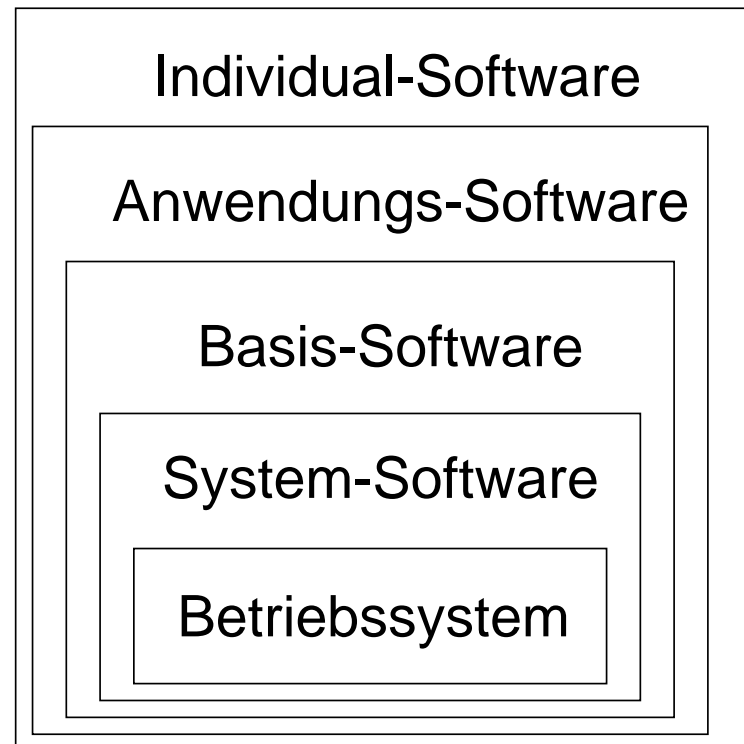


# 1. Grundlegende Konzepte

- ▣➤ Motivation und Historie
- ▣➤ Komponenten und Funktionen
- ▣➤ Einsatzgebiete und Grenzen
- ▣➤ Entwicklungslinien

# Software-Schichten



## Ohne Datenbanken: Datenredundanz

- Basis- oder Anwendungssoftware verwaltet ihre eigenen Daten in ihren eigenen (Datei-)Formaten; *Bsp.:*
  - ◆ Textverarbeitung: Texte, Artikel und Adressen
  - ◆ Buchhaltung: Artikel, Adressen
  - ◆ Lagerverwaltung: Artikel, Aufträge
  - ◆ Auftragsverwaltung: Aufträge, Artikel, Kundenadressen
  - ◆ CAD-System: Artikel, Technische Daten, Technische Bausteine
  - ◆ Produktion: ... , Bestelleingang: ... , Kalkulation: ...
- Daten sind redundant: mehrfach gespeichert; Probleme: Verschwendung von Speicherplatz, "Vergessen" von Änderungen; keine zentrale, "genormte" Datenhaltung

## Ohne Datenbanken: Datenredundanz II

- Andere Software-Systeme (auch Programmiersprachen, Tabellenkalkulation, Dateiverwaltungssysteme, ... ) können große Mengen von Daten nicht effizient verarbeiten
- Mehrere Benutzer oder Anwendungen können nicht parallel auf den gleichen Daten arbeiten, ohne sich zu stören
- Anwendungsprogrammierer / Benutzer können Anwendungen nicht programmieren / benutzen, ohne
  - ◆ interne Darstellung der Daten
  - ◆ Speichermedien oder Rechner (bei verteilten Systemen) zu kennen (**Datenunabhängigkeit** nicht gewährleistet)
- Datenschutz und Datensicherheit sind nicht gewährleistet

## Mit Datenbanken: Datenintegration

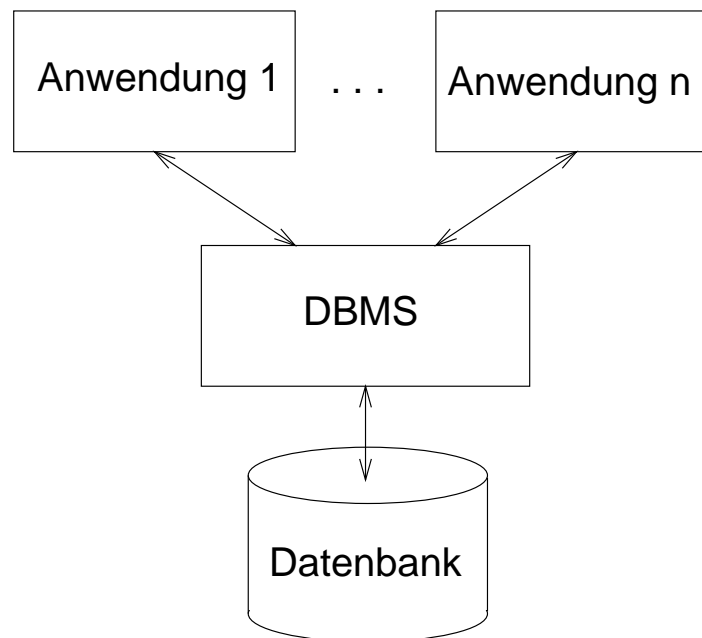
- Die gesamte Basis- und Anwendungssoftware arbeitet auf denselben Daten (Datenbankentwurf, Datendefinition)
- *Bsp.:* Adressen und Artikel werden nur einmal gespeichert
- Auch andere Probleme (Effizienz, Parallelität, Datenschutz, Datensicherheit) werden gelöst
  - ◆ Datenbanksysteme können große Datenmengen effizient verwalten (Anfragesprachen, Optimierung, Interne Ebene)
  - ◆ Benutzer können parallel auf Datenbanken arbeiten (Transaktionskonzept)
  - ◆ Datenunabhängigkeit durch 3-Ebenen-Konzept
  - ◆ Datenschutz (kein unbefugter Zugriff) und Datensicherheit (kein ungewollter Datenverlust) werden vom System gewährleistet

# Historie

- Anfang 60er Jahre: elementare Dateien, anwendungsspezifische Datenorganisation (geräteabhängig, redundant, inkonsistent)
- Ende 60er Jahre: Dateiverwaltungssysteme (SAM, ISAM) mit Dienstprogrammen (Sortieren) (gerateunabhängig, aber redundant und inkonsistent)
- 70er Jahre: Datenbanksysteme (Geräte- und Datenunabhängigkeit, redundanzfrei, konsistent)

# Prinzipien

- DBMS: Datenbank-Management-System
- DBS: Datenbanksystem (DBMS + Datenbank)



# Prinzipien II

## ■ Grundmerkmale

- ◆ verwalten persistente (langfristig zu haltende) Daten
- ◆ verwalten große Datenmengen effizient
- ◆ Datenbankmodell, mit dessen Konzepten alle Daten einheitlich beschrieben werden (Integration)
- ◆ Operationen und Sprachen (DDL, IQL, DML, ... ) deskriptiv, getrennt von einer Programmiersprache
- ◆ Transaktionskonzept, Concurrency Control: logisch zusammenhängende Operationen atomar (unteilbar), Auswirkungen langlebig, können parallel durchgeführt werden
- ◆ Datenschutz, Datenintegrität (Konsistenz), Datensicherheit

## Prinzipien III

- Grundprinzip moderner Datenbanksysteme
  - ◆ 3-Ebenen-Architektur (physische Datenunabhängigkeit, logische Datenunabhängigkeit)
  - ◆ Trennung zwischen Schema (etwa Tabellenstruktur) und Instanz (etwa Tabelleninhalt)

angelehnt an 9 Codd'sche Regeln:

# Die neun Codd'schen Regeln I

## 1. Integration

- einheitliche, nichtredundante Datenverwaltung

## 2. Operationen

- Speichern, Suchen, Ändern

## 3. Katalog

- Zugriffe auf Datenbankbeschreibungen im Data Dictionary

## 4. Benutzersichten

## 5. Konsistenzüberwachung

- auch: Integritätssicherung (Korrektheit des Datenbankinhalts)

## Die neun Codd'schen Regeln II

### 6. Datenschutz

- Ausschluß unauthorisierter Zugriffe

### 7. Transaktionen

- Mehrere DB-Operationen als Funktionseinheit

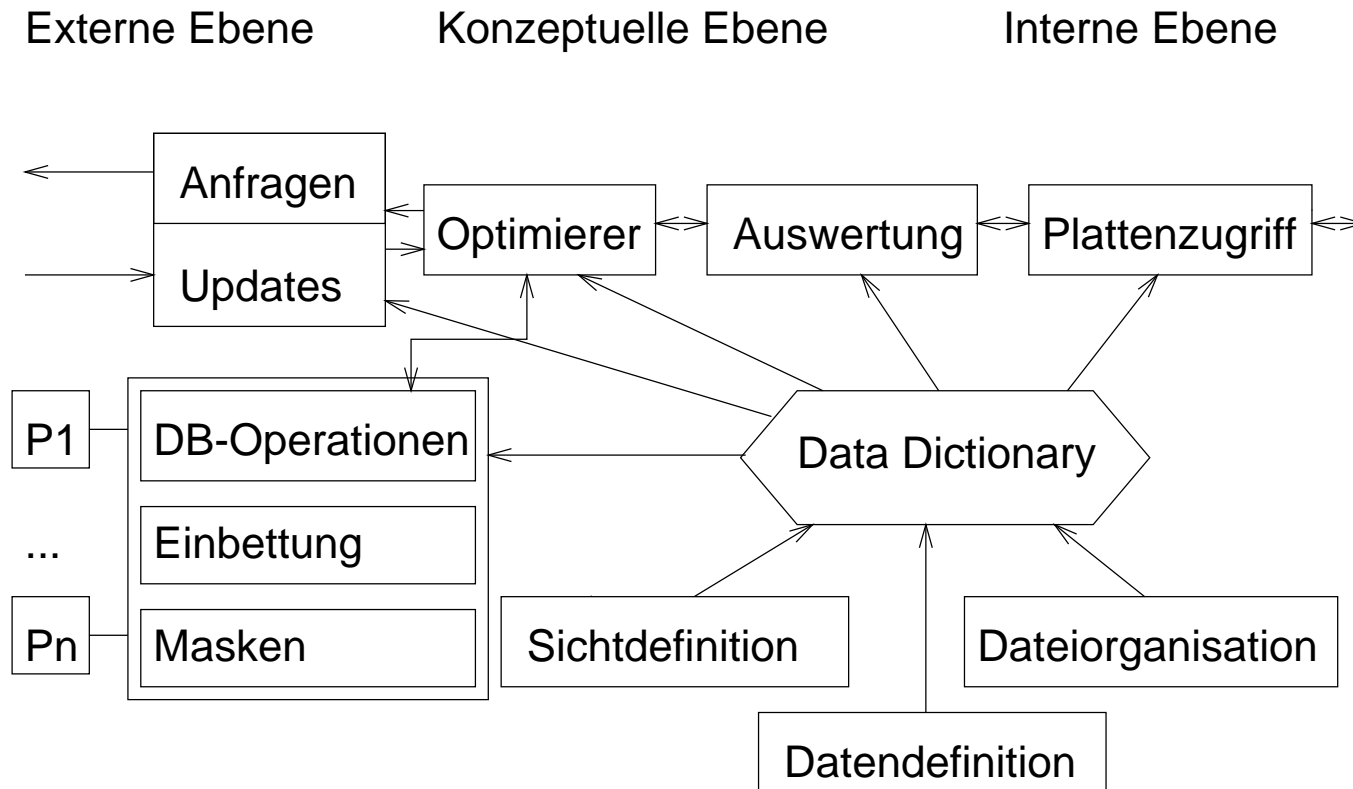
### 8. Synchronisation

- Parallele Transaktionen koordinieren

### 9. Datensicherung

- Wiederherstellung von Daten nach Systemfehlern

# Architektur eines Datenbanksystems I



# Architektur eines Datenbanksystems II

- Dateiorganisation: Definition der (internen) Dateiorganisation und Zugriffspfade
- Datendefinition: Konzeptuelle Datendefinition (konzeptuelles Schema)
- Sichtdefinition: Definition von Benutzersichten (externe Schemata)
- Optimierer und Auswertung: Effiziente Umsetzung dieser Operationen
- Plattenzugriff: Plattenzugriffssteuerung (genauer: Fünf-Schichten-Architektur)

## Architektur eines Datenbanksystems III

- Masken: Entwurf von Menüs und Masken
- DB-Operationen: Anfrage- und Änderungs-Operationen
- Einbettung: Einbettung dieser Operationen in Anwendungsprogramme
- $P_1, \dots, P_n$ : Verschiedene Anwendungsprogramme

# Modell für die konzeptuelle Ebene: Relationenmodell I

Konzeptuell ist die Datenbank eine Menge von Tabellen

AUSLEIH

INV.NR	NAME
4711	Meyer
1201	Schulz
0007	Müller
4712	Meyer

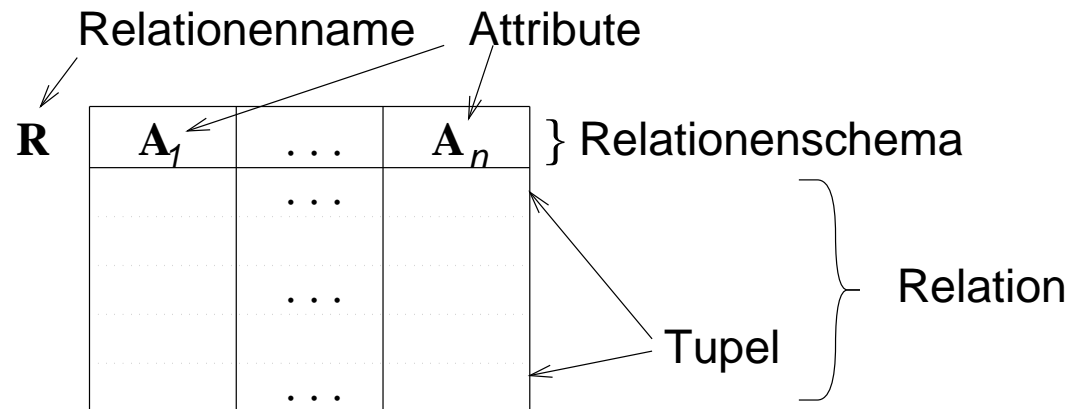
BUCH

INV.NR	TITEL	ISBN	AUTOR
0007	Dr. No	3-125	James Bond
1201	Objektbanken	3-111	Heuer
4711	Datenbanken	3-765	Vossen
4712	Datenbanken	3-891	Ullman
4717	PASCAL	3-999	Wirth

Tabellen = 'Relationen'

# Modell für die konzeptuelle Ebene: Relationenmodell II

- **Fett** geschriebene Zeilen: *Relationenschema*
- Weitere Einträge in der Tabelle: *Relation*
- Eine Zeile der Tabelle: *Tupel*
- Eine Spaltenüberschrift: *Attribut*



# Relationenmodell: Integritätsbedingungen

AUSLEIH	INV.NR	NAME
	4711	Meyer
	1201	Schulz
	0007	Müller
	4712	Meyer

BUCH	INV.NR	TITEL	ISBN	AUTOR
	0007	Dr. No	3-125	James Bond
	1201	Objektbanken	3-111	Heuer
	4711	Datenbanken	3-765	Vossen
	4712	Datenbanken	3-891	Ullman
	4717	PASCAL	3-999	Wirth

## Relationenmodell: Integritätsbedingungen II

- Relationenschema + lokale Integritätsbedingungen
  - ◆ INVENTARNR ist *Schlüssel* für BUCH
  - ◆ d.h.: eine INVENTARNR darf nicht doppelt vergeben werden
  - ◆ d.h.: in Spalte INVENTARNR dürfen keine zwei gleichen Werte auftauchen
  
- Datenbankschema ist Menge von Relationenschemata + globale Integritätsbedingungen
  - ◆ INVENTARNR in AUSLEIH ist *Fremdschlüssel* bezüglich BUCH
  - ◆ d.h.: INVENTARNR taucht in einem anderen Relationenschema als Schlüssel auf
  - ◆ d.h.: die Inventarnummern in AUSLEIH müssen auch in BUCH auftreten

# Anfrageoperationen I

- SELEKTION: Zeilen (Tupel) auswählen  
*SEL [NAME = 'Meyer'] (AUSLEIH)* ergibt:

<b>INVENTARNR</b>	<b>NAME</b>
4711	Meyer
4712	Meyer

## Anfrageoperationen II

- PROJEKTION: Spalten (Attribute) auswählen  
*PROJ [INVENTARNR, TITEL] (BUCH)* ergibt:

<b>INVENTARNR</b>	<b>TITEL</b>
0007	Dr. No
1201	Objektbanken
4711	Datenbanken
4712	Datenbanken
4717	PASCAL

Achtung: doppelte Tupel werden entfernt!

## Anfrageoperationen III

- VERBUND (JOIN): Tabellen verknüpfen über gleichbenannte Spalten und gleiche Werte

```
PROJ [INVENTARNR, TITEL](BUCH)
      JOIN
      SEL [NAME = 'Meyer'](AUSLEIH).
```

ergibt:

<b>INVENTARNR</b>	<b>TITEL</b>	<b>NAME</b>
4711	Datenbanken	Meyer
4712	Datenbanken	Meyer

## Anfrageoperationen IV

- Weitere Operationen: Vereinigung, Differenz, Durchschnitt, Umbenennung
- Alle Operationen beliebig kombinierbar (“Algebra”)

# Sprachen und Sichten I

## Abfragesprache

- Interaktive Möglichkeit, Datenbankabfragen zu formulieren und zu starten
- Relationenalgebra + Funktionen (SUM, MAX, MIN, COUNT, ... ) + arithmetische Operationen
- eventuell graphisch “verpackt”

## SQL als Standard

```
select BUCH.INVENTARNR, TITEL, NAME  
from BUCH, AUSLEIH  
where NAME = 'Meyer' and  
        BUCH.INVENTARNR = AUSLEIH.INVENTARNR
```

# Sprachen und Sichten II

## Änderungs-Komponente

- Interaktive Möglichkeit
  - ◆ Tupel einzugeben
  - ◆ Tupel zu löschen
  - ◆ Tupel zu ändern
  
- Lokale und globale Integritätsbedingungen werden geprüft!

## Sprachen und Sichten III

### Definition von Benutzersichten

- Häufig vorkommende Datenbankabfragen können unter einem “Sichtnamen” als “virtuelle” Tabelle gespeichert werden.
- Bsp.:

```
MEYERS := PROJ [INVENTARNR, TITEL](BUCH)
        JOIN
        SEL [NAME = 'Meyer'](AUSLEIH),
```

ergibt Tabelle von vorhin; ansprechbar wie BUCH oder AUSLEIH über Sichtnamen MEYERS.

# Optimierer I

Problem:

Finde einen Relationenalgebra-Ausdruck, der äquivalent ist (“das gleiche Ergebnis liefert”) wie der gegebene, aber effizienter auszuwerten ist

# Optimierer II

Eine Möglichkeit: algebraische Optimierung

- allgemeine Regel:

1.  $SEL [A = Konst] ( REL1 JOIN REL2 )$  und A aus REL1
2.  $SEL [A = Konst] (REL1) JOIN REL2$

sind äquivalent

- allgemeine Strategie: Selektionen möglichst früh, da sie Tupelanzahlen in Relationen verkleinern

## Optimierer: Beispiel

- REL1 100 Tupel, REL2 50 Tupel  
intern: Tupel sequentiell abgelegt
  1.  $5000 \text{ (JOIN)} + 5000 \text{ (SEL)} = 10000 \text{ Operationen}$
  2.  $100 \text{ (SEL)} + 10 \cdot 50 \text{ (JOIN)} = 600 \text{ Operationen}$   
falls 10 Tupel in REL1 die Bedingung  $A = \text{Konst}$  erfüllen

# Interne Strukturen

- Dateiorganisationsformen
- Relation kann intern als Datei wie folgt abgespeichert werden:
  - ◆ Heap, ungeordnet
  - ◆ Sequentiell, geordnet nach bestimmter Spalte oder Spaltenkombination
  - ◆ Hash-organisiert, gestreut gespeichert, Adreßberechnung durch Formel
  - ◆ Baumartig, Tupel in einem Suchbaum angeordnet
  - ◆ ...

## Interne Strukturen II

Zusätzliche Zugriffspfade:

- statischer Index, einstufig oder mehrstufig
- dynamischer Index

Zwischen Dateiorganisationen/Zugriffspfaden kann beliebig gewechselt werden, ohne Auswirkungen auf konzeptuelle/externe Ebene

**ACHTUNG:** je schneller die Abfrage, desto langsamer der Update

## Zugriffe auf Plattenseiten

Jede Operation (SEL, PROJ, JOIN, ... ) wird nun in optimale Folge von Seitenzugriffen umgesetzt

Dabei werden:

- Zugriffspfade und Dateiorganisation ausgenutzt, wenn es dem System sinnvoll erscheint
- Reihenfolge der Zugriffe nach vorliegenden Zugriffspfaden bestimmt.

## Zugriffe auf Plattenseiten II

Beispiel: *SEL [ NAME = 'Meyer' ] ( SEL [ INVENTARNUMMER > 4500 ] ( AUSLEIH ))*

- Annahme: auf NAME ist ein Zugriffspfad definiert, auf INVENTARNUMMER nicht
- System ändert die Reihenfolge der Selektionen!!

# Einsatzgebiete und Grenzen

Klassische Einsatzgebiete:

- viele Objekte (15000 Bücher, 300 Benutzer, 100 Ausleihvorgänge pro Woche, ... )
- wenige Objekttypen (BUCH, BENUTZER, AUSLEIHUNG)
- etwa Buchhaltungssysteme, Auftragserfassungssysteme, Bibliothekssysteme, ...

# Einsatzgebiete und Grenzen

Normalerweise sind herkömmliche Datenbanksysteme überfordert mit:

- CAD- oder andere technische Anwendungen (viele Objekte, viele Objekttypen, sehr strukturierte Objekte)
  - ◆ ABER: Objektorientierte Datenbanksysteme
- Expertensysteme (wenige Objekte, viele Objekttypen, kompliziertere Operationen)
  - ◆ ABER: Deduktive Datenbanksysteme

## Wann kommt was?

- Datendefinition (Datenbankentwurf): Kapitel 3 – 7
- Dateiorganisation: Datenbanken II
- Integrität, Sichtdefinition, Datenschutz: Kapitel 12 – 14
- Data Dictionary: Datenbanken II
- Plattenzugriffssteuerung, Auswertung, Optimierung:  
Datenbanken II
- Anfragen, Updates: Kapitel 8 – 10
- Datenbankprogrammierung: Kapitel 11

# Sprachen und Verantwortliche

- Datendefinition:
  - ◆ DDL (Data Definition Language) → DBA (Datenbankadministrator)
- Dateiorganisation:
  - ◆ SSL (Storage Structure Language) → Systemadministrator
- Sichtdefinition:
  - ◆ SDDL (Subscheme Data Definition Language), VDL (View Definition Language) → Anwendungsadministrator

## Sprachen und Verantwortliche II

- Anfragen:
  - ◆ IQL (Interactive Query Language), mit Updates  
DML (Data Manipulation Language) → “anspruchsvoller Laie”
- Datenbankprogrammiersprache:
  - ◆ DBPL (Data Base Programming Language) → Anwendungsprogrammierer
- Datenbankprogramme  $P_1, \dots, P_n$  → “parametric user”

# Entwicklungslinien I

60er Jahre

- DBS basierend auf hierarchischem Modell, Netzwerkmodell
  - ◆ Zeigerstrukturen zwischen Daten
  - ◆ Schwache Trennung interne / konzeptuelle Ebene
  - ◆ Navigierende DML
  - ◆ Trennung DML / Programmiersprache

# Entwicklungslinien II

70er und 80er Jahre

- Relationale Datenbanksysteme
  - ◆ Daten in Tabellenstrukturen
  - ◆ 3-Ebenen-Konzept
  - ◆ Deklarative DML
  - ◆ Trennung DML / Programmiersprache

## Entwicklungslinien III

(80er und) 90er Jahre

### ■ Wissensbanksysteme

- ◆ Daten in Tabellenstrukturen
- ◆ Stark deklarative DML
- ◆ Integrierte Datenbankprogrammiersprache

### ■ Objektorientierte Datenbanksysteme

- ◆ Daten in komplexeren Objektstrukturen  
(Trennung Objekt und seine Daten)
- ◆ Deklarative oder navigierende DML
- ◆ Oft integrierte Datenbankprogrammiersprache
- ◆ Oft keine vollständige Ebenentrennung