# Datenbanksysteme

## VU 184.686, WS 2022

## Transaktionsverwaltung

Johannes Fichte, Felix Winter

Institut für Logic and Computation
Technische Universität Wien

# Acknowledgments

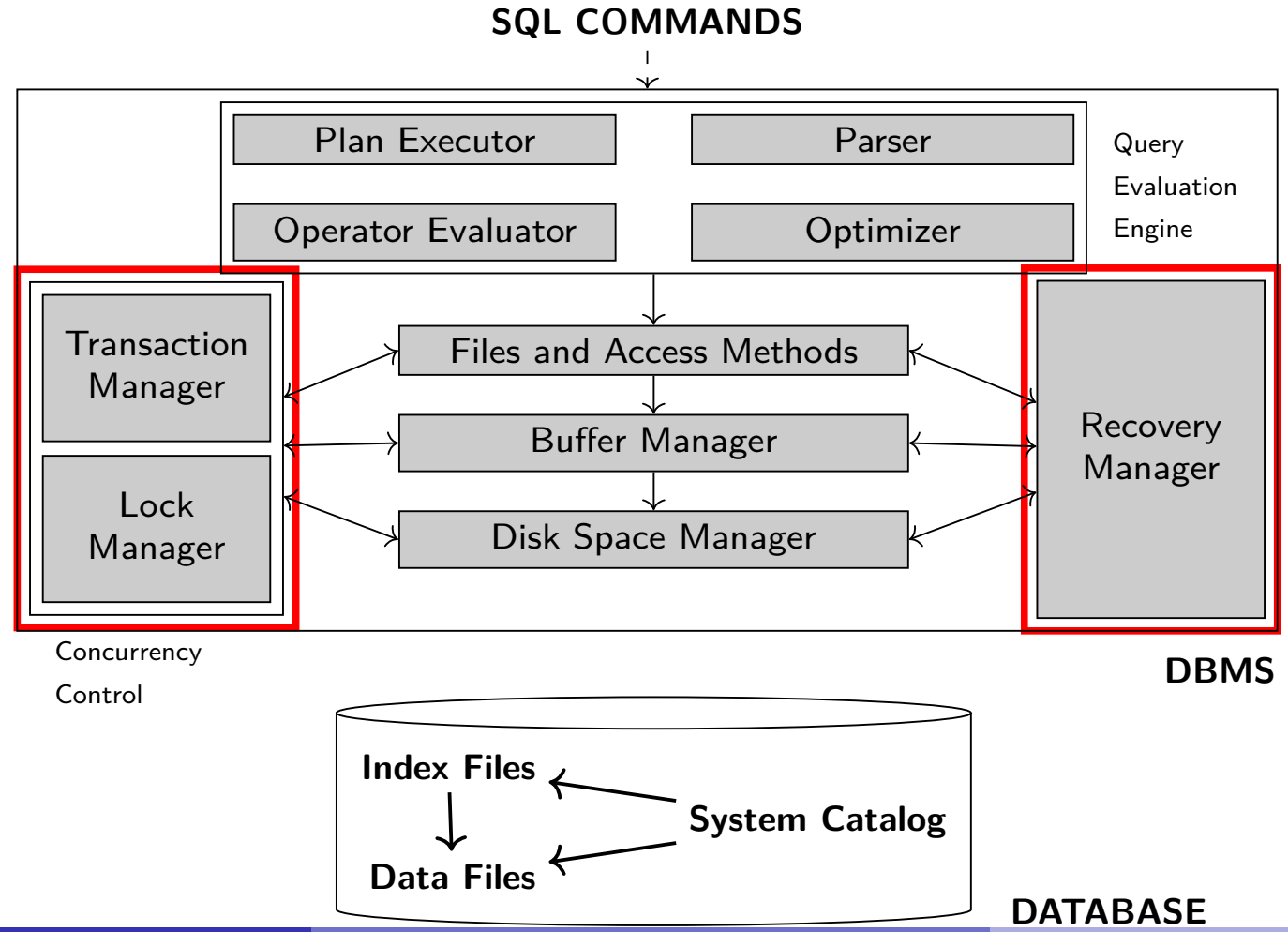
Die Folien sind eine Weiterentwicklung der Folien von Reinhard Pichler und Sebastian Skritek.

Der Inhalt basiert auf und behandelt Kapitel 9 des Lehrbuchs
(Kemper, Eickler: Datenbanksysteme – Eine Einführung).
Die meisten Beispiele und Abbildungen sind von dort entnommen.

# Transaktionsverwaltung

# Inhalt

# Architektur eines DBMS



# Concurrency Control

**Transaction Manager:**

- Steuert die Abarbeitung der Transaktionen

**Lock Manager:**

- Verwaltet die Sperranforderungen auf Datenbankobjekte (Tupel, Seite, ...)
- Erfüllt Sperranforderungen für Datenbankobjekte, sobald diese verfügbar sind

# Recovery Manager

Im laufenden Betrieb:

- Verwaltung der Log Datei (bzw. des Logs)

Beim Wiederanlauf nach einem Systemausfall/fehler:

- Wiederherstellung eines konsistenten Zustandes, d.h.
  - Redo aller verlorenen Operationen von erfolgreich abgeschlossenen Transaktionen
  - Undo aller Operationen von nicht erfolgreich abgeschlossenen Transaktionen

# Inhalt

# Inhalt

# Was ist eine Transaktion?

## Beispiel

Typische Transaktion in einer Bankanwendung:

Konto $A$, Konto $B$

1. Lese den Kontostand von $A$ in die Variable $a$: **read**$(A,a)$;
2. Reduziere den Kontostand um 50 €: $a := a - 50$;
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 €: $b := b + 50$;
6. Schreibe den neuen Kontostand in die Datenbasis: **write**$(B,b)$;

# Transaktion

## Definition (Transaktion)

Eine Transaktion ist eine Folge von Datenbankoperationen welche die Datenbasis von einem konsistenten Zustand in einen anderen konsistenten Zustand überführen. Eine Transaktion wird atomar
(= (logisch) ununterbrechbar) ausgeführt, d.h.

- als Einheit fehlerfrei ausgeführt, und
- ohne Beeinflussung durch andere Transaktionen.

# Operationen einer Transaktion

zur **Datenverarbeitung:**

read($A$,$a$) liest den Wert von Feld $A$ in eine *lokale Variable* $a$.

write($A$,$a$) schreibt den Wert $a$ in das Feld $A$.

zur **Transaktionskontrolle:**

BOT (begin of transaction)
kennzeichnet den Beginn einer Transaktion.

commit leitet die erfolgreiche Beendigung einer Transaktion ein.

abort leitet den Abbruch einer Transaktion ein. Veranlasst das DBMS dazu die Datenbasis wieder in den Zustand vor Beginn der Transaktionsausführung zurückzusetzen.

# Operationen einer Transaktion

**Zusätzliche** Operationen zur Transaktionskontrolle:

define savepoint definiert einen Sicherungspunkt, auf den die (noch aktive) Transaktion zurückgesetzt werden kann.
Vollständiger Abbruch mittels **abort** weiterhin möglich.

rollback to savepoint leitet das Zurücksetzen der aktiven Transaktion auf einen Sicherungspunkt ein.
Abhängig vom DBMS nur Rücksetzen auf letzten Sicherungspunkt oder auch auf ältere Sicherungspunkte möglich.

# Historie

**Historie** beschreibt die Reihenfolge der elementaren Operationen bei einer verzahnten Ausführung mehrerer Transaktionen.

## Beispiel

| | $T_1$ | $T_2$ | $T_3$ |
|---|---|---|---|
| 1 | BOT | | |
| 2 | | BOT | |
| 3 | | | BOT |
| 4 | | read($B$,$b_1$) | |
| 5 | write($A$, $a_1$) | | |
| 6 | | | read($C$, $c_1$) |
| 7 | commit | | |
| 8 | | | abort |

# Abschluss einer Transaktion

1. Erfolgreicher Abschluss durch ein commit
2. Wirkungsloser Abschluss (benötigt anschließendes zurücksetzen)
   - durch Benutzer initiiert mittels abort (auch rollback)
   - durch DBMS initiiert aufgrund eines Fehlers

| **BOT** | **BOT** | **BOT** |
|---|---|---|
| $op_1$ | $op_1$ | $op_1$ |
| $op_2$ | $op_2$ | $op_2$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $op_n$ | $op_m$ | $op_k$ |
| **commit** | **abort** | Fehler |

# Inhalt

# Eigenschaften von Transaktionen – ACID

**A**tomicity Eine Transaktion ist die kleinste, nicht weiter zerlegbare Einheit ("alles oder nichts").

**C**onsistency Eine Transaktion hinterlässt nach Beendigung einen konsistenten Zustand der Datenbasis.

**I**solation Nebenläufig ausgeführte Transaktionen dürfen sich nicht beeinflussen.

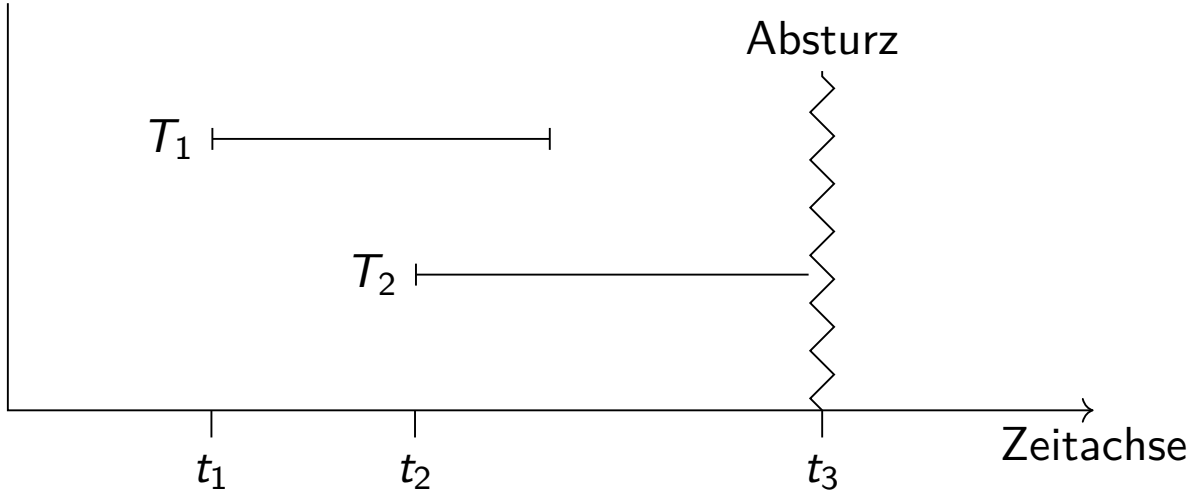
**D**urability Die Auswirkungen einer erfolgreich abgeschlossenen Transaktion gehen nicht mehr verloren
(auch nicht bei Systemfehlern).

# Komponenten der Transaktionsverwaltung

**Mehrbenutzersynchronisation:** Stellt die Isolation sicher; kontrolliert die Nebenläufigkeit.
Nebenläufigkeit ist notwendig für Performance.

**Recovery:** Stellt Atomicity und Durability sicher.
Garantiert „alles oder nichts“ und sorgt dafür, dass Änderungen einer erfolgreich beendeten Transaktion auch bei Systemfehlern nicht verloren gehen.

# Atomicity and Durability




- Transaktion $T_1$ muss nach dem Wiederanlauf vorhanden sein.
- Transaktion $T_2$: Sämtliche Datenbankänderungen durch $T_2$ müssen nach dem Wiederanlauf aus der DB entfernt sein.

# Lernziele

- Was ist eine Transaktion?
- Sind die wichtig?
- Welche Operationen innerhalb eine Transkation betrachten wir?
- Was sind die möglichen Enden einer Transaktion?
- Was besagen die ACID-Eigenschaften und was sind ihre Auswirkungen auf die Transaktionsverwaltung?

# Inhalt

# Transaktionsverwaltung in SQL

*Anmerkung:* Detailunterschiede zwischen verschiedenen DBMS

**Starten einer Transaktion:**

- Implizit durch eine Anweisung
- Explizit durch SQL Kommando

```
START TRANSACTION
BEGIN [ WORK | TRANSACTION ]
```

# Transaktionsverwaltung in SQL

**Beenden einer Transaktion:**

- Implizit ("autocommit")
- Explizit

```
COMMIT [ WORK | TRANSACTION ]
```

  - Sofern keine Probleme auftreten (wenn z.B. Konsistenzverletzungen festgestellt werden) werden die Änderungen festgeschrieben.

```
ROLLBACK [ WORK | TRANSACTION ]
ABORT [ WORK | TRANSACTION ]
```

  - Änderungen werden zurückgesetzt
  - DBMS muss erfolgreiche Ausführung garantieren.

# Implizites Transaktionsende

**Implizites `commit`**

- Bei implizitem Beginn und `AUTOCOMMIT = ON`:
  nach jedem DML/DDL Kommando
- Abhängig vom DMBS immer nach
  DDL (`CREATE TABLE`, ...) und DCL (`GRANT`, ...) Kommandos

**Implizites `rollback`**

- Bei Problemen wie Systemabsturz, Verbindungsabbrüchen,
  Konsistenzverletzungen bei `commit`, ...

**Guter Stil:** Nach Möglichkeit Transaktionen explizit beenden.

# Transaktionsverwaltung in SQL – Savepoints

- `SAVEPOINT` *sp_name*
  - Definiert Rücksetzpunkt innerhalb der laufenden Transaktion.
  - Erlaubt Rücksetzen der Transaktion zu diesem Punkt.
  - "Sparsamer" Einsatz empfohlen:
    Kann große Transaktion evtl. in kleinere zerlegt werden?
- `ROLLBACK [WORK | TRANSACTION]`
  `TO [SAVEPOINT]` *sp_name*
  - Macht Änderungen der Transaktion seit *sp_name* Rückgängig.
- `RELEASE [SAVEPOINT]` *sp_name*
  - Löscht Sicherungspunkt (sonst keine Auswirkung)