



LUDWIG-  
MAXIMILIANS-  
UNIVERSITY  
MUNICH



DEPARTMENT  
INSTITUTE FOR  
INFORMATICS



DATABASE  
SYSTEMS  
GROUP

Skript zur Vorlesung:

# Einführung in die Informatik: Systeme und Anwendungen

Sommersemester 2011

## Kapitel 3: Datenbanksysteme

Vorlesung: Prof. Dr. Christian Böhm

Übungen: Annahita Oswald

Skript © 2004 Christian Böhm, Peer Kröger

<http://www.dbs.ifi.lmu.de/Lehre/InfoNF>

# Überblick

3.1 Einleitung

3.2 Das Relationale Modell

3.3 Die Relationale Algebra

3.4 Mehr zu SQL

3.5 Das E/R-Modell

[ 3.6 Normalformen ]

## 3.1 Einleitung

- Bisher:
  - Eigentlich nur Betrachtung des Hauptspeichers
  - Objekte zur Datenverarbeitung werden im Hauptspeicher erzeugt und nach dem Programmablauf wieder entfernt
  - Swapping / virtueller Speicher: Objekte werden möglicherweise ausgelagert, aber „eigentliche Aktionen“ (Verarbeitung) nur im Hauptspeicher
- Warum ist dies nicht ausreichend?
  - Viele Anwendungen müssen Daten **permanent** speichern
  - Arbeitsspeicher ist häufig nicht groß genug, um z.B. alle Kundendaten einer Bank oder Patientendaten einer Klinik zu speichern

## 3.1 Einleitung

- Daten können auf dem sog. **Externspeicher** (auch **Festplatte** genannt) permanent gespeichert werden

### Arbeitsspeicher:

- rein elektronisch (Transistoren und Kondensatoren)
- flüchtig
- schnell: 10 ns/Zugriff \*
- wahlfreier Zugriff
- teuer:  
300-400 € für 1 GByte\*

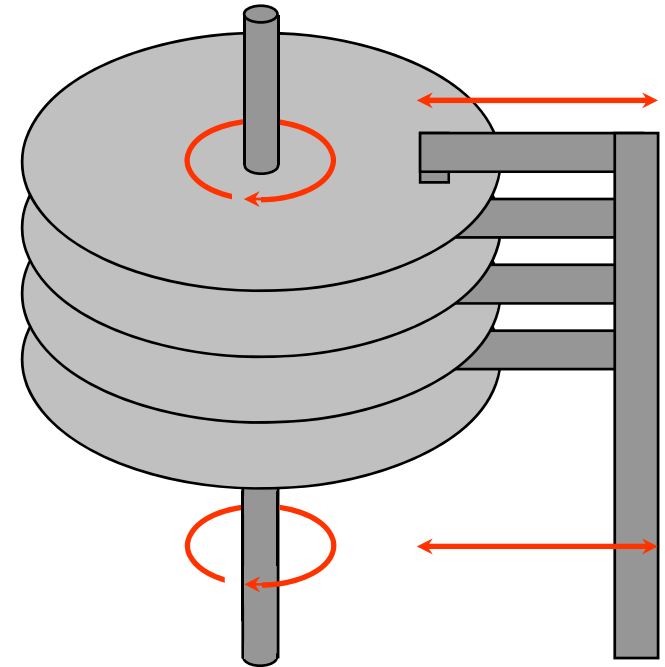
### Externspeicher:

- Speicherung auf magnetisierbaren Platten (rotierend)
- nicht flüchtig
- langsam: 5 ms/Zugriff \*
- blockweiser Zugriff
- wesentlich billiger:  
200-400 € für 100 GByte\*

\*April 2002 .

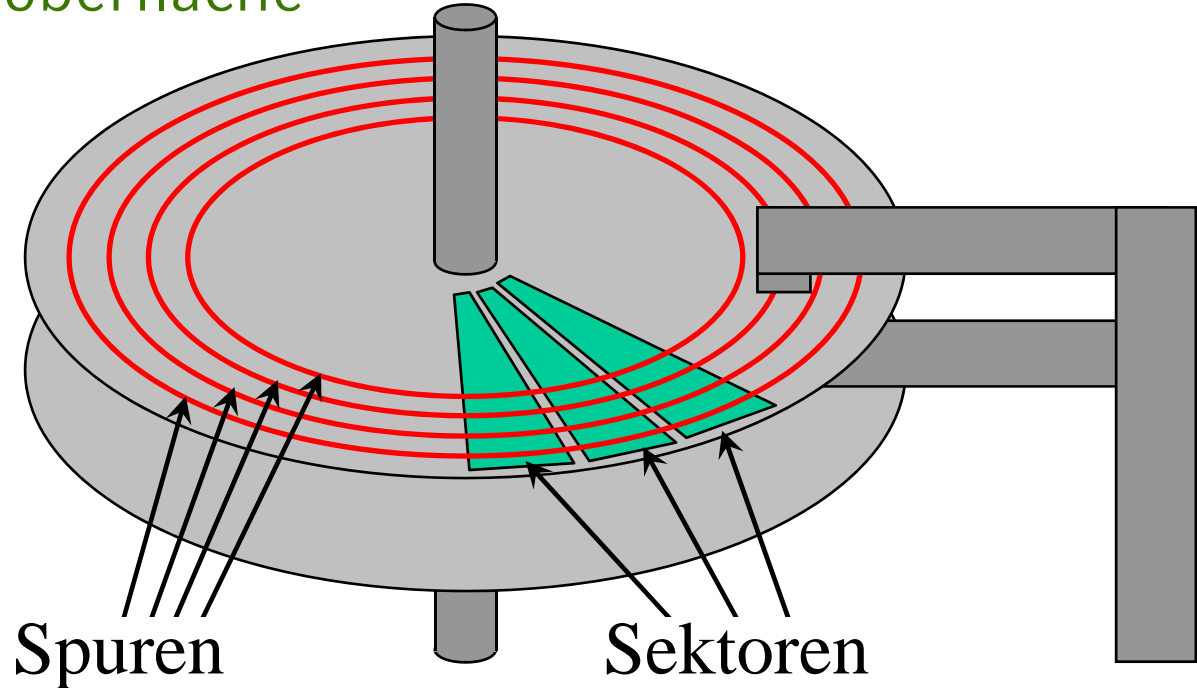
# 3.1 Einleitung

- Aufbau einer Festplatte
  - Mehrere magnetisierbare **Platten** rotieren z.B. mit 7.200 Umdrehungen\* pro Minute um eine gemeinsame Achse (\*z. Z. 5400, 7200, 10000 upm)
  - Ein Kamm mit je zwei **Schreib-/Leseköpfen** pro Platte (unten/oben) bewegt sich in radialer Richtung.



# 3.1 Einleitung

## Einteilung der Plattenoberfläche



- (interne) Adressierung einer Information:  
[Platten-Nr | Oberfl.-Nr | Spur-Nr | Sektor-Nr | Byte-Nr]
- Berechnung der Kapazität:  
 $\# \text{ Platten} * 2 * \# \text{ Spuren} * \# \text{ Sektoren} * \text{ Bytes pro Sektor}$

## 3.1 Einleitung

- Lesen/Schreiben eines Sektors
  - Positionieren des Kamms mit den Schreib-/Leseköpfen auf der Spur
  - Warten bis die Platte so weit rotiert ist, dass der Beginn des richtigen Sektors unter dem Schreib-/Lesekopf liegt
  - Übertragung der Information von der Platte in den Arbeitsspeicher (bzw. umgekehrt)

### ***Achtung:***

Es ist aus technischen Gründen nicht möglich, einzelne Bytes zu lesen bzw. zu schreiben, sondern mindestens einen ganzen Sektor

# 3.1 Einleitung

- Speicherung in Dateien
  - Adressierung mit Platten-Nr., Oberfl.-Nr. usw. für den Benutzer nicht sichtbar
  - Arbeit mit Dateien:
    - Dateinamen
    - Verzeichnishierarchien
    - Die Speicherzellen einer Datei sind byteweise von 0 aufsteigend durchnummeriert.
    - Die Ein-/Ausgabe in Dateien wird gepuffert, damit nicht der Programmierer verantwortlich ist, immer ganze Sektoren zu schreiben/lesen.



# 3.1 Einleitung

- Beispiel: Dateizugriff in Java
  - Werden die Objekte einer Applikation in eine Datei geschrieben, ist das Dateiformat vom Programmierer festzulegen:

Name (10 Zeichen)										Vorname (8 Z.)								Jahr (4 Z.)			
F	r	a	n	k	l	i	n			A	r	e	t	h	a			1	9	4	2

- Wo findet man dieses Datei-Schema im Quelltext z.B. des Java-Programms ?  
 Das Dateischema wird nicht explizit durch den Quelltext beschrieben, sondern implizit in den Ein-/Auslese-Prozeduren der Datei, d.h. im Algorithmus zum Ein-/Auslesen der Datei

## 3.1 Einleitung

- Konsequenz:
  - Wenn eine Änderung des Dateiformates nötig ist (z.B. zusätzliche Objektattribute in neuer Programmversion):
    - Alte Datendateien können nicht mehr verwendet werden oder müssen z.B. durch extra Programme konvertiert werden
    - Die Änderung muss in allen Programmen nachgeführt werden, die mit den Daten arbeiten, auch in solchen, die logisch von Änderung gar nicht betroffen sind

⇒ ***Logische Datenabhängigkeit***

## 3.1 Einleitung

- Meist werden die Datensätze anhand ihrer Position adressiert/separiert:
  - z.B. jeder Satz hat 22 Zeichen:
  - 1. Satz: Adresse 0; 2. Satz: Adresse 22 usw.
- Suche gemäß bestimmten Attributwerten (z.B. Namen des Kunden) muss im Programm codiert werden
- Soll die Datei z.B. mit einem Suchbaum unterstützt werden, dann gleiche Konsequenzen wie bei logischer Änderung

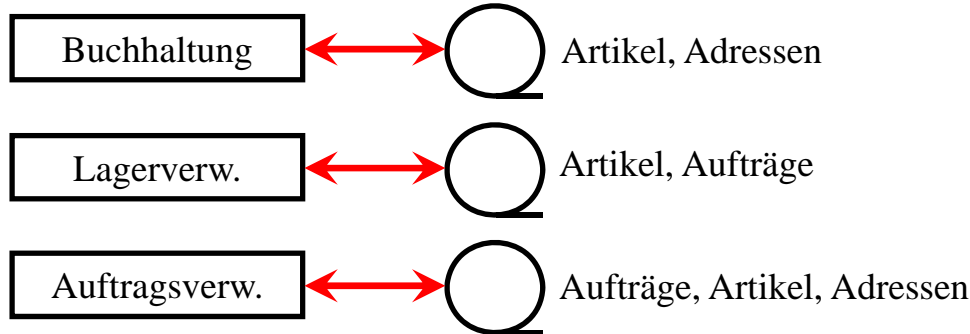
⇒ ***Physische Datenabhängigkeit***

## 3.1 Einleitung

- Informationssysteme sind große Software-Systeme mit viele einzelnen Programmen die teils mit gemeinsamen Daten, teils mit unterschiedlichen arbeiten, z.B.
  - **Buchhaltung**: Artikel- und Adressinformation
  - **Lagerverwaltung**: Artikel und Aufträge
  - **Auftragsverwaltung**.: Aufträge, Artikel, Adressen
  - **CAD-System**: Artikel, techn. Daten, Bausteine
  - **Produktion, Bestelleingang, Kalkulation**: ...

# 3.1 Einleitung

- Daten werden meist mehrfach gespeichert

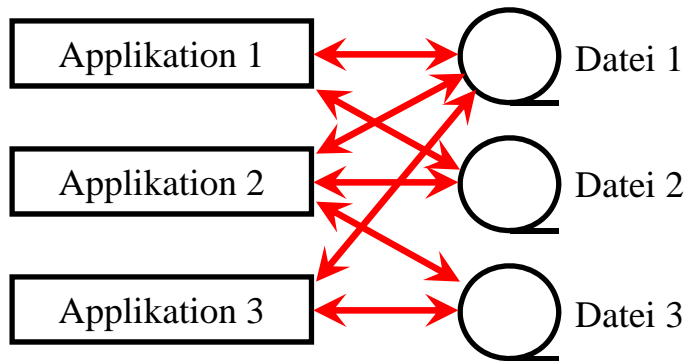


## Konsequenz:

u.a. **Änderungs-Anomalien**

Bei Änderung einer Adresse müssen viele Dateien nach den Einträgen durchsucht werden

- Alternative Implementierung



## Nachteile:

- unübersichtlich
- Schnittstellenproblematik: bei logischen oder physischen Änderungen des Dateischemas müssen viele Programme angepasst werden

## 3.1 Einleitung

- Zusammenfassung Probleme von Dateien
  - Datenabhängigkeit (logische, physische)
  - Redundanz

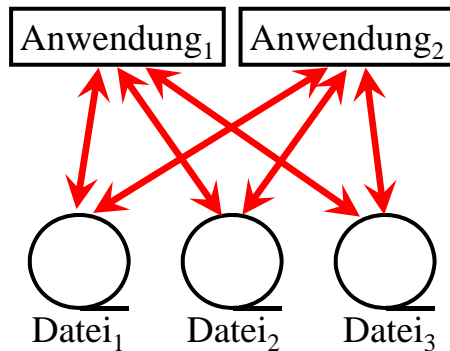
desweiteren:

- In großen Informationssystemen arbeiten viele Benutzer gleichzeitig mit den Daten
  - Dateisysteme bieten zu wenige Möglichkeiten, um diese Zugriffe zu synchronisieren
- Dateisysteme schützen nicht in ausreichendem Maß vor Datenverlust im Fall von Systemabstürzen und Defekten
- Dateisysteme bieten nur unflexible Zugriffskontrolle (Datenschutz)

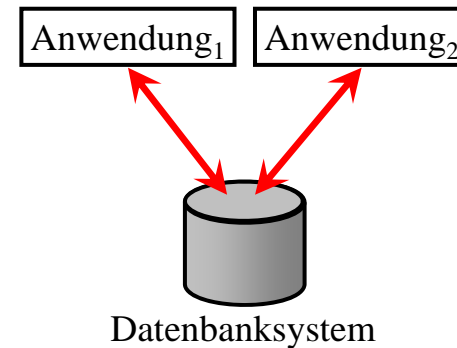
# 3.1 Einleitung

- Probleme von Dateien:
  - Datenabhängigkeit (logische, physische)
  - Redundanz
  - Keine Synchronisation für Mehrbenutzerbetrieb
  - Kein ausreichender Schutz vor Verlust, unerwünschtem Zugriff, ...
  - ...
- Um diese Probleme mit einheitlichem Konzept zu behandeln, setzt man **Datenbanken** ein:

Lösung mit Dateisystem



Lösung mit Datenbanksystem



# 3.1 Einleitung

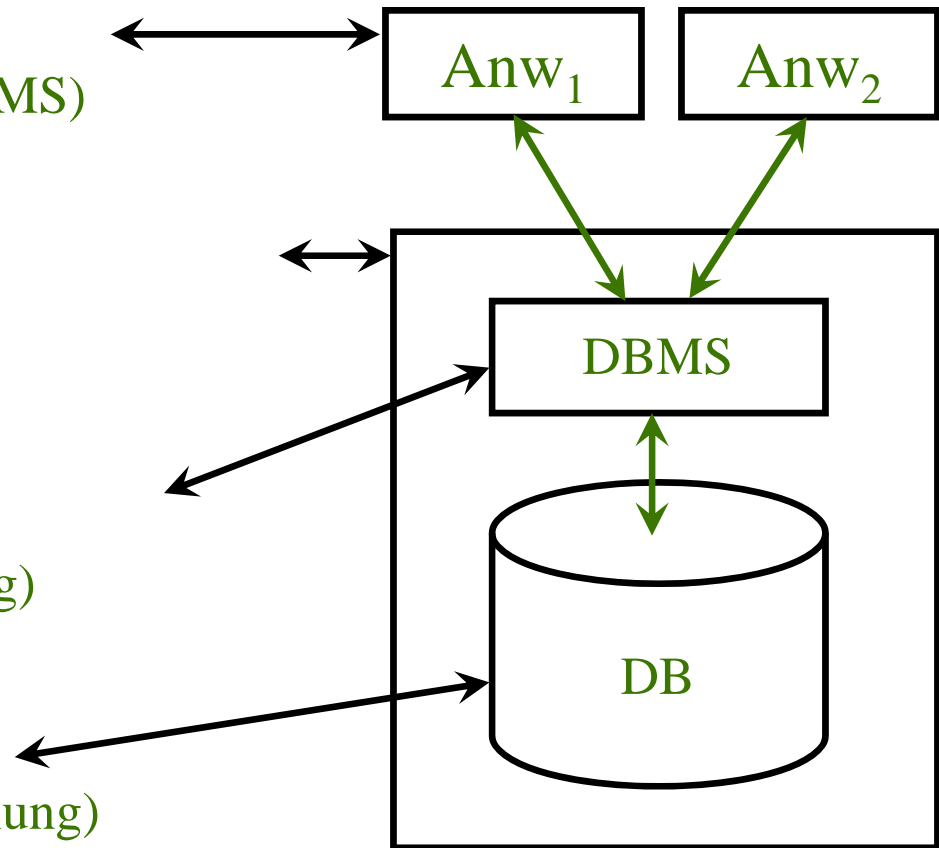
## Komponenten eines DBS

**DB-Anwendungen**  
(kommunizieren mit DBMS)

**DBS: Datenbanksystem**  
(DB + DBMS)

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

**DB: Datenbank**  
(eigentliche Datensammlung)





## 3.1 Einleitung

### Liste von 9 Anforderungen an ein DBS (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

## 3.1 Einleitung

- **Zugriffskontrolle**  
Ausschluss unauthorisierter Zugriffe
- **Transaktionen**  
Zusammenfassung einer Folge von Änderungsoperationen zu einer Einheit, deren Effekt bei Erfolg permanent in der 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 nach der letzten erfolgreichen Transaktion)

## 3.1 Einleitung

Man unterscheidet zwei Ebenen von DBS:

- Intensionale Ebene: ***Datenbankschema***
  - beschreibt ***möglichen*** Inhalt der DB
  - Struktur- und Typinformation der Daten (Metadaten)
  - Art der Beschreibung vorgegeben durch Datenmodell
  - Änderungen möglich, aber selten (Schema-Evolution)
- Extensionale Ebene: ***Ausprägung der Datenbank***
  - ***tatsächlicher*** Inhalt der DB (DB-Zustand)
  - Objektinformation, Attributwerte
  - Struktur vorgegeben durch Datenbankschema
  - Änderungen häufig (Flugbuchung: 10000 TA/min)

# 3.1 Einleitung

Einfaches Beispiel:

- Schema:

Name (10 Zeichen)	Vorname (8 Z.)	Jahr (4 Z.)																						
<table border="1"> <tr> <td> </td><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td> </tr> </table>											<table border="1"> <tr> <td> </td><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td><td> </td> </tr> </table>									<table border="1"> <tr> <td> </td><td> </td><td> </td><td> </td> </tr> </table>				

- DB-Zustand:

F	r	a	n	k	l	i	n			A	r	e	t	h	a			1	9	4	2
R	i	t	c	h	i	e				L	i	o	n	e	l			1	9	4	9

- Nicht nur DB-Zustand, sondern auch DB-Schema wird in DB gespeichert.
- Vorteil: Sicherstellung der Korrektheit der DB

# 3.1 Einleitung

- Schema in Datenbanken
  - Explizit modelliert (Textdokument oder grafisch)
  - In Datenbank abgespeichert
  - Benutzer kann Schema-Informationen auch aus der Datenbank ermitteln: **Data Dictionary, Metadaten**
  - DBMS überwacht Übereinstimmung zwischen DB-Schema und DB-Zustand
  - Änderung des Schemas wird durch DBMS unterstützt (Schema-Evolution, Migration)

# 3.1 Einleitung

- Schema in Dateien
  - Kein Zwang, das Schema explizit zu modellieren
  - Schema implizit in den Prozeduren zum Ein-/Auslesen
  - Schema gehört zur Programm-Dokumentation
  - oder es muss aus Programmcode herausgelesen werden.  
Hacker-Jargon: RTFC (read that f\*\*\*ing code)
  - Fehler in den Ein-/Auslese-Prozeduren können dazu führen, dass gesamter Datenbestand unbrauchbar wird:

F	r	a	n	k	l	i	n			A	r	e	t	h	a		1	9	4	2	R
i	t	c	h	i	e				L	i	o	n	e	l			1	9	4	9	

- Bei Schema-Änderung müssen Migrations-Prozeduren programmiert werden, um bestehende Dateien auf das neue Format umzustellen

# 3.1 Einleitung

- Datenmodelle
  - Formalismen zur Beschreibung des DB-Schemas
    - Objekte der Datenbank
    - Beziehungen zwischen verschiedenen Objekten
    - Integritätsbedingungen
  - Verschiedene Datenmodelle unterscheiden sich in der Art und Weise, wie Objekte und Beziehungen dargestellt werden
  - Die wichtigsten Datenmodelle sind:
    - Hierarchisches Datenmodell
    - Netzwerk-Datenmodell
    - Relationales Datenmodell
    - Objektorientiertes Datenmodell
    - Objekt-relationales Datenmodell

# 3.1 Einleitung

- Datenbank-Sprachen
  - Data Definition Language (DDL)
    - Deklarationen zur Beschreibung des Schemas
    - Bei relationalen Datenbanken:  
Anlegen und Löschen von Tabellen, Integritätsbedingungen usw.
  - Data Manipulation Language (DML)
    - Anweisungen zum Arbeiten mit den Daten in der Datenbank (Datenbank-Zustand)
    - lässt sich weiter unterteilen in Konstrukte
      - zum reinen Lesen der DB (Anfragesprache)
      - zum Manipulieren (Einfügen, Ändern, Löschen) des Datenbankzustands



# 3.1 Einleitung

- SQL (Structured Query Language)
  - **DIE** Anfragesprache für relationale Datenbanksysteme
  - Enthält DDL und DML Konstrukte
  - Typischerweise interaktiv (über DB-Oberfläche) als auch aus einem Programm heraus aufrufbar (z.B. aus Java mittels JDBC)
  - Entwickelt seit 1974 bei IBM, erster Standard 1986
  - Beispiel:
    - Anlegen einer Datenbank  

```
CREATE DATABASE <DBName>;
```
    - Löschen einer Datenbank  

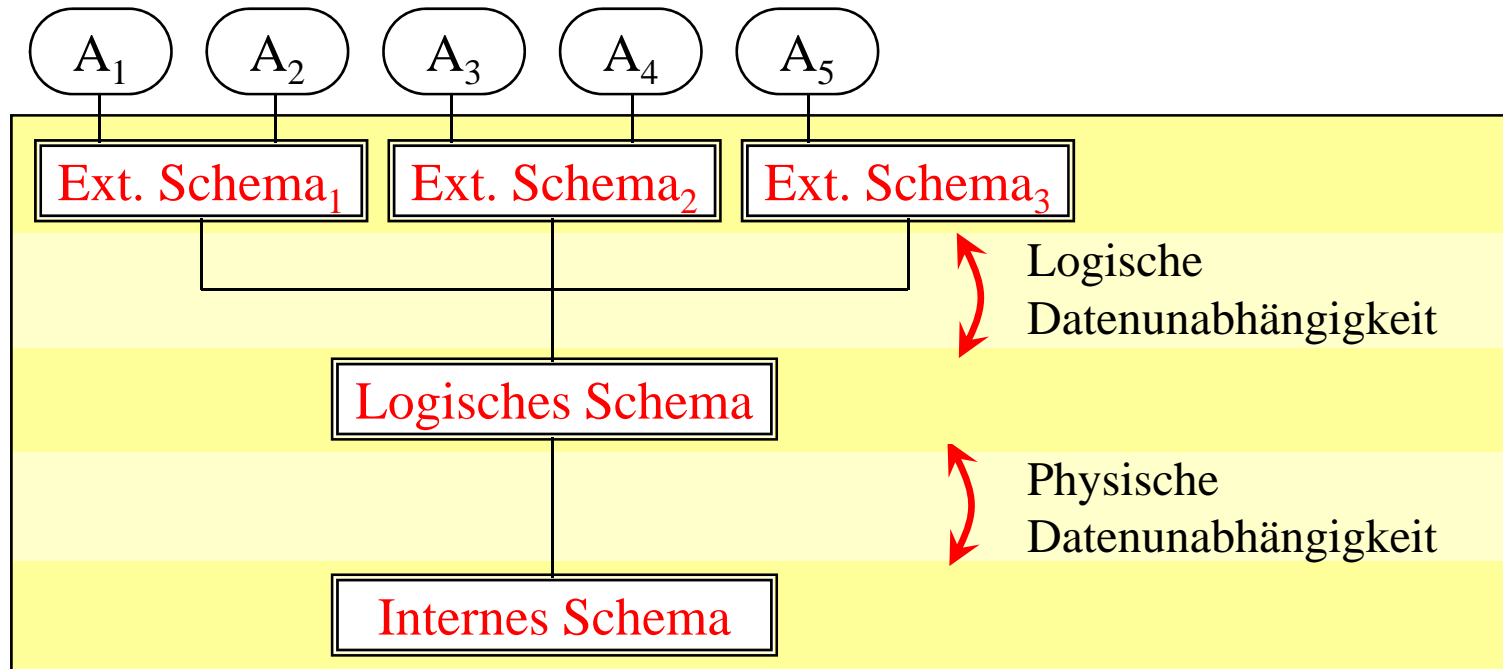
```
DROP DATABASE <DBName>;
```
  - Details später ... (Kapitel 3.4)

# 3.1 Einleitung

Drei-Ebenen-Architektur zur Realisierung von

- physischer und
- logischer

Datenunabhängigkeit (nach ANSI/SPARC)



# 3.1 Einleitung

- Konzeptionelle Ebene
  - Logische Gesamtsicht *aller* Daten der DB unabhängig von den einzelnen Applikationen
  - Niedergelegt in konzeptionellem (logischem) Schema
  - Ergebnis des (logischen) Datenbank-Entwurfs (siehe Kapitel 3.5)
  - Beschreibung aller Objekttypen und Beziehungen
  - Keine Details der Speicherung
  - Formuliert im Datenmodell des Datenbanksystems
  - Spezifiziert mit Hilfe einer Daten-Definitionssprache (Data Definition Language, DDL)

# 3.1 Einleitung

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

# 3.1 Einleitung

- Interne Ebene
  - Das interne Schema beschreibt die systemspezifische Realisierung der DB-Objekte (physische Speicherung), z.B.
    - Aufbau der gespeicherten Datensätze
    - Indexstrukturen wie z.B. Suchbäume
  - Das interne Schema bestimmt maßgeblich das Leistungsverhalten des gesamten DBS
  - Die Anwendungen sind von Änderungen des internen Schemas nicht betroffen (physische Datenunabhängigkeit)