Crash wieder in einen konsistenten Zustand zu bringen
  - *Konsistenter Zustand*: Nur die Effekte von beendeten Transaktionen sind sichtbar

Zusammenfassung und Ausblick

- Einführung und Begriffe
- Transaktionsverwaltung
- Scheduling von Transaktionen
- Sperrverfahren
- Recovery

In der nächsten Veranstaltung:

- Data Warehouse

Sie erinnern sich: Ihre Aufgaben

- **Aufgabe 1 Anfragen & Modellierung“**
  Denken Sie mal darüber nach, welche Anfragen Sie an die AOL Daten stellen möchten. Bitte Sie bitte ein logisches und physisches Schema zur Beantwortung dieser Anfragen.
- **Aufgabe 2 „SQL und Anfrageausführung“**
  Bitte formulieren Sie für Ihre Analyseideen aus 1.) die SQL Anfragen. Sie verstehen auch Möglichkeiten der Anfrageausführung bzw. Optimierung.
- **Aufgabe 3 „Datenintegration“**
  Zur Ausführung der Ausführung fehlen Ihnen noch externe Daten, z.B. aus dem Internet Archive, DMOZ oder Freebase.org. Bitte ergänzen Sie Ihr Schema und die Datenbasis.
- **Aufgabe 4 „Analyse, Erkenntnisgewinn und Wert“**
  Stellen Sie in 5 Minuten die wichtigsten Erkenntnisse aus den Daten vor. Bewerten Sie den Erkenntnisgewinn, z.B. gegenüber Ihren Kommilitonen oder der Literatur! Welche Erkenntnisse hätten einen kommerziellen Wert?

Die Themen

- Was sind Datenbanken?
  - Motivation, Historie, Datenunabhängigkeit, Einsatzgebiete
- Datenbankentwurf im ER-Modell & Relationaler Datenbankentwurf
  - Entities, Relationships, Kardinalitäten, Diagramme
  - Relationales Modell, ER -> Relational, Normalformen, Transformationseigenschaften
- Relationale Algebra & SQL
  - Kriterien für Anfragesprachen, Operatoren, Transformationen
  - SQL DDL, SQL DML, SELECT … FROM … WHERE …
- Datenintegration & Transaktionsverwaltung
  - JDBC, Cursor, ETL
  - Mehrbenutzerbetrieb, Serialisierbarkeit, Sperrprotokolle, Fehlerbehandlung, Isolationsebenen in SQL
- Ausblick
  - Map/Reduce, HDFS, Hive …
  - Wert von Daten

ÜBUNGEN & EXKURS

EXKURS: NUTZER DEFINIERT ATOMARITÄT IN ANWENDUNG

Transaktionsparadigma Atomarität

- Benutzer versteht Transaktion als atomare Einheit
  - Benutzer kann Transaktionsgrenzen (in Anwendung) festlegen
  - Alle Aktionen während Transaktion sind temporär:
    → Logging durch DBMS

- Mögliche Transaktionsresultate:
  - Erfolgreiches Ende eine
  - Abbruch der Transaktio

- Im Falle des Abbruchs:
  - DBMS schreibt generell
  - Auswertung Log-Datei z
    Zustand vor Transaktio

Beispiel: (Auszug aus Prozedur, kein Autocommit!)

```
update aufpos set termin = to_date(´01.03.´,“DD.MM.‘)
where termin = to_date(´29.02.´,‘DD.MM.‘);
select aufnr,posnr,termin from aufpos;
…
if to_date(sysdate,‘YYYY‘) in (‘2008‘,‘2012‘,…) then
rollback;
else
commit;
end if;
select aufnr,posnr,termin from aufpos;
```

Transaktionsparadigma Atomarität – Transaktionsgrenzen:

Transaktionsumfang:

- Einzelnes SQL-Statement
- Mehrere SQL-Statements

Transaktionsbeginn:

- implizit oder
- ***start transaction*** (erst ab SQL`99)

Transaktionsende:

- **Implizit**:
  - Sitzungsende (Commit)
  - Fehlerfall (Rollback)
  - Ausführung DDL-Anweisung (Commit)

Transaktionsparadigma Atomarität – Transaktionsgrenzen:

Transaktionsende:

- **Explizit**:
  - **Commit** (alle Aktionen der Transaktion sind dauerhaft und auch außerhalb der Transaktion sichtbar)

commit ***[work]*** *[**comment** <Kommentar>]*
*[**force** <Transaktions-ID> [<Änderungsnummer>]];*

  - **Rollback** (Rücksetzen vor Transaktionsbeginn)

rollback *[**work**] [**to** [**savepoint**] <Sicherungspunkt>]*
*[**force** <Transaktions-ID>];*

Transaktionsparadigma Atomarität – Transaktionsgrenzen:

- Autocommit: automatisches Commit bei überschaubaren Aktionen, vor allem im Mehrnutzerbetrieb zur Erhöhung des Durchsatzes
  - Transaktionsende implizit nach jeder DML-Operation

  set autocommit [on | off] (Achtung: SQLDeveloper z.T. off)

- Transaktionsgrenzen bei Schemadeklaration:
  - create schema – Anweisung

create schema authorization *student*
create table *vert (…)*
create table *artst (…)*
create table *aufkopf (…)*
create view *auftragssicht*
as select *…;*

Transaktionsparadigma Atomarität – Savepoints:

Savepoint = Sicherungspunkt, sinnvoll bei Transaktionen mit vielen Aktionen

- kein Commit !!!
- Sichert Änderungen nur vorläufig
- Rollback kann Ergebnis eines

Savepoints wieder rückgängig machen

Was bewirkt ein rollback hier?

Was bewirkt ein rollback hier?

Beispiel:

```
update kdst set …;
savepoint punkt1;
insert into kdst …;
savepoint punkt2;
…
if … then begin
rollback to savepoint punkt1;
dbms_output.put_line
(‘Nach dem Savepoint: …‘);
end;
else
commit;
end if;
commit;   ← wozu?
```

Übungsaufgabe

- Bestimmen Sie, welche Isolationsebene Sie mindestens für folgende Transaktionen benötigen um sie konsistent auszuführen.
- Die Transaktion muss parallel mit anderen Transaktionen desselben Typs ausgeführt werden können.

```
BEGIN WORK;
SET TRANSACTION ISOLATION LEVEL ???

UPDATE BESTELLPOSTEN
  SET Rabatt = Rabatt * 0.9
  WHERE Bestell_Nr = X;

UPDATE BESTELLPOSTEN
  SET Preis = Preis + (Preis * Rabatt * 1.111);
  WHERE Bestell_Nr = X;

COMMIT WORK;
```

REPEATABLE READ;

→ Sonst "Lost-Update Problem"
→ SERIALIZABLE nicht nötig.

Übungsaufgabe

- Welche Isolationsebene benötigen sie mindestens für folgende Transaktion um korrekte Ergebnisse
- Nehmen Sie an, das gleichzeitig Upd
  auf der "Bestellposten" Tabelle
  durchgeführt werden können.

READ COMMITTED;

→ Summe enthält nur sinnvolle und vollständige Änderungen.
→ REPEATABLE READ nicht notwendig, da keine konkurrierenden Updates durchgeführt werden, die sich aus gelesenen Daten errechnen.

```
BEGIN WORK;
SET TRANSACTION ISOLATION LEVEL ???;

X = SELECT sum(preis) FROM Bestellposten
     WHERE bestelldatum BETWEEN '1/1/1998' AND '3/31/1998';

INSERT INTO Quartale VALUES (1998, "QUARTAL-1", X);

COMMIT WORK;
```


EXKURS: JDBC

Transaktionen mit JDBC

- Auto Commit ein-/ausschalten
  - `Connection#setAutoCommit(boolean autoCommit)`
- Isolationsebene sezten:
  - `Connection#setTransactionIsolation(int level)`

    mit folgenden Konstanten möglich:
    - `Connection.TRANSACTION_READ_COMMITTED`
    - `Connection.TRANSACTION_READ_UNCOMMITED`
    - `Connection.TRANSACTION_REPEATABLE_READ`
    - `Connection.TRANSACTION_SERIALIZABLE`
- Wie bei DB2 kein explizites Starten einer Transaktion
  - `Connection#commit()`
  - `Connection#rollback()`

Transaktionen mit JDBC

- Beispiel:

```
Connection con = ...;
con.setAutoCommit (false);
try {
    Statement statement =
con.createStatement(...);
    statement.execute ("SELECT ... ");
    statement.executeUpdate ("INSERT INTO ...
");
    con.commit ();
} catch (SQLException e) {
    // something went wrong
    // do some Exception handling here
    ...
    // do an explicit rollback,
    // JavaDoc says it‘s nice to do so
    con.rollback();
}
```