



Skript zur Vorlesung
Datenbanksysteme II
Sommersemester 2005

Kapitel 1: Einführung

Vorlesung: Christian Böhm
Übungen: Elke Achtert, Peter Kunath

Skript © 2005 Christian Böhm

<http://www.dbs.informatik.uni-muenchen.de/Lehre/DBSII>



Inhalt der Vorlesung

0. Einführung
1. Synchronisation
2. Recovery
3. Anfragebearbeitung
4. Multimedia-Datenbanken



Literaturliste, Teil 1

Datenbanksysteme II
Kapitel 1: Einführung

3

Die Vorlesung orientiert sich nicht an einem bestimmten Lehrbuch. Empfehlenswert sind aber u.a...

- A. Kemper, A. Eickler:
Datenbanksysteme
Oldenbourg, 4. Auflage (2001). 39,80 €
- R. Elmasri, S. B. Navathe:
Grundlage von Datenbanksystemen
Pearson Studium, 3. Auflage (2004). 59,95 €
- G. Saake, A. Heuer, K.-U. Sattler:
Datenbanken: Implementierungstechniken
mitp, 2. Auflage (2005). 40,95 €



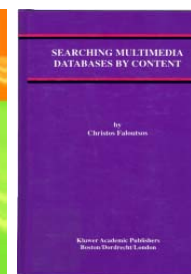
Literaturliste, Teil 2

Datenbanksysteme II
Kapitel 1: Einführung

4

Die Vorlesung orientiert sich nicht an einem bestimmten Lehrbuch. Empfehlenswert sind aber u.a...

- T. Härder, E. Rahm:
Datenbanksysteme - Konzepte und Techniken der Implementierung
Springer, 2. Auflage (2001). 39,95 €
- C. Faloutsos:
Searching Multimedia Databases by Content
Kluwer, 1. Auflage (1996). 136,50 €
- R. Ramakrishnan, J. Gehrke:
Database Management Systems
McGraw Hill, 2. Auflage (2000). 122,16 €





Inhalt

Datenbanksysteme II
Kapitel 1: Einführung

5

1. Grundbegriffe
2. Architekturen von DBMS
3. Transaktionen



Inhalt

Datenbanksysteme II
Kapitel 1: Einführung

6

1. Grundbegriffe
2. Architekturen von DBMS
3. Transaktionen



Komponenten eines DBS

Ein Datenbanksystems (DBS) besteht aus...

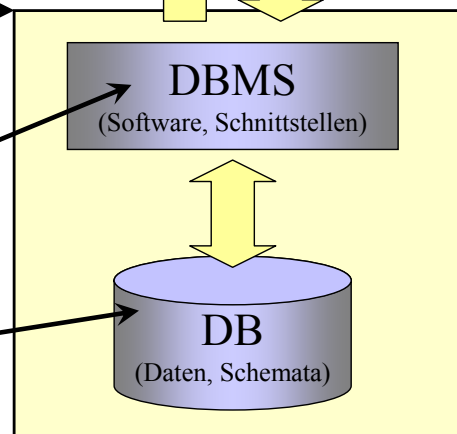
DB-Anwendungen (kommunizieren mit DBMS)

Anwendungsprogramme,
Benutzer

DBS: Datenbanksystem (DB + DBMS)

**DBMS: Datenbank-
Management-System** (Software zur Verwaltung)

DB: Datenbank (eigentliche Datensammlung)



Aufgaben eines DBS

Primäre Aufgabe eines DBS ist die ...

- Beschreibung
- Speicherung und Pflege
- und Wiedergewinnung

umfangreicher Datenmengen, die von verschiedenen Anwendungsprogrammen dauerhaft (persistent) genutzt werden



Anforderungen an ein DBS (1)

Liste von 9 Anforderungen (Edgar F. Codd, 1982)

- **Integration**
Einheitliche Verwaltung **aller** von Anwendungen benötigten Daten.
Redundanzfreie Datenhaltung des gesamten Datenbestandes
- **Operationen**
Operationen zur Speicherung, zur Recherche und zur Manipulation der Daten müssen vorhanden sein
- **Data Dictionary**
Ein Katalog erlaubt Zugriffe auf die Beschreibung der Daten
- **Benutzersichten**
Für unterschiedliche Anwendungen unterschiedliche Sicht auf den Bestand
- **Konsistenzüberwachung**
Das DBMS überwacht die Korrektheit der Daten bei Änderungen



Anforderungen an ein DBS (2)

- **Zugriffskontrolle**
Ausschluss unauthorisierter Zugriffe
- **Transaktionen**
Zusammenfassung einer Folge von Änderungsoperationen zu einer Einheit, deren Effekt bei Erfolg permanent in DB gespeichert wird
- **Synchronisation**
Arbeiten mehrere Benutzer gleichzeitig mit der Datenbank dann vermeidet das DBMS unbeabsichtigte gegenseitige Beeinflussungen
- **Datensicherung**
Nach Systemfehlern (d.h. Absturz) oder Medienfehlern (defekte Festplatte) wird die Wiederherstellung ermöglicht (im Gegensatz zu Datei-Backup Rekonstruktion des Zustands der letzten erfolgreichen TA)

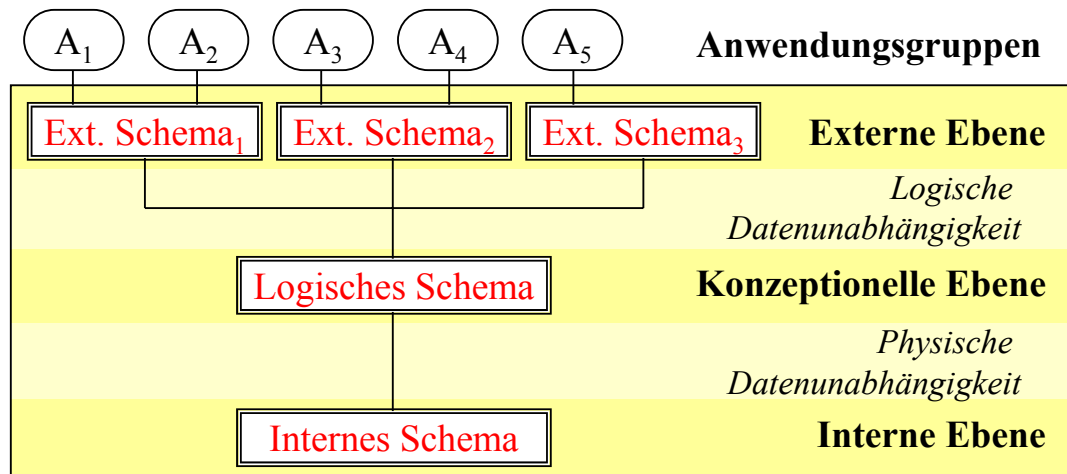


Ebenenmodell (1)

Drei-Ebenen-Architektur zur Realisierung von

- **physischer**
- **und logischer**

Datenunabhängigkeit (nach ANSI/SPARC)



Ebenenmodell (2)

Externe Ebene

- Sammlung der individuellen Sichten aller Benutzer- bzw. Anwendungsgruppen in mehreren externen Schemata
- Benutzer soll keine Daten sehen, die er nicht sehen will (Übersichtlichkeit) oder nicht sehen darf (Datenschutz)
 - Beispiel: Klinik-Pflegepersonal benötigt andere Aufbereitung der Daten als die Buchhaltung
- Datenbank wird damit von Änderungen und Erweiterungen der Anwenderschnittstellen abgekoppelt (*logische Datenunabhängigkeit*)



Ebenenmodell (3)

Konzeptionelle Ebene

- Logische Gesamtsicht *aller* Daten der DB unabhängig von den einzelnen Applikationen
- Niedergelegt in konzeptionellem (logischem) Schema
- Ergebnis des (logischen) Datenbank-Entwurfs
- Beschreibung aller Objekttypen und Beziehungen
- Keine Details der Speicherung
- Formulierung im Datenmodell des Datenbanksystems
- Spezifikation mit Hilfe einer Daten-Definitionssprache (Data Definition Language, DDL)



Ebenenmodell (4)

Interne Ebene

- Beschreibung der systemspezifischen Realisierung der DB-Objekte (physische Speicherung), z.B.
 - Aufbau der gespeicherten Datensätze
 - Indexstrukturen wie z.B. Suchbäume
- Bestimmt maßgeblich das Leistungsverhalten des gesamten DBS
- Die Anwendungen sind von Änderungen des internen Schemas nicht betroffen (*physische Datenunabhängigkeit*)



Inhalt

Datenbanksysteme II
Kapitel 1: Einführung

15

1. Grundbegriffe

2. Architekturen von DBMS

3. Transaktionen



Schichtenmodell (1)

Datenbanksysteme II
Kapitel 1: Einführung

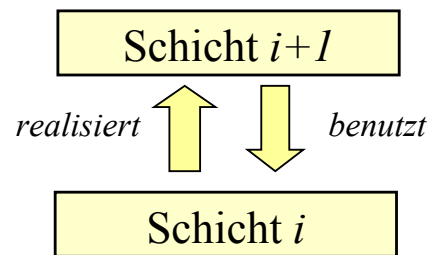
16

- Schichtenmodell aus dem SW-Engineering
 - Komponenten eines komplexen Systems sind hierarchisch strukturiert
 - Keine Schicht ruft Operationen aus einer höheren Schicht auf.
- Jede Schicht definiert eine abstrakte Maschine
 - Darüber liegende Schichten setzen auf dem jeweiligen Abstraktionsgrad auf
 - Darunter liegende Schichten stellen den jeweiligen Abstraktionsgrad zur Verfügung



Schichtenmodell (2)

- Schnittstelle zwischen Schicht i und Schicht $i+1$ besteht aus Operationen O :
 - Schicht i realisiert die Operationen O der Schnittstelle
 - Schicht $i+1$ benutzt die Operationen O der Schnittstelle



Schichtenmodell (3)

Vorteile einer Schichtenarchitektur

- Einfache Implementierung von Komponenten aus höheren Schichten: Sie können auf dem Abstraktionsgrad tiefer liegender Schichten aufbauen.
- Änderungen in höheren Ebenen wirken sich nicht auf tiefere Ebenen aus.
- Beim Entfernen höherer Ebenen bleiben tiefere Ebenen dennoch funktionsfähig.
- Tiefere Ebenen können getestet werden, bevor höhere Ebenen lauffähig sind.
- Verändert man auf einer tieferen Ebene die Implementierung, aber nicht die Schnittstelle (weder syntaktisch noch semantisch), so muss auch in höheren Schichten nichts geändert werden.



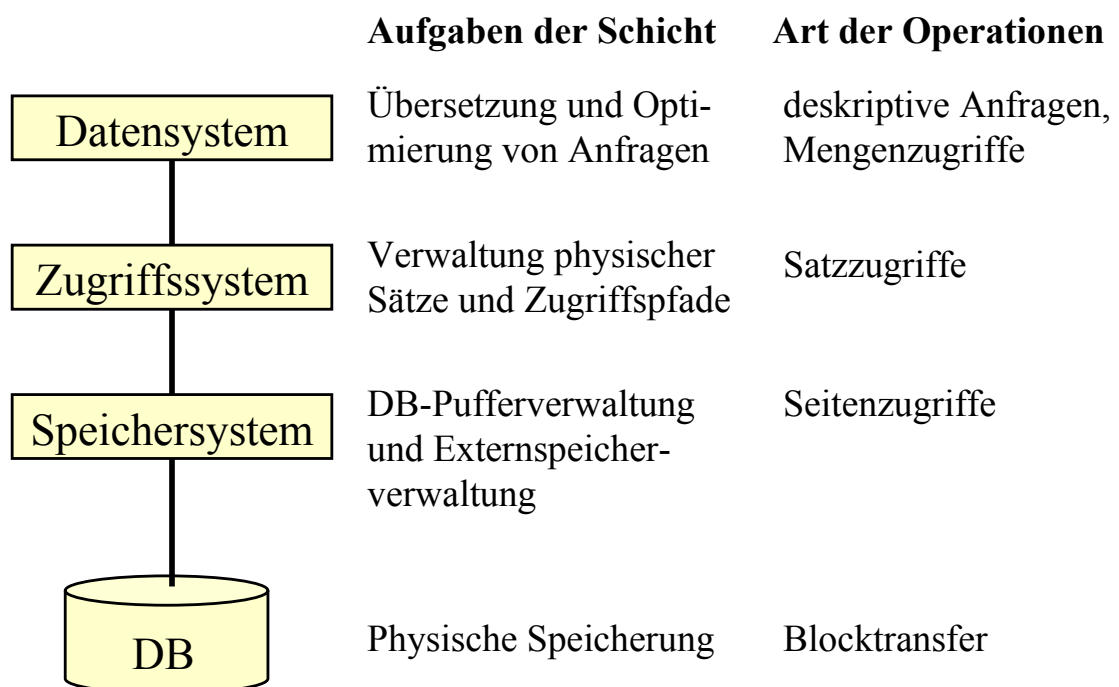
Schichtenmodell (4)

Optimale Anzahl n von Schichten

- zu wenige Schichten (z.B. $n=1$):
Nachteil: komplexe monolithische Komponenten, schwer wartbar
- zu viele Schichten (z.B. $n > 10$):
 - *Vorteil:* Reduktion der Komplexität einzelner Schichten; System gut erweiterbar
 - *Nachteil:* Hohe Anzahl zu durchlaufender Schnittstellen kann zu Leistungseinbußen führen; Fehlerbehandlung kann aufwändig sein

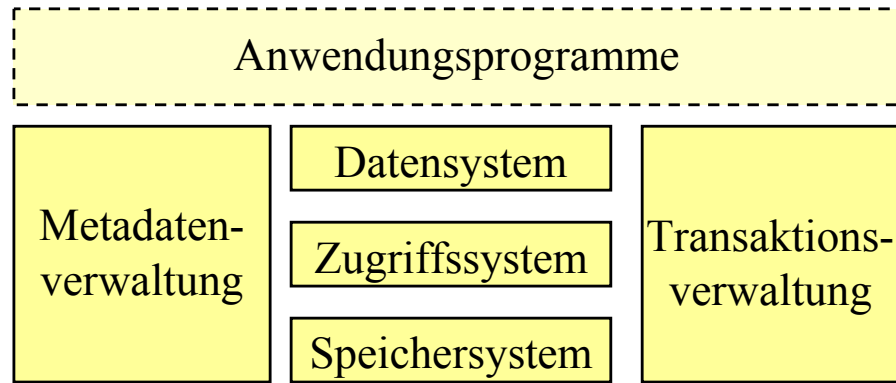


Schichten des DBMS-Kerns





Gesamtarchitektur eines DBMS

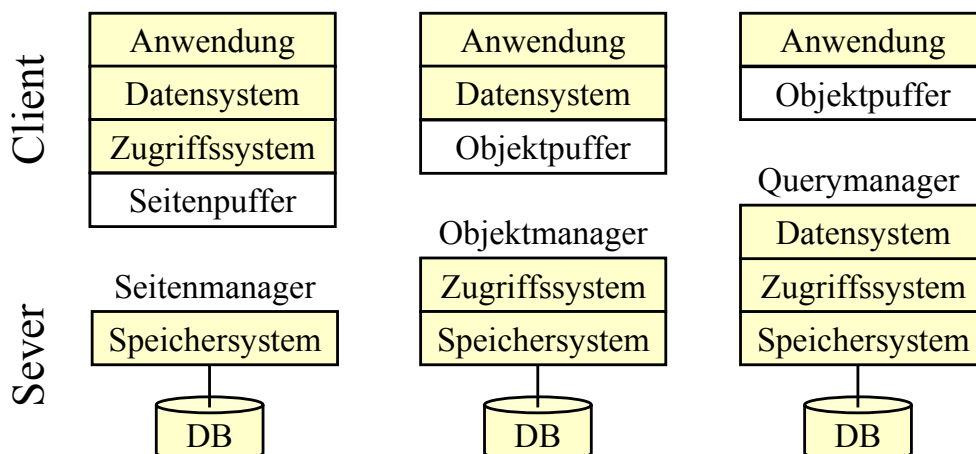


- **Daten-, Zugriffs- und Speichersystem** (wie oben) für die Grundfunktionalität
- **Metadatenverwaltung** für modellspezifische Daten (Schema, Indexe, Data Dictionary)
- **Transaktionsverwaltung** für Synchronisation und Datensicherheit



Client/Server-Architekturen

- Die hierarchische Schichtung der Systemkomponenten bestimmt die **Aufrufstruktur**, nicht aber die **Prozessstruktur** (Zuordnung zu physischen Recheneinheiten)
- Folgende Client/Server-Modelle sind gebräuchlich:

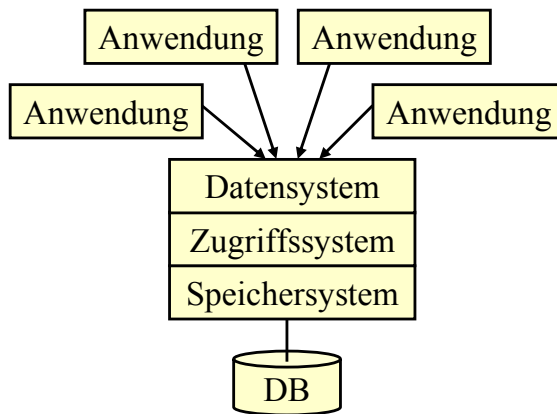




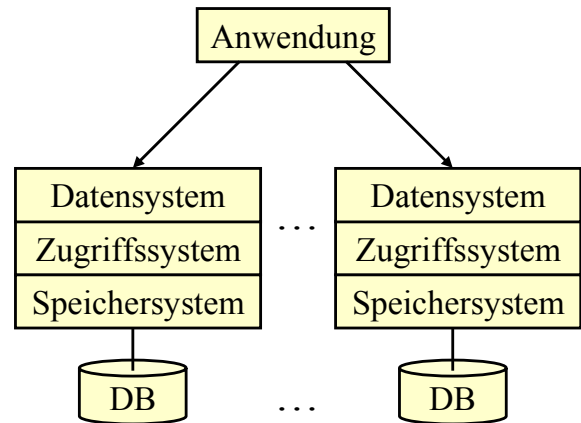
Mehrbenutzer- und verteilte DBS

Verteilte Mehrbenutzer-DBS ($m:n$) vereinigen die folgenden beiden Prinzipien:

Mehrbenutzerbetrieb ($m:1$)



Verteilte DBS ($1:n$)



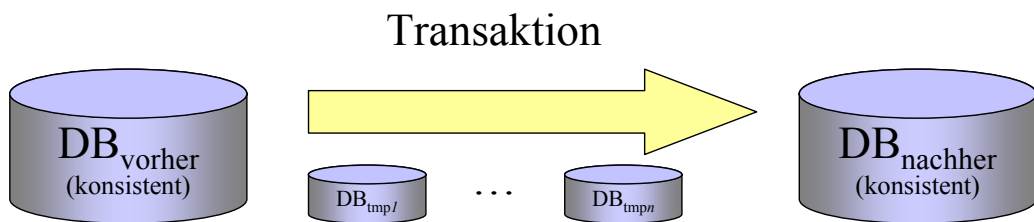
Inhalt

1. Grundbegriffe
2. Architekturen von DBMS
3. Transaktionen



Transaktionskonzept (1)

- Transaktion: Folge von Aktionen (*read*, *write*), die die DB von einem **konsistenten** Zustand in einen anderen **konsistenten** Zustand überführt
- Transaktionen: Einheiten **integritätserhaltender Zustandsänderungen** einer Datenbank
- Hauptaufgaben der Transaktions-Verwaltung
 - Synchronisation (Koordination mehrerer Benutzerprozesse)
 - Recovery (Behebung von Fehlersituationen)



Transaktionskonzept (2)

Beispiel Bankwesen:

Überweisung von Huber an Meier in Höhe von 200 €

- Mgl. Bearbeitungsplan:
 - (1) Erniedrige Stand von Huber um 200 €
 - (2) Erhöhe Stand von Meier um 200 €
- Möglicher Ablauf

Konto	Kunde	Stand	(1)	Konto	Kunde	Stand	(2)
	Meier	1.000 €	→		Meier	1.000 €	↓
	Huber	1.500 €			Huber	1.300 €	

System-absturz



Eigenschaften von Transaktionen

- **ACID-Prinzip**
 - **Atomicity** (Atomarität)
Der Effekt einer Transaktion kommt entweder ganz oder gar nicht zum Tragen.
 - **Consistency** (Konsistenz, Integritätserhaltung)
Durch eine Transaktion wird ein konsistenter Datenbankzustand wieder in einen konsistenten Datenbankzustand überführt.
 - **Isolation** (Isoliertheit, logischer Einbenutzerbetrieb)
Innerhalb einer Transaktion nimmt ein Benutzer Änderungen durch andere Benutzer nicht wahr.
 - **Durability** (Dauerhaftigkeit, Persistenz)
Der Effekt einer abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank erhalten.



Steuerung von Transaktionen (1)

- **begin of transaction (BOT)**
 - markiert den Anfang einer Transaktion
 - Transaktionen werden implizit begonnen, es gibt kein `begin work` o.ä.
- **end of transaction (EOT)**
 - markiert das Ende einer Transaktion
 - alle Änderungen seit dem letzten BOT werden festgeschrieben
 - SQL: `commit` oder `commit work`
- **abort**
 - markiert den Abbruch einer Transaktion
 - die Datenbasis wird in den Zustand vor BOT zurückgeführt
 - SQL: `rollback` oder `rollback work`
- **Beispiel**

```
UPDATE Konto SET Stand = Stand-200 WHERE Kunde = 'Huber';  
UPDATE Konto SET Stand = Stand+200 WHERE Kunde = 'Meier';  
COMMIT;
```



Steuerung von Transaktionen (2)

Unterstützung langer Transaktionen durch

- **define savepoint**

- markiert einen zusätzlichen Sicherungspunkt, auf den sich die noch aktive Transaktion zurücksetzen lässt
- Änderungen dürfen noch nicht festgeschrieben werden, da die Transaktion noch scheitern bzw. zurückgesetzt werden kann
- SQL: `savepoint <identifizier>`

- **backup transaction**

- setzt die Datenbasis auf einen definierten Sicherungspunkt zurück
- SQL: `rollback to <identifizier>`



Ende von Transaktionen

- **COMMIT gelingt**

→ der neue Zustand wird dauerhaft gespeichert.

- **COMMIT scheitert**

→ der ursprüngliche Zustand wie zu Beginn der Transaktion bleibt erhalten (bzw. wird wiederhergestellt). Ein COMMIT kann z.B. scheitern, wenn die Verletzung von Integritätsbedingungen erkannt wird.

- **ROLLBACK**

→ Benutzer widerruft Änderungen

