Vorlesung

DATENBANKSYSTEME

für Studiengang Informatik
und
für Studiengang Ingenieurinformatik

TU Ilmenau
Fachgebiet Datenbanken und Informationssysteme
Dr. R. Schindler

Wintersemester 2001/02

Literatur

- Alfons Kemper, Andre Eickler
  Datenbanksysteme. Eine Einführung.
  Oldenbourg, 4., überarb. u. erw. Aufl. 2001

- Andreas Heuer, Gunter Saake
  Datenbanken. Konzepte und Sprachen.
  MITP, Bonn, 2., aktualis. u. erw. Aufl. 2000

- Gottfried Vossen
  Datenbankmodelle, Datenbanksprachen und Datenbankmanagementsysteme
  Oldenbourg, 4., korr. u. erg. Aufl. 2000

Weitere Namen:

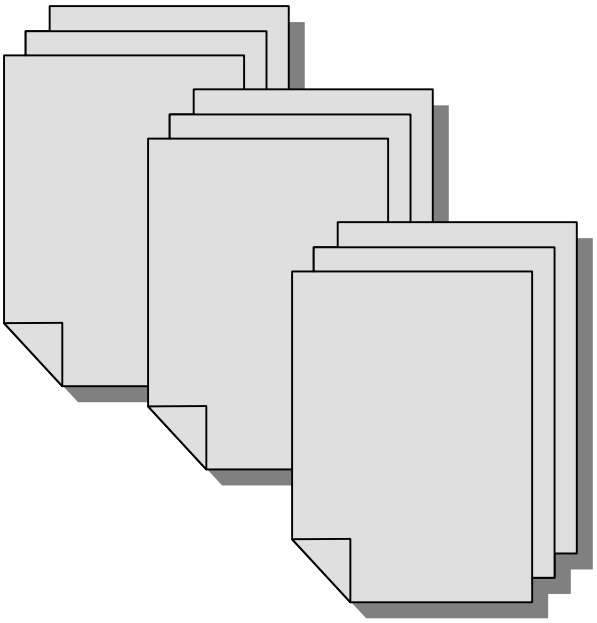
- Date
- Ullman / Widom
- Korth / Silberschatz
- Elmasri / Navathe

Weitere Bücher:

- DB-Handbuch
- Informatik-Handbuch

Einführung

Flut von Daten, Informationen, Dokumenten, Bildern, ...



1.2347	9.9	4.6578	100.34
1.2355	9.0	4.6543	102.01
1.2423	9.4	3.9999	101.17
1.1989	9.0	3.9999	100.97
1.2333	9.5	4.5760	101.03
1.2111	9.6	4.6234	101.23
1.2387	9.7	4.6578	100.98
1.2756	9.8	4.7001	100.85
1.2345	9.7	4.6998	101.97
1.3001	-	4.6798	101.44
1.2001	-	4.6002	101.09

21.02.2001

3.1754

Sachbuch

PI

alstr. 17

1,7

2,7

3,7

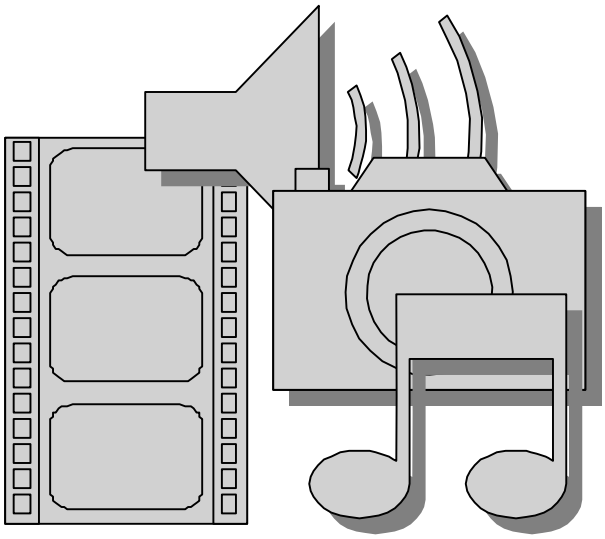
3,3

1,3

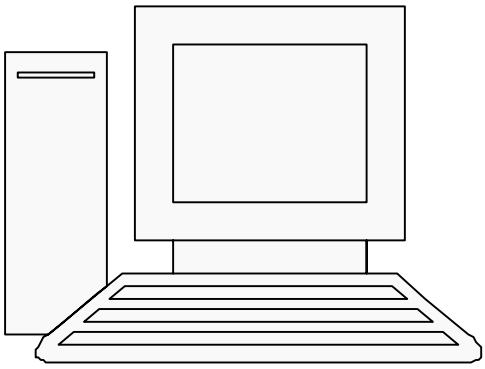
Müller

Wetteraussichten

- **bewölkt**
- **Regen**
- **kalt**



Ablegen (Speichern) und Wiederfinden



Dateisystem

- direkte Arbeit auf den physischen Speicherstrukturen
- sehr viele Daten - Suche auf Datei ineffizient
- Lesezugriff auf ganze Datei - Datenschutz
- Problem : gleichzeitige Änderungen
- Problem : korrektes Rücksetzen nach Absturz
- komfortable Datenverwaltung **nur** in der Anwendung

Datenbank

- (logisches) Datenmodell und DDL, DML
- Anfragesprache mit einfachen, generischen Operationen
- Datenschutz
- Datensicherheit (Rücksetzen, Sichern)
- Mehrbenutzerbetrieb
- Externspeicherorganisation und –verwaltung
- Sichten
- Integritätskontrolle
- Anfrage-Optimierung

↷ **persistente Datenhaltung**

1. Konzeptuelle Datenmodellierung

Konzeptuelle Datenmodelle

- viele semantische Datenmodelle

- hier :
  Entity-Relationship Model(l) (ERM)

  - 1976 P. P. Chen

- fungiert heute faktisch als Standardmodell für den Entwurf von Datenbankentwicklungen

- für erste Modellierung der Semantik der Anwendungs“welt“

- bisher viele Erweiterungen verschiedener Autoren

  - **Extended Entity-Relationship Model (EERM)**

- graphisch :
  Entity-Relationship Diagram(m) (ERD)

Konzepte des ERM

- **Objekt**
  Informationseinheit ;
  ein “Ding“ der realen Welt mit einer unabhängigen (physischen oder konzeptuellen) Existenz

- **Attribut**
  Eigenschaft, Merkmal (eines Objekts oder einer Beziehung)

- **Wert**
  Beschreibung der Eigenschaften durch Attributwerte ; sind später ein Großteil der in der DB gespeicherten Daten

- **Attribute und Werte**
  - atomar
    zusammengesetzt
  - einwertig
    mehrwertig
  - gespeichert
    abgeleitet
  - NULL-Werte

- **Domäne**
  Wertevorrat für ein Attribut

- **Objekttyp**
  Menge von Objekten mit gleichen Attributen ;
  Intension - Schema, d.h. Name und Liste von Attributen
  Extension - Objektmenge

- **Schlüsselattribut**
  Attribute eines Objekttyps, deren Werte unterschiedlich sind für jedes einzelne Objekt einer Objektmenge, d.h. eindeutig, **identifizierend**

- **Beziehungstyp**
  über n Objekttypen (häufig n = 2) ;
  beschreibt eine Menge von Beziehungen zwischen Objekten dieser Typen

- **Beziehung**
  ist eine Verbindung zwischen Objekten, die genau ein Objekt aus jeder der am Beziehungstyp beteiligten Objektmengen enthält

- **Rekursive Beziehungen**
  Beziehung zwischen unterschiedlichen Objekten des gleichen Objekttyps ;
  d.h. in manchen Fällen kann der gleiche Objekttyp mehrfach in den Beziehungstyp eingehen, aber in verschiedenen **Rollen**

- **Beziehungen als Attribute**
  Tatsache, dass Beziehungen zu anderen Objekten bestehen, wird als Attribut dem Objekttyp zugeordnet **(Achtung!)**

Restriktionen für Beziehungstypen

begrenzen Möglichkeiten der Objekt-Kombinationen für konkrete Beziehungen innerhalb eines Typs entsprechend der Semantik der Anwendungswelt

1. **Funktionalität**
   legt die Anzahl der Beziehungen mit Objekten der „Gegenseite“ für ein Objekt fest

   1:1 z.B. Professor **leitet** Lehrstuhl

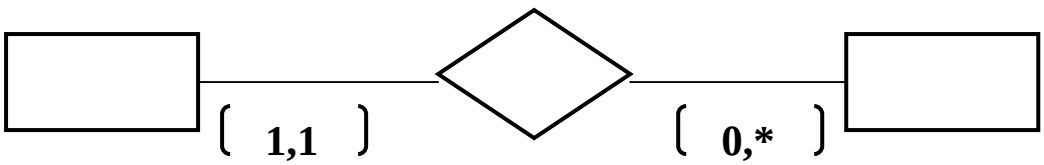
   1:N z.B. Student **leiht_aus** Bücher

   M:N z.B. Studenten **hören** Vorlesungen

2. **Kardinalität**
   spezifiziert die Teilnahme von Objekten an Beziehungen eines Typs:

   ⟨ **min, max** ⟩

   d.h.
   min Objekte des Objekttyps sind an Ausprägungen (Beziehungen) des Beziehungstyps beteiligt und höchstens **max** Objekte



- **Schwacher Objekttyp**
  Objekttyp, der über eine spezifische Beziehung an einen anderen (Eigner-)Objekttyp gekoppelt ist;
  die Existenz eines schwachen Objekts ist abhängig von der Existenz seines Eigner-Objekts;

  Partieller Schlüssel
  Attributmenge zur eindeutigen Identifizierung der abhängigen Objekte **eines** Eigner-Objekts

  Schlüssel des schwachen Objekts (global)
  partieller Schlüssel des schwachen Objekts **plus** Schlüssel des Eigner-Objekts

- **Spezialisierung (IS_A)**
  Objekte der Spezialisierung sind eine Teilmenge der Allgemeinheit (Obermenge);
  Objekte der Untermenge sind durch zusätzliche, spezielle Eigenschaften ausgezeichnet und unterscheiden sich damit auch von anderen Untermengen;
  zu jedem Objekt der Untermenge gibt es ein entsprechendes Objekt in der Obermenge mit den konkreten, aber allgemeingültigen Eigenschaften (Attributen)

  Schlüssel
  wird von der Obermenge geerbt

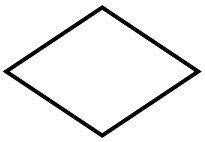
- **Beziehungen höheren Grades**
  zwischen mehr als zwei Objekttypen

Notation des ERD

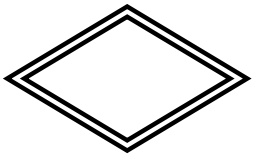
Objekttyp

Schwacher Objekttyp

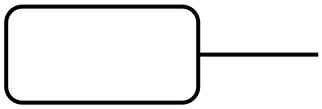
Beziehungstyp



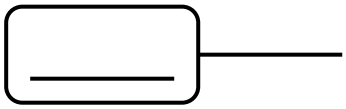
Identifizierender Beziehungstyp für schwachen Objekttyp



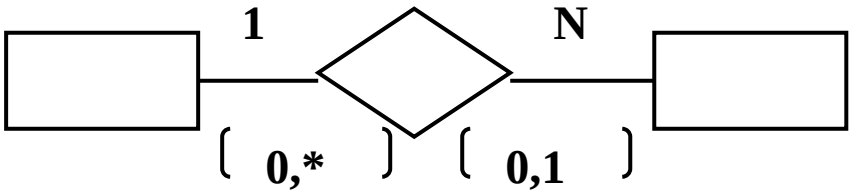
Attribut



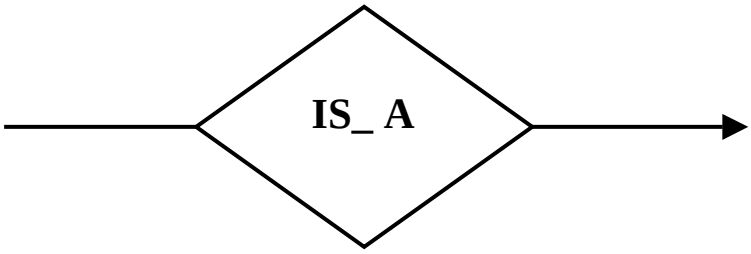
Schlüsselattribut



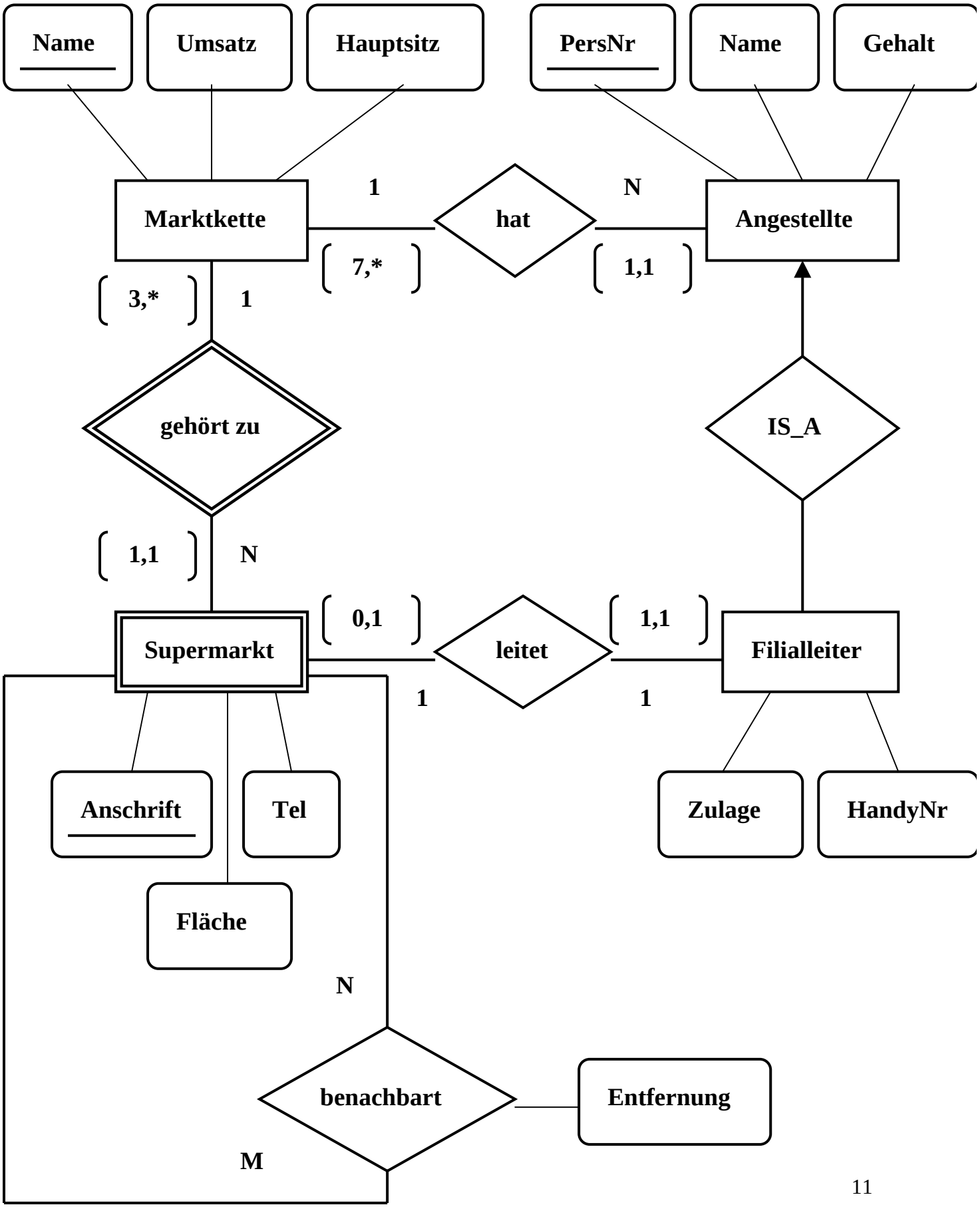
Beziehungstyp mit Funktionalität und Kardinalität



Spezialisierung



ERD am Beispiel



2. Logische Datenmodellierung

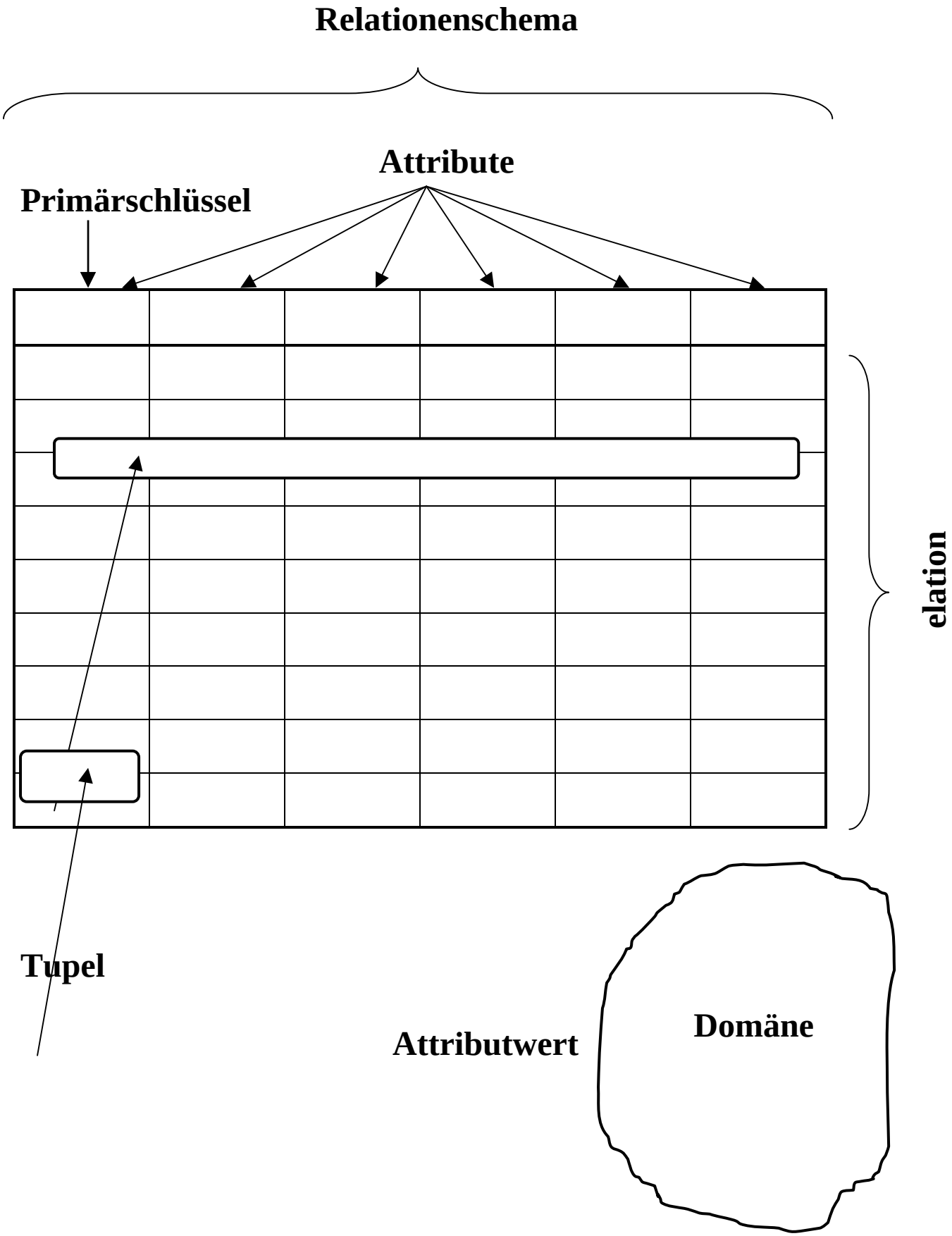
Das relationale Datenmodell (RDM)

- E.F.Codd (IBM) - 1976
- „jüngstes“ unter den klassischen Datenmodellen
- weit verbreitet
- große Palette von „Produkten“, basierend auf diesem Datenmodell:
  DB2, ORACLE, INGRES, INFORMIX, SYBASE, …
  MS-SQL-Server, Gupta SQL-Base, …
  MS-Access, Paradox, dBase, ...

- repräsentiert die Datenbank „nach außen” als eine Menge von Tabellen (Relationen)
- jede Zeile einer Tabelle beschreibt über die Werte in jeder Spalte **Objekte** oder **Beziehungen**

- Relationenalgebra - Operationenteil des RDM
  Menge von generischen Operationen;
  vollständig;
  Grundlage von Anfragesprachen

Das relationale Datenmodell



Modellierungskonzepte des RDM

- **Attribut A**
  Merkmal;
  wie im ERM

- **Domäne D**
  Menge möglicher **atomarer** Werte für ein Attribut, d.h. Wertevorrat;

  $D = dom(A_i)$

- **Relationenschema $R(A_1, A_2, \ldots , A_n)$**
  Liste (geordnete Menge) der beschreibenden Attribute;
  R - Name der Relation
  n - Grad der Relation (Anzahl der Attribute)

- **Relation r(R)**
  endliche Menge von n-Tupeln über der Liste von Attributen des Schemas ;

  $r = \{ t_1, t_2, \ldots , t_m \}$

  endliche Untermenge des kartesischen Produkts der Domänen der Relation

  $r(R) \subseteq (dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n))$

- **n-Tupel t**
  Liste (geordnete Menge) von n Werten aus den entsprechenden Domänen der Attribute oder NULL-Werte

- **Primärschlüssel PK**
  kleinste, identifizierende Attribut(menge) für Tupel

Eigenschaften, Bedingungen, Restriktionen des RDM

- Keine Ordnung für Tupel auf logischer Ebene
  (nur für Speicherung auf physischer Ebene)
  ABER
  Ordnung der Werte in einem Tupel gemäß der Ordnung der Attribute im Schema

- **Schlüssel**

Superschlüssel:	identifizierende Attributmenge
Schlüssel:	kleinste, identifizierende Attributmenge
Schlüsselkandidat:	wenn mehrere Schlüssel
Primärschlüssel:	ausgewählter Schlüssel

- **Fremdschlüssel FK**
  „normales“ Attribut einer Relation, das in einer anderen (fremden) Relation auch vorkommt und dort *Schlüssel* ist;
  weitere Voraussetzungen:
  - Attribute haben die gleichen Domänen
  - Fremdschlüssel-Wert eines Tupels muss auch als Primärschlüssel-Wert in der anderen Relation vorhanden sein

- Objektintegrität
  kein Primärschlüssel-Wert kann NULL sein

- Referentielle Integrität
  Schlüssel - Fremdschlüssel - Beziehung

Relationaler DB-Entwurf: Vom ERM zum RDM

- **Objekttyp E**
  ⇒ Relation R

  A_i alle einfachen Attribute von E
  (bei zusammengesetzten Attributen nur die einfachen Komponentenattribute)

  PK einen Schlüssel von E zum PK erklären

- **Beziehungstyp R (binär)**
  ⇒ Relation S
  (zusätzlich zu den Relationen für die beiden Objekttypen)

  A_i Primärschlüsselattribute der an der Beziehung beteiligten Objekttypen (als Fremdschlüssel FK)
  UND
  einfache Attribute von R

  PK in Abhängigkeit von Funktionalität und Kardinalität:

M:N	alle FK zusammen
1:N, partiell	nur FK der N-Seite
1:1, partiell	ein FK beliebiger Seite

- **Beziehungstyp R (1:1** bzw. **1:N, total)**
  ⇒ Auswahl einer bzw. der N-Seite entsprechenden Relation und Hinzufügen des PK des anderen an der Beziehung beteiligten Objekttyps
  (Relation für Beziehungstyp entfällt dann)

- **Abhängiger Objekttyp W (von E)**
  ⇒ Relation R

  A_i alle einfachen Attribute von W
  UND
  Primärschlüsselattribut(e) des Eigner-Objekttyps E
  PK Primärschlüssel von E
  UND
  partieller Schlüssel von W

- **Mehrwertiges Attribut A (der Relation E)**
  ⇒ Relation R

  A_i ein A entsprechendes Attribut
  UND
  Primärschlüsselattribut(e) K von E
  PK A plus K

- **Mehrfach-Beziehungstyp R (n > 2)**
  ⇒ Relation S

  A_i Primärschlüsselattribute aller an der Beziehung beteiligten Objekttypen (als Fremdschlüssel)
  UND
  einfache Attribute von R
  PK (bis zu) alle Fremdschlüsselattribute zusammen

- **Spezialisierung IS_A**
  ⇒ Relation S (für übergeordneten Objekttyp)
  mit Primärschlüssel K und weiteren Attributen

  ⇒ Relationen R_i für die Spezialisierungen

  A_i Primärschlüssel des übergeordneten Objekttyps
  UND
  jeweilige Attribute des speziellen Objekttyps
  PK Primärschlüssel K von S

Relationales DBschema für ERD aus Beispiel

Angestellte (PersNr , Name , Gehalt)
, MkName)

Filialleiter (PersNr , Zulage , HandyNr)
, MkName , Anschrift)

Marktkette (MkName , Umsatz , Hauptsitz)

Supermarkt (MkName , Anschrift , Fläche , Tel)

hat (MkName , PersNr)

leitet (MkName , Anschrift , PersNr)

benachbart (MkName , Anschrift ,

MkNameB , AnschriftB , Entfernung)

Relationenalgebra

- Operationenteil des RDM
- Menge von implizit im Modell enthaltenen Operationen
  ⇒ **generische Operationen**
- Implizit: nicht anwendungsspezifisch

- Anfragesystem
  (es gibt weitere für das RDM)
- Grundlage für Anfragesprachen

- **Vollständig**
  ⇒ Maßstab für Anfragesysteme und -sprachen

- **Operationen**
  Selektion
  Projektion
  Natürlicher Verbund
  Mengenoperationen
  Umbenennung
- Ergebnis: wieder eine **Relation**

Selektion

- selektiert alle Tupel (Zeilen) einer Relation (Tabelle), die der formulierten Bedingung genügen

- Bezeichnung: $\sigma_{P}(r)$

- gegebene Relation

  $r(A_1, A_2, \ldots, A_n)$

- Ergebnisrelation

  $q(A_1, A_2, \ldots, A_n)$
  mit allen Tupeln, die die Bedingung P erfüllen

- Bedingung P
  Attribut – Attribut – Vergleich
  Attribut – Konstanten – Vergleich
  logische Verknüpfung von Vergleichen mit $\wedge$, $\vee$, $\neg$

- **Beispiel**
  Gesucht sind alle Studenten einer bestimmten Studienrichtung aus der Studenten-Relation

  $\sigma_{\text{Studienrichtung = Chemie}}(\text{Student})$

Projektion

- Projiziert alle gewünschten Attribute und ihre Werte (Spalten) aus einer Relation (Tabelle)

- Bezeichnung: $\pi_{A}(r)$

- gegebene Relation

  $r(A_1, A_2, \ldots, A_n)$

- Ergebnisrelation

  $q(A_1, A_2)$

  mit allen Werten der Attribute A_1 und A_2
  ohne Wiederholungen

  (unter der Annahme: $\pi_{A1, A2}(r)$)

- Projektionsliste A
  Liste von Attributen aus dem Schema der Relation r

- **Beispiel**
  Gesucht sind nur die Angaben zu Name und Anschrift der Studenten aus der Studenten-Relation

  $\pi_{\text{Name, Anschrift}}(\text{Student})$

z.B.

Name	Anschrift
Schulz	IL, Waldstr.
Schulz	IL, Talstr.

keine Wiederholung !

Mengenoperationen

- gegebene Relationen

  $r (A_1, A_2, \ldots , A_n)$

  $s (A_1, A_2, \ldots , A_n)$
- Ergebnisrelation

  $q (A_1, A_2, \ldots , A_n)$

Vereinigung

- Ergebnisrelation mit allen Tupeln aus der Relation r **und** allen Tupeln aus der Relation s (ohne Duplikate)

- Bezeichnung: **r ∪ s**

Durchschnitt

- Ergebnisrelation mit den Tupeln, die **sowohl** in der Relation r **als auch** in der Relation s vorkommen

- Bezeichnung: **r ∩ s**

Differenz

- Ergebnisrelation mit den Tupeln der Relation r, die **nur** in r **und nicht** auch noch in s vorkommen

- Bezeichnung: **r – s**

Kartesisches Produkt

- stellt alle nur möglichen Kombinationen der Tupel zweier Relationen her

- Bezeichnung: **r** $\times$ **s**

- gegebene Relationen

  r (A_1, A_2, ... , A_n)

  s (B_1, B_2, ... , B_m)

- Ergebnisrelation

  q (A_1, A_2, ... , A_n, B_1, B_2, ... , B_m)
  mit allen nur denkbaren Tupel-Kombinationen über dem vereinigten Schema der Ausgangsrelationen

- ergibt (sehr schnell) **sehr große** Ergebnisrelation

- in der Praxis semantisch selten sinnvoll

- **Beispiel**

  Ein Autohersteller hat u. a. zwei Relationen
  Typen und Farben.
  Jeder Autotyp wird in allen Farblackierungen angeboten.

  Typen $\times$ **Farben**

Natürlicher Verbund

- stellt nur „sinnvolle“ Kombinationen von Tupeln zweier Relationen über gleichen Attributen mit gleichen Werten her

- Bezeichnung: **r** |><| **s**

- gegebene Relationen

  r (A_1, A_2, ... , A_k)

  s (A_k, A_{k+1}, ... , A_n)

- Ergebnisrelation

  q (A_1, A_2, ... , A_k, A_{k+1}, ... , A_n)

  mit den Tupel-Kombinationen, die unter A_k in r und s jeweils den **gleichen** Wert aufweisen

- häufig gebräuchliche Operation zum Verknüpfen von Daten über den Relationen der Datenbank

- kartesisches Produkt, wenn keine gemeinsamen Attribute

- **Beispiel**

  Eine Bibliothek hat u.a. zwei Relationen

  Ausleihe (RegNr , Rückgabe , NutzerNr)

  Nutzer (NutzerNr , Name , Vorname , Anschrift)

  Für Mahnungen sind Name und Anschrift von Nutzern gesucht, die ein bestimmtes Rückgabedatum überzogen haben.

Verbund

- stellt Kombinationen von Tupeln zweier Relationen unter einer selbst formulierten Verknüpfungsbedingung her

- Bezeichnung: **r [P] s**

- gegebene Relationen

  r (A_1, A_2, ... , A_n)

  s (B_1, B_2, ... , B_m)

- Ergebnisrelation

  q (A_1, A_2, ... , A_n, B_1, B_2, ... , B_m)
  mit nur den Tupel-Kombinationen, die der Verknüpfungsbedingung genügen

- Bedingung P
  Attribut – Attribut – Vergleich
  mit: $<$, $>$, $\leq$, $\geq$, $\neq$

- **Beispiel**
  Gegeben seien die beiden Relationen
  Urlaubsziele (Ort , Zeitraum , Art , Preis)
  Kunden (KdNr , Name , Tel , Vorliebe , Budget)
  Gesucht sind Urlaubsangebote für die Kunden, wobei die Art des Urlaubs den Vorlieben des Kunden entspricht, der Preis aber das verfügbare Budget nicht überschreitet.

Division

- ergibt Tupel einer Relation, die über einem Teil des Schemas mit einem vorgegebenen „Muster“ (Relation über diesem Teil des Schemas) übereinstimmen

- Bezeichnung: **r ÷ s**

- gegebene Relationen
  r (Z) wobei Z – Menge von Attributen
  s (X) wobei $X \subseteq Z$, $Y = Z - X$
- Ergebnisrelation
  q (Y)
  mit Tupeln von r über dem „Rest“-schema Y, die auf dem „Teil“-schema X vollständig mit allen Tupeln von s übereinstimmen

- **Beispiel**
  Im Prüfungsamt gibt es u.a. zwei Relationen
  Bestandene_Prüfungen (StudNr , Name , Fach)
  Vordiplom (Fach)
  Gesucht sind die Studenten, die alle Anforderungen für das Vordiplom erfüllt haben.

Umbenennung

- keine Operation im eigentlichen Sinne;
  ersetzt nur eine Attributbezeichnung im Schema der Relation durch eine neue Bezeichnung

- Bezeichnung: $\delta_{\mathbf{A \leftarrow B}}\mathbf{(\ r\ )}$

- gegebene Relation
  r (A, C, D, ...)
- Ergebnisrelation
  r (B, C, D, ...)
  mit den Tupeln von r **unverändert**

- **Beispiel**
  Im Prüfungsamt gibt es u.a. die Relation
  Studenten (<u>StudNr</u> , Name , ...)
  Zur Abstimmung mit anderen Fakultäten soll die Attributbezeichnung für die Studiennummer geändert werden
  Studenten (<u>StudienNr</u> , Name , ...)

Relationenkalkül

- **Logikbasierter Ansatz** für DBanfragen

  - Datenbankinhalte
    entsprechen
    Belegung von Prädikaten einer Logik
  - Anfragen
    entsprechen
    abgeleiteten Prädikaten

- **deskriptive Anfragen**

  da bei der Definition abgeleiteter Prädikate
  angegeben wird, **WAS** zum Ergebnis gehören soll,
  und nicht **WIE** es berechnet wird
  (siehe Algebra - prozedural)

$\Rightarrow$ Anfrage-Kalkül

formale logische Sprache zur Fomulierung
von Anfragen

1. **Tupelkalkül**

⇒ Anfragen

$$\{ t \mid \Phi (t) \}$$

wobei

- t - Tupelvariable
- $\Phi (t)$ - spezielle Formel der Prädikatenlogik

2. **Domänenkalkül**

⇒ Anfragen

$$\{x_1 \ldots x_n \mid \Phi (x_1, \ldots, x_n)\}$$

wobei

- ist eine Formel über den in der Ergebnisliste aufgeführten Variablen
- Das Ergebnis ist eine Menge von Tupeln. Die Tupelkonstruktion erfolgt implizit aus den Werten der Variablen in der Ergebnisliste.

3. Nutzung von Datenbanken

- DBMS bietet Möglichkeit zur Nutzung der in der Datenbank gespeicherten Daten über die Formulierung entsprechender **Anfragen** in einer **Anfragesprache**

- **Anfragen**
  sind
  - die Suche bzw. Auswahl von Daten nach vorgegebenen Kriterien und/oder
  - einfache Verknüpfungsoperationen über Daten

- **Anfragesprache**
  Ausdrucksmittel zur Formulierung von Anfragen auf höherer Ebene entsprechend dem logischen Datenmodell

- **Beispiel**
  Gesucht sind Angaben (Studiennummer, Name und Fakultät) zu Studenten, die die Regelstudienzeit von 10 Semestern überschritten haben.

```
SELECT  StudNr, Name, Fakultaet
  FROM  Student
WHERE  Semester > 10 ;
```

SQL

die standardisierte

- Datendefinitions-
- Datenmanipulations- und
- Anfrage-**Sprache**

für **relationale** Datenbanksysteme

- entstanden
  System/R (IBM)
  Entwicklung 1973 – 1979
  ↷ SEQUEL
  Structured **E**nglish **QUE**ry **L**anguage

- **SQL** - **S**trctured **Q**uery **L**anguage

1986	SQL1 bzw. SQL86	grundsätzliche Festlegung Standard bei ANSI und ISO
1989	SQL89	Erweiterungen
1992	SQL2 bzw. SQL92	erhebliche Erweiterungen Entry-SQL2 Intermediate SQL2 Full SQL2
1999	SQL3 bzw. SQL99	u.a. objektrelationale Erweiterungen

Auswahl von SQL-Anweisungen

Syntax-Beschreibung mit folgenden Festlegungen:

fett, groß	Schlüsselwörter der Sprache
normal	nutzerdefinierte Angaben
|	Alternativtrennung
[. . .]	optionale Angaben (muss nicht sein, **kann** aber)
{ . . . | . . . }	Alternativen (**eine muss** ausgewählt werden)

Literatur:

- Chris J. Date, Hugh Darwen
  SQL – Der Standard.
  SQL/92 mit den Erweiterungen CLI und PSM
  Addison-Wesley, München
  Deutsche Ausgabe des amerikanischen Klassikers

- C. J. Date, Hugh Darwen
  A Guide to the SQL Standard
  Addison Wesley Longman Publishing Co; 4th Ed., 1997

Professor

PNr	PName	Gehalt	Fachgebiet	Lehrsoll	Tel	Raum
101	Albert Einstein	9000	Theoret. Physik	6	4512	413
102	David Hilbert	8500	Algebra	7	2345	212
103	Henri Poincare	7500	Analysis	8	2346	214
104	John v. Neumann	8000	Informatik	7	5432	510

Vorlesung

VName	Studiengang	SWS	Semester	PNr
Algebra	Mathematik	4	WS	102
Geometrie	Mathematik	4	SS	102
Partielle DGL	Physik	3	WS	103
Rechnerarchitektur	Informatik	3	WS	104
Differentialgleichungen	Mathematik	3	SS	103
Automatentheorie	Informatik	2	SS	104
Relativitätstheorie	Physik	4	WS	101
Logik	Mathematik	2	WS	104
Quantentheorie	Physik	4	SS	101
Radioaktivität	Chemie	5	SS	105

Student

StudNr	SName	Studiengang
2001	Fritz Müller	Informatik
2002	Max Schneider	Mathematik
2003	Hans Schulz	Physik
2004	Otto Meier	Mathematik

Einschreibung

StudNr	VName	Semester	StudJahr
2001	Automatentheorie	SS	99
2001	Logik	WS	99
2001	Algebra	WS	00
2003	Partielle DGL	WS	99
2003	Relativitätstheorie	WS	00
2003	Quantentheorie	SS	99
2002	Algebra	WS	00
2002	Relativitätstheorie	WS	00
2005	Radioaktivität	SS	00

Schema-Definition

- **CREATE TABLE tabname (**
  attribut datentyp [**zusätze**]
  [**, attribut datentyp** [**zusätze**] **, ...**]
) ;

  tabname
  nutzerdefinierter Tabellenname
  attribut
  nutzerdefinierte Attributbezeichnug
  datentyp
  real, integer, character, date, ...
  zusätze:
  NOT NULL
  keine NULL-Einträge zulässig
  PRIMARY KEY
  erklärter Primärschlüssel
  UNIQUE
  eindeutige Werte gefordert, d.h. keine Wiederholungen (z.B. für Schlüsselkandidaten)
  CHECK (attributbedingung)
  einfache Integritätsbedingung für Attribut
  FOREIGN KEY
  REFERENCES tabname (attribut)
  Fremdschlüsselbedingung

- **ALTER TABLE tabname**
  ADD attribut datentyp ;

- **DROP TABLE tabname ;**

Beispiele

- CREATE TABLE Professor
  (PNr INTEGER PRIMARY KEY ,
  PName VARCHAR(20) NOT NULL ,
  Gehalt DECIMAL(7,2) ,
  Fachgebiet VARCHAR(20) ,
  Lehrsoll INTEGER CHECK (Lehrsoll >= 0) ,
  Tel INTEGER UNIQUE) ;

- ALTER TABLE Professor
  ADD Raum INTEGER ;

- CREATE TABLE Vorlesung
  (Vname VARCHAR(25) ,
  Studiengang VARCHAR(20) ,
  SWS INTEGER ,
  Semester CHAR(2)
  CHECK (Semester IN ('WS','SS')) ,
  PNr INTEGER ,
  PRIMARY KEY (Vname , Studiengang) ,
  FOREIGN KEY (PNr) REFERENCES Professor) ;

- DROP TABLE Professor ;

  ⇒ Tabelle **mit** Schema gelöscht **!!!**

Datenmanipulation

- **INSERT INTO tabname [(attribut [, attribut , …])]**
  VALUES (const1 , const2 , …) ;

- **INSERT INTO tabname [(attribut [, attribut , …])]**
  SELECT …
  FROM …
  WHERE … ;

- **UPDATE tabname**
  SET attribut = ausdruck [, attribut = ausdruck , …]
  [WHERE bedingung] ;

- **DELETE FROM tabname**
  [WHERE bedingung] ;

Beispiele

- INSERT INTO Professor
  VALUES (101 , ’Albert Einstein’ , 9000 ,
  ’Theoret. Physik’ , 6 , 4512 , 413) ;

- CREATE TABLE Vorlesungsverzeichnis
  (…) ;

  INSERT INTO Vorlesungsverzeichnis
  SELECT VName , PName
  FROM Vorlesung , Professor
  WHERE Vorlesung.PNr = Professor.PNr
  AND Semester = ’WS’ ;

- UPDATE Professor
  SET Gehalt = Gehalt + Gehalt ∗ 0.1 ;

- UPDATE Professor
  SET Gehalt = Gehalt + 100
  WHERE PName = ’Marie Curie’ ;

- DELETE FROM Student ;

- DELETE FROM Student
  WHERE SName = ’Otto Meier’ ;

Anfragen

Die zentrale SQL-Anweisung für Anfragen ist die
SFW-Formel (SELECT ... FROM ... WHERE …)

- **SELECT** [**DISTINCT**]{**attribut** | **ausdruck** | **aggregat**}
  FROM tabname [**,** **tabname** **,** **…**]
  [**WHERE bedingung1**
    [{ **AND** | **OR** } **bedingung2**
      [**,** { **AND** | **OR** } **bedingung3** **,** **…**]]]
  [**GROUP BY attribut**
    [**HAVING bedingung4**]]
  [**ORDER BY attribut** { **DESC** | **ASC** }] **;**

Bemerkung:
In der Bedingung der WHERE-Klausel kann wiederum eine SELECT-Anweisung enthalten sein.
⇒ Unteranfrage
⇒ geschachtelte Anfragen

Abarbeitungsreihenfolge und Erläuterungen

FROM

- spezifiziert die benötigten (Basis-) Relationen oder Sichten
- Verknüpfung dieser über ein **kartesisches Produkt**

WHERE

- spezifiziert **Selektion**sbedingungen und/oder **Verbund**bedingungen
- **Schachtelung** von Anfragen durch neuen SFW-Block
- neuer SFW-Block ist Teil der Bedingung

GROUP BY

- **gruppiert** die bisherigen Ergebnis-Tupel nach den angegebenen Attributen
- alle in der SELECT-Klausel genannten Attribute (außer den aggregierten) müssen auch in der GROUP BY – Klausel aufgeführt werden

HAVING

- spezifiziert **Selektion**sbedingungen **für** die **Gruppen**
- nur mit GROUP BY verwendbar !

SELECT

- **projiziert** die Tupel auf die genannten Attribute
- legt damit Schema der Ergebnis-Relation fest
- mit DISTINCT können Duplikate entfernt werden

ORDER BY

- **sortiert** die Ergebnis-Tupel nach den gen. Attributen
- Attribute müssen auch schon in SELECT-Klausel sein
- **DESC** – abfallend, **ASC** – aufsteigend (voreingestellt)

Weitere Konzepte zur Verwendung im SFW-Block

IN

- Mengenmitgliedschaft
- Menge oft durch Unterabfrage erzeugt
- Verwendung in der WHERE-Klausel
- Negation: **NOT IN**

attribut **BETWEEN** const1 **AND** const2

- Bereichs-Selektion
- Verwendung in der WHERE-Klausel
- Bewirkt: const1 ≤ attribut ≤ const2

attribut **LIKE** spezialkonstante

- Ungewissheits-Selektion
- Verwendung in der WHERE-Klausel
- die Spezialkonstante enthält Sondersymbole % oder _
- % steht für kein oder beliebig viele Zeichen
- _ steht für genau ein Zeichen

attribut **IS NULL**

- NULL-Selektion
- Verwendung in der WHERE-Klausel

ALL, ANY, SOME, EXISTS

- Quantoren
- für quantifizierte Bedingungen
- Verwendung in der WHERE-Klausel
- im Zusammenhang mit Unterabfragen

AND, OR, NOT

- Konnektoren
- zum Verknüpfen von Einzelbedingungen bzw. zur Negation
- Verwendung in der WHERE-Klausel

COUNT, SUM, AVG, MIN, MAX

- Aggregatfunktionen
- arbeiten über Tupelmengen
- COUNT - Anzahl
  SUM - Summe
  AVG - Mittelwert
  MIN - Minimum
  MAX - Maximum
- Argumente: Attribut, skalarer Ausdruck, * (bei COUNT)

SFW-block1 **UNION** SFW-block2

- Mengenoperation der Vereinigung
- vereinigt zwei Tupelmengen über dem gleichen Schema
- ausführbar, wenn positionsweise korrespondierende Attribute kompatible Wertebereiche haben
- einzigste Mengenoperation in SQL-89
- **Achtung !** Ergebnis nicht weiter in Anfragen einsetzbar

Weitere Mengenoperationen in SQL-92:

INTERSECT - Durchschnitt
EXCEPT - Differenz

Weitere explizite Operatoren in SQL-92

CROSS JOIN
JOIN ... ON
JOIN … USING
NATURAL JOIN
OUTER JOIN
LEFT OUTER JOIN
RIGHT OUTER JOIN
NATURAL LEFT OUTER JOIN
NATURAL RIGHT OUTER JOIN

Einfache Anfragen

- Studenten des Studienganges Chemie

```
SELECT  StudNr,  SName
  FROM  Student
WHERE  Studiengang = ’Chemie’ ;
```

- Alle Angaben zu allen Studenten

```
SELECT  *
  FROM  Student ;
```

- Wie sähe die Belastung der Professoren bei einer angedachten Erhöhung des Lehrsolls um 2 SWS für die Professoren der Informatik aus ?

```
SELECT  PName,  Lehrsoll + 2  SollNeu
  FROM  Professor
WHERE  Fachgebiet = ’Informatik’ ;
```

PName	*SollNeu*
John v. Neumann	9

Anfrage mit Sortierung

- Sortierung der Einschreibungen nach Studienjahren

```
SELECT  StudNr,  VName,  StudJahr
  FROM  Einschreibung
ORDER  BY  StudJahr  DESC ;
```

Anfragen mit Duplikateliminierung

- Für welche Jahre sind Einschreibungen erfasst ?

```
SELECT  DISTINCT  StudJahr
  FROM  Einschreibung ;
```

- Für welche Vorlesungen gibt es Einschreibungen ?

```
SELECT  DISTINCT  VName
  FROM  Einschreibung ;
```

- In welchen Studiengängen gibt es immatrikulierte Studenten ?

```
SELECT  DISTINCT  Studiengang
  FROM  Student ;
```

Anfragen über mehrere Relationen

- Zusammenstellung aller Vorlesungen, die Hans Schulz bisher gehört hat (anhand der Einschreibungen)

```
SELECT  VName
  FROM  Einschreibung ,  Student
WHERE  Einschreibung . StudNr = Student . StudNr
    AND  SName = ' Hans Schulz ' ;
```

Relationenalgebraischer Ausdruck:

π … (σ … (Einschreibung $\times$ Student))

- Liste der Namen mit Telefon und Raum von den Professoren, deren Vorlesungen Fritz Müller im WS 2000 besuchte

```
SELECT  PName ,  Tel ,  Raum
  FROM  Professor  p ,  Vorlesung  v ,
        Student  s ,  Einschreibung  e
WHERE  SName = ' Fritz Müller '
    AND  s . StudNr = e . StudNr
    AND  Semester = 'WS '
    AND  StudJahr = 00
    AND  e . Vname = v . Vname
    AND  v . PNr = p . PNr ;
```

Mengenoperationen (mit geschachtelten Anfragen)

- Namensliste aller Uni-Angehörigen (Professoren und Studenten)

```
( SELECT  PName  Name
     FROM  Professor  )
UNION
( SELECT  SName  Name
     FROM  Student  )
```

Nicht weiter verwendbar **!**

- Professor mit dem größten Lehrsoll

```
SELECT  PName
  FROM  Professor
WHERE  Lehrsoll > = ALL ( SELECT  Lehrsoll
                            FROM  Professor ) ;
```

Achtung !

kein vollwertiger Allquantor ;
nur Vergleich mit ALL

- Professoren, die im Moment keine Vorlesung anbieten

```
SELECT  PName
  FROM  Professor  p
WHERE  NOT  EXISTS  ( SELECT  *
                        FROM  Vorlesung  v
                       WHERE  v . PNr = p . PNr ) ;
```

Anders interpretiert:
Professoren, für die kein Eintrag unter ihrer Personalnummer im Vorlesungsverzeichnis existiert

- Professoren, die im Moment keine Vorlesung anbieten

```
SELECT  PName
  FROM  Professor
WHERE  PNr  NOT  IN  ( SELECT  PNr
                         FROM  Vorlesung  ) ;
```

Aggregatfunktionen und Gruppierung

- Durchschnittsgehalt der Professoren

```
SELECT  AVG ( Gehalt )
  FROM  Professor ;
```

- Summe der SWS der angebotenen Vorlesungen pro Vorlesenden

```
SELECT  PNr ,  SUM ( SWS )
  FROM  Vorlesung
GROUP  BY  PNr ;
```

- Professoren, deren Summe der SWS der angebotenen Vorlesungen unter ihrem Lehrsoll liegt

```
SELECT  PNr ,  PName ,  SUM ( SWS )
  FROM  Professor ,  Vorlesung
WHERE  Professor . PNr = Vorlesung . PNr
GROUP  BY  PNr ,  PName
   HAVING  SUM ( SWS ) < Lehrsoll ;
```

Bemerkungen:
Für jede Gruppe entsteht **ein** Ergebnistupel !
Alle in der SELECT - Klausel genannten Attribute – außer den aggregierten –
müssen auch in der GROUP BY – Klausel aufgeführt werden.
Damit wird garantiert, dass sich das Attribut nicht innerhalb der Gruppe ändert.

Geschachtelte Anfragen

- Vorlesung, die vom Umfang größer ist als das minimale Lehrsoll eines Professors

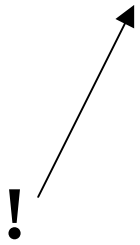
```
SELECT  VName
  FROM  Vorlesung
WHERE  SWS > ( SELECT  MIN ( Lehrsoll )
                 FROM  Professor ) ;
```

- Größter Vorlesungsumfang

```
SELECT  MAX ( SWS )
  FROM  Vorlesung ;
```

- Vorlesung, die den größten Umfang hat

```
SELECT  VName
  FROM  Vorlesung
WHERE  SWS = ( SELECT  MAX ( SWS )
                 FROM  Vorlesung ) ;
```



Korrelierte und unkorrellierte Unteranfrage

- Welche Vorlesung ist umfangreicher als ein Lehrsoll ?

```
SELECT  VName
  FROM  Vorlesung  v
WHERE  EXISTS  ( SELECT  *
                   FROM  Professor  p
                  WHERE  p . Lehrsoll <= v . SWS );
```

Bemerkung:
Unteranfrage wird für jedes Tupel der äußeren Anfrage wiederholt vollständig ausgeführt.

- Welche Vorlesung ist umfangreicher als das minimale Lehrsoll ?

```
SELECT  VName
  FROM  Vorlesung
WHERE  SWS > = ( SELECT  MIN ( Lehrsoll )
                   FROM  Professor ) ;
```

Bemerkung:
Die Unteranfrage wird einmalig ausgewertet im Gegensatz zur korrellierten Unteranfrage.

Entschachtelung von Unteranfragen

- Welcher Professor liest zwei oder mehr verschiedene Vorlesungen ?

```
SELECT  x . PNr
  FROM  Vorlesung  x
WHERE  x . PNr  IN
                ( SELECT  y . PNr
                    FROM  Vorlesung  y
                   WHERE  y . VName < > x . VName ) ;
```

- *Entschachtelt*

```
SELECT  x . PNr
  FROM  Vorlesung  x ,  Vorlesung  y
WHERE  x . PNr = y . PNr
    AND  x . VName < > y . Vname ;
```

- *mit Aggregatfunktion (ohne Tupelvariable)*

```
SELECT  PNr
  FROM  Vorlesung
GROUP  BY  PNr
HAVING  COUNT ( * ) > 1 ;
```

Spezielle Sprachkonstrukte

- Vorlesungen mit einem Umfang zwischen 2 und 4 SWS

```
SELECT  *
  FROM  Vorlesung
WHERE  SWS > 2  AND  SWS < = 4 ;
```

- Professoren mit einem Gehalt zwischen 7000 und 9000

```
SELECT  *
  FROM  Professor
WHERE  Gehalt  BETWEEN  7000  AND  9000 ;
```

- Professoren in den Räumen 212, 214, 216 bzw. 218

```
SELECT  PName ,  Tel
  FROM  Professor
WHERE  Raum  IN  ( 212, 214, 216, 218 ) ;
```

- Theorie-Vorlesungen
  (Vorlesungen, die im Namen mit ...theorie enden)

```
SELECT  *
  FROM  Vorlesung
WHERE  VName  LIKE  ‘ % theorie ‘  ;
```

NULL-Werte

- unbekannter Wert für ein Merkmal (Attribut)
  1. Wert existiert, ist nur unbekannt
     z. B. PName (Name des Professors)
  2. unbekannt, ob dieser Wert überhaupt existiert
     z. B. Tel (Telefonnummer des Professors)

- NULL-Werte können aber auch im Ergebnis von Anfragen entstehen

- manchmal abweichende Anfrageergebnisse, wenn NULL-Werte in Relation vorkommen

```
SELECT  AVG ( Gehalt )
  FROM  Professor ;

SELECT  SUM ( Gehalt ) / COUNT ( * )
  FROM  Professor ;
```

  Bemerkung:
  Die NULL-Werte werden durch die Aggregatfunktionen (hier: AVG, SUM) nicht berücksichtigt.

- SQL hat dreiwertige Logik
  true, false, unknown

Regeln für den Umgang mit NULL-Werten

1. Arithmetische Ausdrücke

   Ein Operand NULL ⇒ Ergebnis NULL
   z. B. NULL + 17 ⇒ NULL

2. Vergleichsoperatoren

   Mindestens ein Argument NULL ⇒
   Auswertung zu **unknown**
   z. B. 17 > NULL ⇒ **unknown**

3. WHERE-Bedingung

   Es werden nur Tupel weitergereicht, für die die Bedingung **true** ist.

   Auch Auswertungen zu **unknown** kommen nicht ins Ergebnis.

4. Gruppierung

   NULL ist ein eigenständiger Wert;
   wird in eine eigene Gruppe eingeordnet

... und weitere Regeln

Sichten

- **CREATE VIEW sichtname [(liste von attributen)]**
  AS SELECT …
  FROM …
  WHERE … ;

Bemerkungen:

- erzeugt keine neue (Basis-)Relation – nur **virtuelle Relation**
- Änderbarkeit von Sichten sehr eingeschränkt
  ⇒ PROBLEME **!**
- Nutzung für

  Datenschutz
  (auch mit Zugriffsrechten verknüpft)

  Vereinfachung von häufigen Anfragen

  Datenunabhängigkeit

Beispiele

- **Datenschutz**

  Sicht auf Professoren ohne Gehaltsangaben u.a.

```
CREATE  VIEW  ProfSicht
AS  SELECT  Pname ,  Fachgebiet ,  Tel ,  Raum
        FROM  Professor ;
```

- **Vereinfachung** von häufigen Anfragen

  Zusammenstellung von Prüfungslisten entsprechend der Einschreibungen mit Vorlesung und Professor

```
CREATE  VIEW  Prüfungslisten
                    ( StudNr ,  Name ,  Prof ,  Fach )
AS  SELECT  s . StudNr ,  s . SName ,
                p . PName ,  v . VName
       FROM  Student  s ,  Einschreibung  e ,
             Vorlesung  v ,  Professor  p
     WHERE  p . PNr = v . PNr
         AND  v . VName = e . VName
         AND  e . StudNr = s . StudNr ;
```

  Ausdruck der Liste für Professor Hilbert zum Fach Algebra

```
SELECT  StudNr ,  Name
  FROM  Prüfungslisten
WHERE  Prof = ' David Hilbert '
    AND  Fach = ' Algebra ' ;
```

Vergabe von Zugriffsrechten

GRANT { liste_der_rechte | ALL PRIVILEGS}
ON [TABLE] tabname
TO { liste_der_nutzer | PUBLIC }
[WITH GRANT OPTION] ;

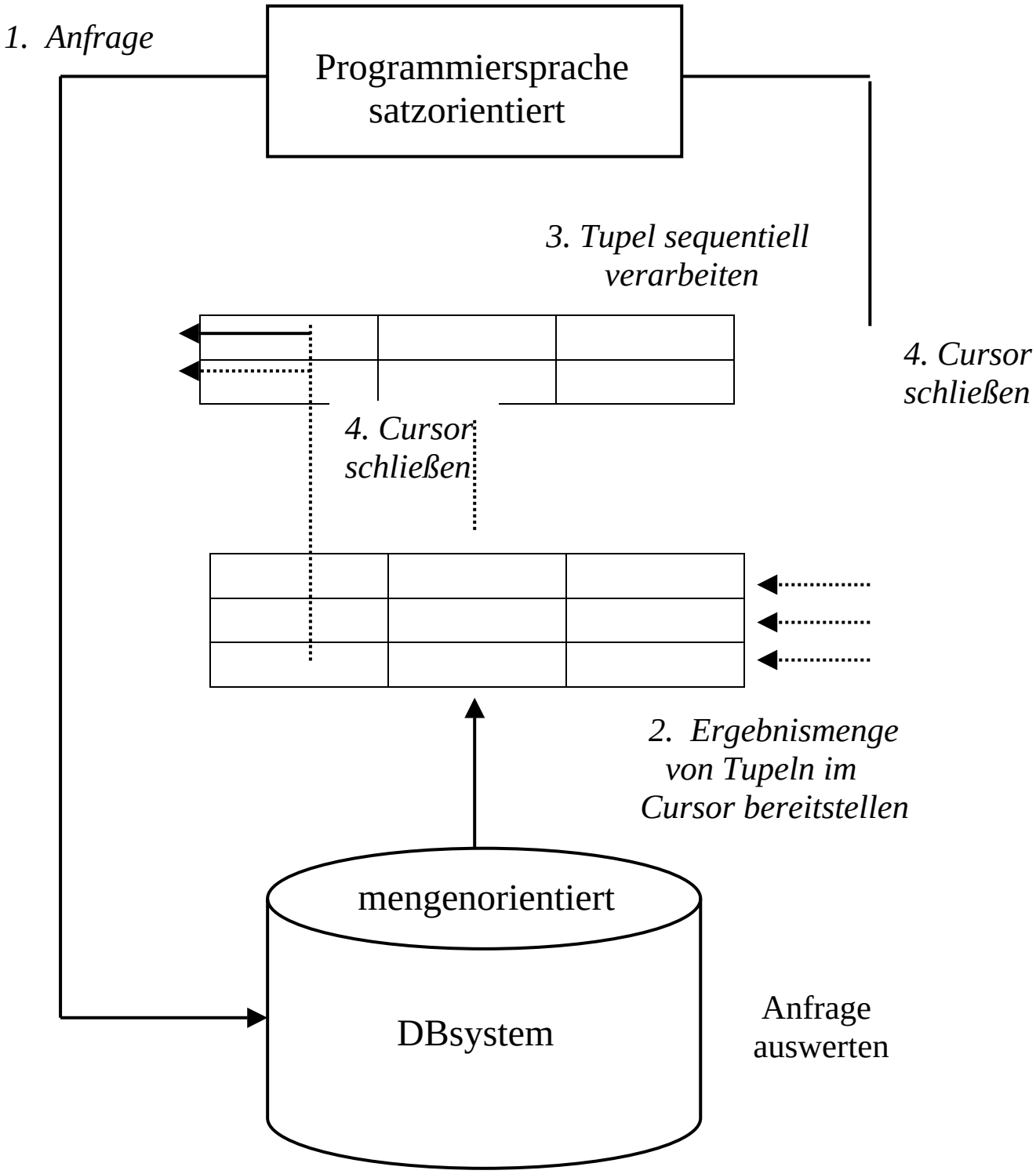
liste_der_rechte : =
{ SELECT | INSERT | DELETE | UPDATE
[(attribut [, attribut] …)] }

Entzug von Rechten

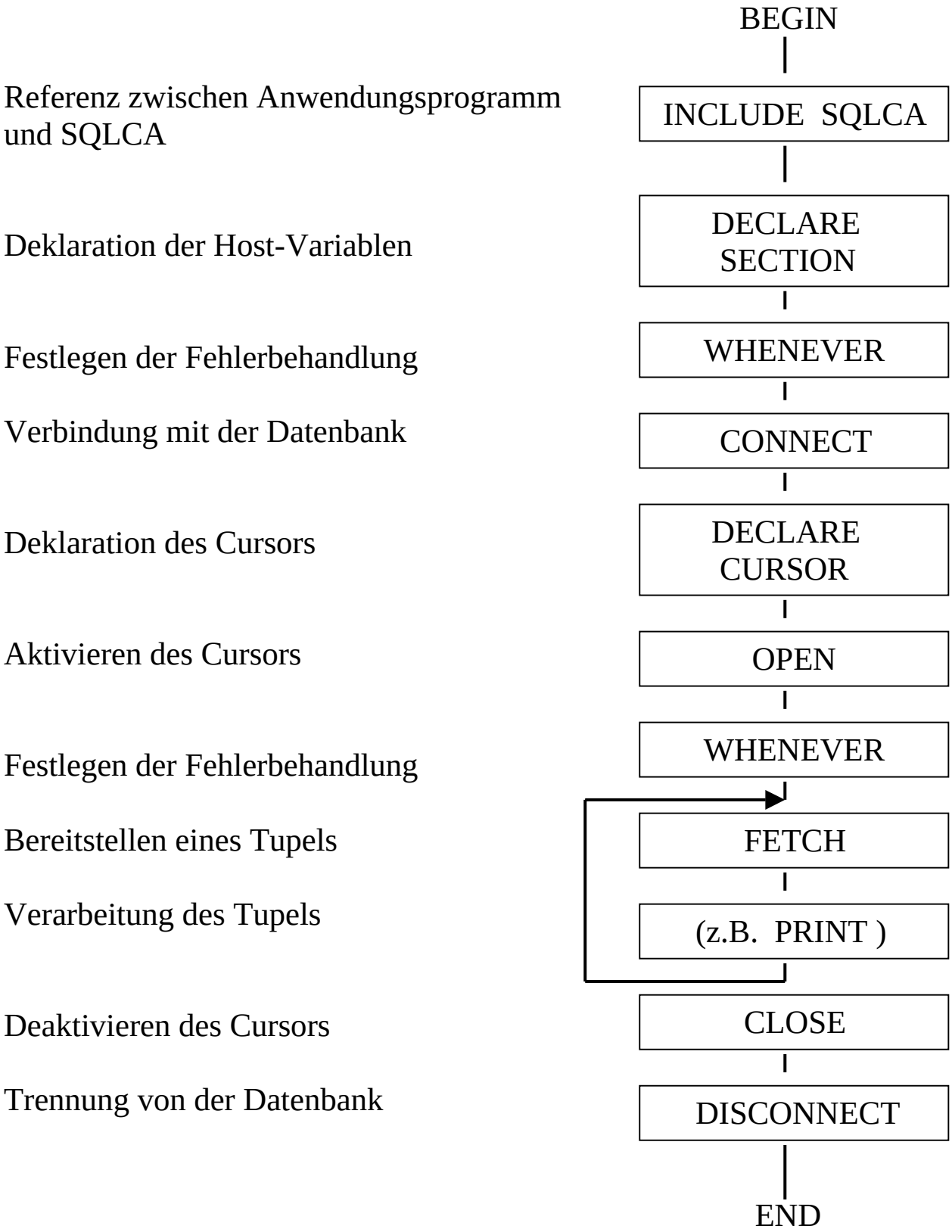
REVOKE recht
ON [TABLE] tabname
FROM nutzer ;

ESQL (Embedded SQL)

für Anfragen in Anwendungsprogrammen



Programmstruktur für die Einbettung von SQL in ein Host-Programm



CURSOR – **CUR**rent **S**et **O**f **R**ecords

Beispiel (Programm in C)

```
# include  <stdio.h>

main ( )
{
  exec  sql  include  sqlca ;
  exec  sql  begin  declare  section ;
     int  matrnr ;
     char  name [25] ;
  exec  sql  end  declare  section :
  exec  sql  whenever  sqlerror  stop ;
  exec  sql  connect  StudentenDB ;
  exec  sql  declare  Informatiker  cursor  for
                SELECT  StudNr ,  Sname
                  FROM  Student
                WHERE  Studiengang  LIKE  ' % informatik ' ;
  exec  sql  open  Informatiker ;
  exec  sql  whenever  not  found  goto  closeInf ;
     loop  indefinitly
        exec  sql  fetch  Informatiker  into  : matrnr ,  : name ;
        printf  ( " \n %d %s " , matrnr , name ) ;
     end  loop ;
     closeInf ;
  exec  sql  close Informatiker ;
  exec  sql  disconnect ;
}
```

Weitere relationale Anfragesprachen

QUEL

QBE

4. Formaler (relationaler) DBentwurf

Auch schon beim Datenbank-Entwurf gibt es weitere Möglichkeiten Semantik zu „erfassen“.

Ziele des Entwurfs

- Semantik der Anwendungswelt korrekt widerspiegeln
- Verringerung von Redundanz
  $\Rightarrow$ **Fehlerquelle !**
- Verringerung von NULL-Werten
- Vermeidung widersprüchlicher Einträge

Ziele (aus der Theorie relationaler DB)

1. Abhängigkeitstreue
2. 3NF
3. minimales DBschema
4. Verbundtreue

Abteilungspersonal

PName	PNr	GebDat	ANr	AName	Gebäude
Meier, R.	1234	09-01-1955	5	Forschung	B1
Schulz, B.	1235	08-12-1965	5	Forschung	B1
Koch, A.	1237	19-07-1979	4	Verwaltung	B1
Walter, S.	1243	15-09-1962	3	Technik	A2
Scholz, G.	1256	31-07-1972	4	Verwaltung	B1

1. Einfüge-Anomalie

Einfügen eines neuen Mitarbeiters in Abt.5
⇒ Angaben zur Abt.5 **korrekt** wiederholen

Einfügen einer neuen Abteilung, die noch kein Personal hat
⇒ NULL-Werte (auch für PNr)

2. Löschanomalie

Kündigung des Mitarbeiters Walter, S.
⇒ Angaben über Technik-Abt. weg

3. Änderungsanomalie

Umzug der Abt.5 von B1 nach B2
⇒ Änderung **aller** Mitarbeiter-Tupel dieser Abt.

4. Redundanz

Bestellannahme

EingNr	ArtikelNr	Artikel	Menge	KundenNr	Name
1001	M5643, K1234	Bluse, Toaster	1, 1	3124	Fischer
1002	K1234	Toaster	2	5761	Abel
1003	M0038, M5666	Socken, Bluse	3, 2	4389	Meier
1004	S8973	Schrank	1	3124	Fischer
1005	K1234	Toaster	1	8913	Meier
1006	M0038, K1234	Socken, Toaster	4, 1	3124	Fischer

Erste Normalform (1NF)

Eine Relation befindet sich in **1NF**, wenn alle Attributwerte **atomar** sind im Sinne der Anwendung
(keine Mengen, Aufzählungen usw.).

Bemerkung:
Das ist eigentlich eine Grundannahme des relationalen Datenmodells.

Ist das nicht der Fall, spricht man von einer **unnormalisierten** Relation.

Bestellannahme (1NF)

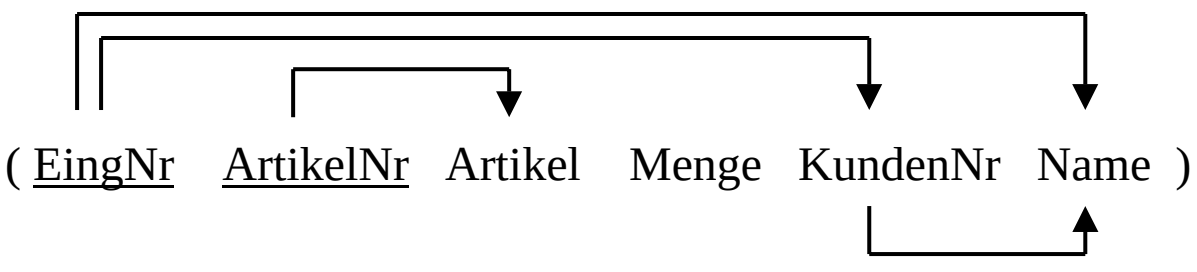
EingNr	ArtikelNr	Artikel	Menge	KundenNr	Name
1001	M5643	Bluse	1	3124	Fischer
1001	K1234	Toaster	1	3124	Fischer
1002	K1234	Toaster	2	5761	Abel
1003	M0038	Socken	3	4389	Meier
1003	M5666	Bluse	2	4389	Meier
1004	S8973	Schrank	1	3124	Fischer
1005	K1234	Toaster	1	8913	Meier
1006	M0038	Socken	4	3124	Fischer
1006	K1234	Toaster	1	3124	Fischer

→ noch mehr Redundanz

→ Erweiterung des Schlüssels

Zweite Normalform (2NF)

Eine Relation befindet sich in **2NF**, wenn sie in 1NF ist und wenn jedes Nichtschlüsselattribut vom Schlüssel **voll** funktional abhängig ist (nicht nur partiell).



Funktionale Abhängigkeit

Ein Attribut B ist funktional abhängig von einem Attribut A, wenn zu jedem Wert von A genau ein Wert von B existiert,
d.h.
der Wert von B wird eindeutig durch A bestimmt
oder
für zwei Tupel r und s mit gleichen A-Werten müssen auch die B-Werte übereinstimmen, d.h.

$$r(A) = s(A) \quad \Rightarrow \quad r(B) = s(B)$$

Bezeichnung: $\mathbf{A} \longrightarrow \mathbf{B}$

- Jede funktionale Abhängigkeit ist eine semantische Integritätsbedingung.

- Jeder Schlüssel hat die Eigenschaft, dass die Nichtschlüsselattribute vom Schlüssel funktional abhängig sind.

- **volle funktionale Abhängigkeit**
  wenn ein zusammengesetzter Schlüssel allein in seiner Gesamtheit die übrigen Nichtschlüsselattribute eindeutig bestimmt

- **partielle funktionale Abhängigkeit**
  wenn Nichtschlüsselattribute auch schon von Teilen eines zusammengesetzten Schlüssels funktional abhängig sind

- **transitive funktionale Abhängigkeit**
  Ein Nichtschlüsselattribut ist transitiv vom Schlüssel abhängig, wenn es auch noch eine funktionale Abhängigkeit von einem anderen Nichtschlüsselattribut gibt.

  $A \longrightarrow B, \quad B \longrightarrow C$

  $A \not\longleftarrow B$

Normalisierung durch Dekomposition

Relation r (R) , K – Schlüssel , Attribute A , B $\subset$ R

A $\longrightarrow$ B verstößt gegen Forderung der Normalform

1. neue Relation für Attribute A, B der „störenden“ Abhängigkeit

2. determinierendes Attribut A wird dort Schlüssel

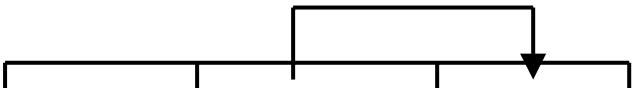
3. determinierendes Attribut A verbleibt auch in der ursprünglichen Relation
   (als Fremdschlüssel und damit Grundlage für den natürlichen Verbund mit der neuen Relation)

Ergebnis:

R1 = R – B	Schlüssel: K
R2 = A B	Schlüssel: A

$$\mathbf{r} = \pi_{R1}(\mathbf{r}) \;|><|\; \pi_{R2}(\mathbf{r})$$

Eingänge (2NF)



EingNr	KundenNr	Name
1001	3124	Fischer
1002	5761	Abel
1003	4389	Meier
1004	3124	Fischer
1005	8913	Meier
1006	3124	Fischer

Artikelliste (2NF)

ArtikelNr	Artikel
M5643	Bluse
K1234	Toaster
M0038	Socken
M5666	Bluse
S8973	Schrank

Bestellannahme (2NF)

EingNr	ArtikelNr	Menge
1001	M5643	1
1001	K1234	1
1002	K1234	2
1003	M0038	3
1003	M5666	2
1004	S8973	1
1005	K1234	1
1006	M0038	4
1006	K1234	1

Dritte Normalform (3NF)

Eine Relation befindet sich in **3NF**, wenn sie in 1NF ist und jedes Nichtschlüsselattribut nicht transitiv abhängig vom Schlüssel ist.

Eingänge (3NF)

EingNr	KundenNr
1001	3124
1002	5761
1003	4389
1004	3124
1005	8913
1006	3124

Kunden (3NF)

KundenNr	Name
3124	Fischer
5761	Abel
4389	Meier
8913	Meier

Artikelliste (3NF)

ArtikelNr	Artikel
M5643	Bluse
K1234	Toaster
M0038	Socken
M5666	Bluse
S8973	Schrank

Bestellannahme (3NF)

EingNr	ArtikelNr	Menge
1001	M5643	1
1001	K1234	1
1002	K1234	2
1003	M0038	3
1003	M5666	2
1004	S8973	1
1005	K1234	1
1006	M0038	4
1006	K1234	1

Das ist die relationale Datenbank in 3. Normalform.

Aufgabe

Überlegen Sie sich die Vor- und Nachteile der Normalisierung .

5. Datenbanksysteme

Warum Datenbanksysteme ?

1. Herauslösen der Datenverwaltungsfunktionen aus den Anwendungsprogrammen

2. Mehr Funktionalität und höhere Abstraktion für Nutzer als einfache Dateiverwaltungssysteme (siehe Einführung)

3. Einmal angelegte (zentrale) und immer aktuelle Datensammlung für viele Nutzergruppen mit unterschiedlichen Interessen

Datenbanksysteme

sind rechnergestützte Hilfsmittel, um Daten einer Anwendungs“welt“

- aufzubewahren,
- fortzuschreiben,
- verschiedenen Anwendern in geeigneter Weise – auch gleichzeitig – zugänglich zu machen

DBS = DBMS + (n) DB

DBS database system
DBMS database management system
DB database

DBMS

spezielles Softwaresystem zur

1. Beschreibung
2. Aufbau
3. Handhabung

} **Metadaten** entstehen

von DB für verschiedene Anwendungen

1. Logische Beschreibung der DB entsprechend dem Datenmodell
   (Datenmengen, Beziehungen, Strukturen, Restriktionen)

2. Physische Speicherung der Daten auf einem externen Speichermedium, das durch das DBMS verwaltet wird

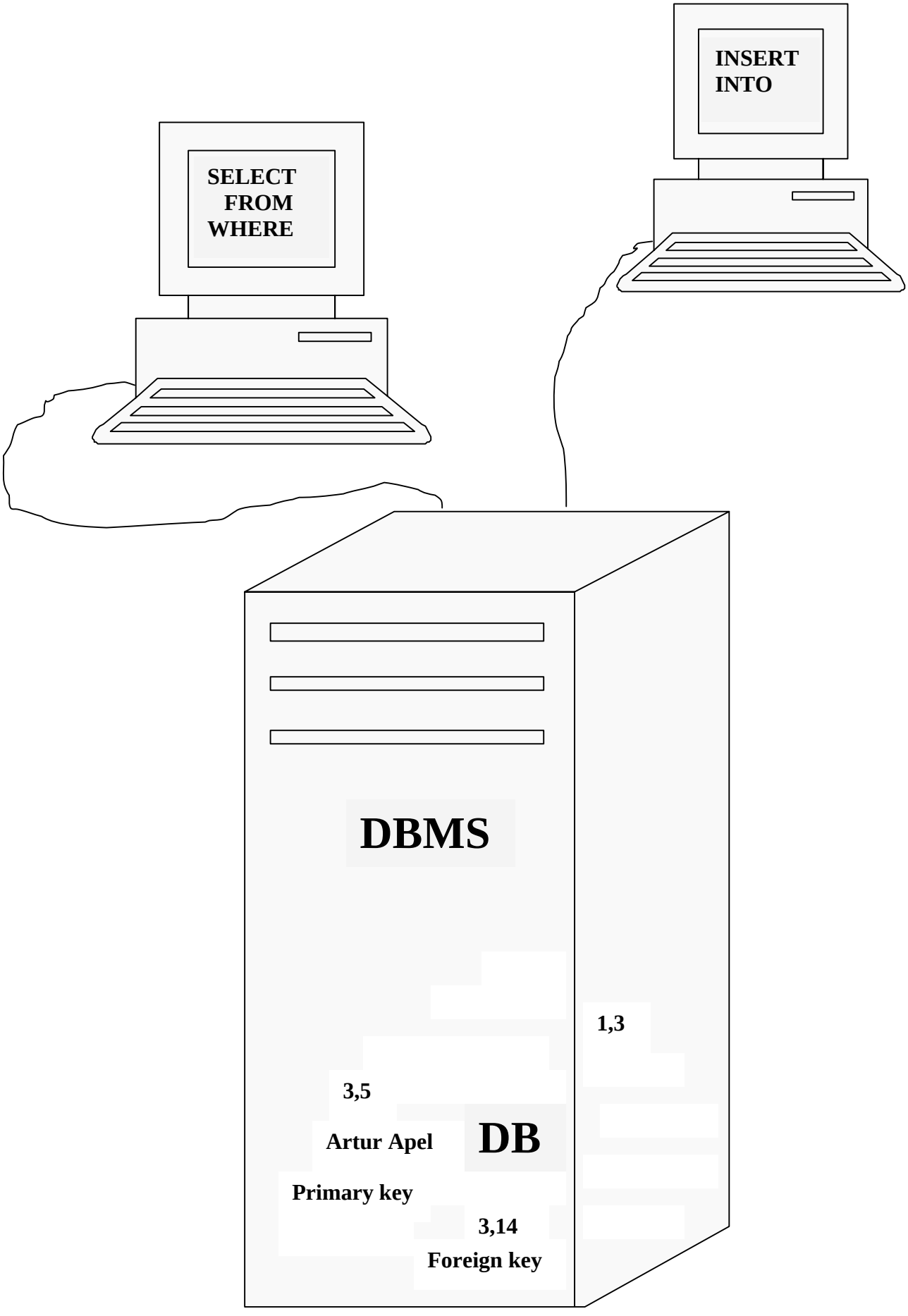
3.
   - Anfragen zum Auffinden bestimmter Daten
   - Änderungen in der DB entsprechend Veränderungen in der realen Welt
   - Zusammenstellungen und Verknüpfungen von Daten

UND

Verwaltung der Metadaten

UND

Organisation des „reibungslosen“ Ablaufs


INSERT
INTO
SELECT
FROM
WHERE
DBMS
1,3
3,5
Artur Apel
DB
Primary key
3,14
Foreign key

Datenbanksysteme –
Ansatz und Architektur

Grundlegende Charakteristik des Datenbank-Ansatzes:

realisiert ein gewisses Maß an **Datenabstraktion**
durch Verbergen von Details der Datenabspeicherung,
die für die meisten Anwender nicht notwendig sind.

Datenmodell DM

- Menge von Konzepten zur Beschreibung der Struktur (Datenmengen, Beziehungen, Restriktionen) der DB

- unterschiedliche Abstraktionsniveaus

  1. **Konzeptuelles DM** (high-level)
     Erfassen der Semantik der Anwendungs“welt“,
     z.B. ERM

  2. **Logisches DM** (implementation DM)
     z.B.
     relationales DM
     Netzwerk-DM
     hierarchisches DM
     objektorientiertes DM

  3. **Physisches DM** (low-level)
     beschreibt Form der Abspeicherung der Daten
     (Satz-Formate, Speicherungsform, Zugriffspfade, ...)

DBschema

- Beschreibung der DB entsprechend dem logischen DM
- abgelegt im DBMS-Katalog
- keine häufigen Änderungen zu erwarten

  Schema-Konstrukt ⇒ **Intension**

  z.B. Relationenschema

DBzustand

- Daten in der DB zu einem bestimmten Zeitpunkt (Gegensatz dazu: *temporäre Datenbanken*)

⇒

- jede Änderung führt zu neuem DBzustand
- jeder Schemakonstrukt hat aktuelle Menge von Instanzen

  Menge von Instanzen ⇒ **Extension**

  z.B. aktuelle Menge von Tupeln

3-Schichten-Architektur

Grundgedanke: Trennung der Nutzer-Anwendungen von der physischen DB

1. Externe Ebene $\Rightarrow$ externes Schema (Sichten)

- beschreibt Teile der DB für jede Nutzergruppe (andere Teile verborgen)
- nutzt ein konzeptuelles oder logisches DM
- Menge von Teil-Schemen

2. Konzeptuelle Ebene $\Rightarrow$ konzeptuelles Schema

- beschreibt die Struktur der gesamten DB für alle Nutzergruppen
  (Objekte, Datentypen, Beziehungen, Restriktionen, Nutzeroperationen, ...)
- nutzt ein konzeptuelles und / oder logisches DM
- verbirgt Details der Speicherstrukturen

3. Interne Ebene $\Rightarrow$ internes Schema

- beschreibt die physischen Speicherstrukturen der DB (Details der Datenspeicherung, Zugriffspfade, ...)
- nutzt ein physisches DM

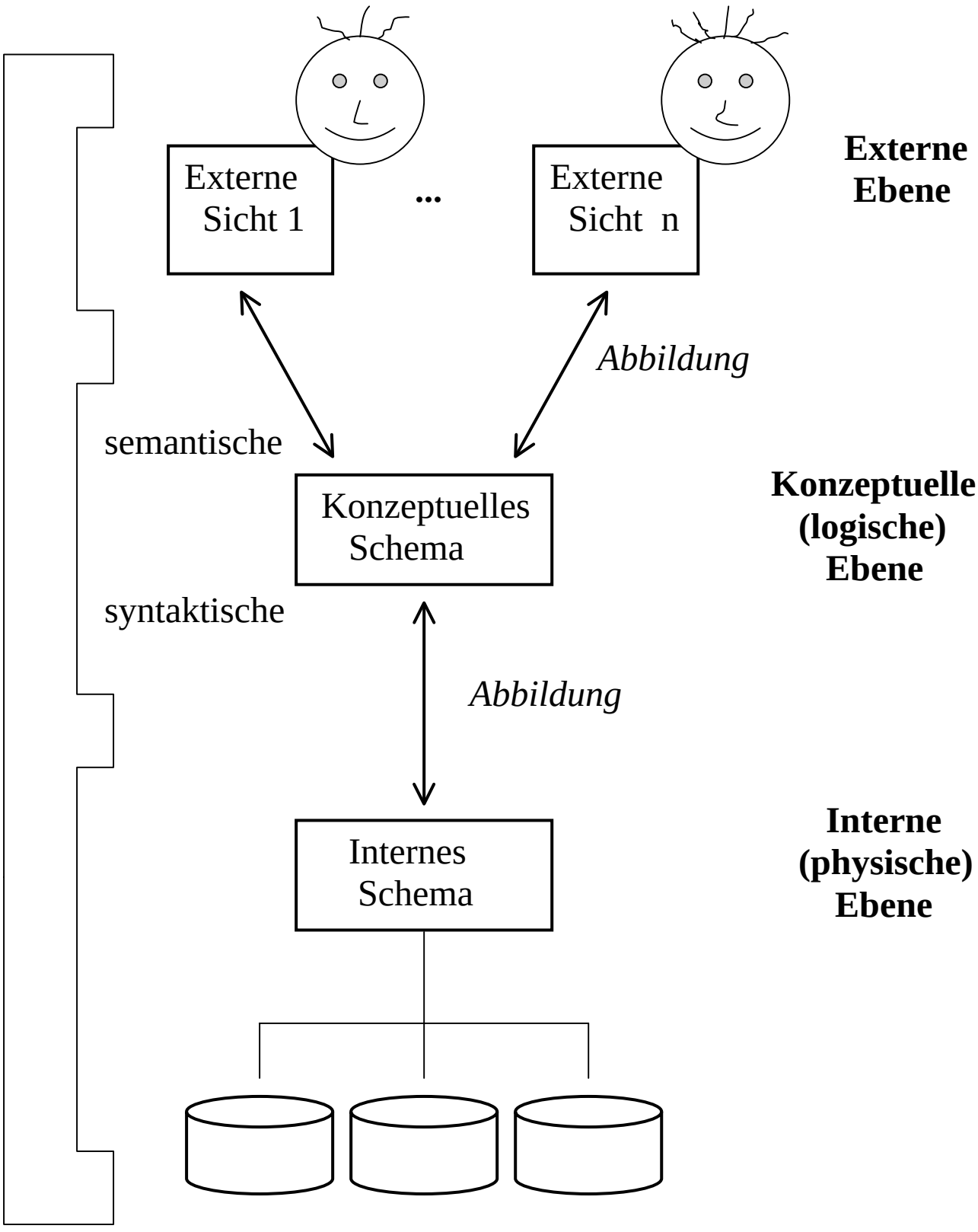


Abbildung – Transformation von Anfragen und Resultaten zwischen den Ebenen

Datenunabhängigkeit

Möglichkeit der Änderung des Schemas auf einer Ebene ohne das Schema auf der nächsthöheren Ebene ändern zu müssen

1. **logische Datenunabhängigkeit**

   Möglichkeit der Änderung des konzeptuellen Schemas ohne nachfolgende Änderung des externen Schemas oder der Anwendungsprogramme

2. **physische Datenunabhängigkeit**

   Möglichkeit der Änderung des internen Schemas ohne nachfolgende Änderung des konzeptuellen (oder externen) Schemas

Datenunabhängigkeit

bedeutet

- **KEINE** Änderung des Schemas auf höherer Ebene
- **ABER** Änderung der Abbildung zwischen den Ebenen

  $\Rightarrow$ Änderung der Abbildungsinformation in den Metadaten

Datenbank-Entwurf (Überblick)

1. Anforderungsanalyse

- Daten-Anforderungen
- Funktionale Anforderungen
  - nutzerdefinierte Operationen
  - Suche, Änderungen, ...

2. Konzeptueller Entwurf

- Entwurf des konzeptuellen Schemas,
  d.h. detaillierte Beschreibung der
  Datenmengen,
  Beziehungen,
  Restriktionen
- Kontrolle der Vollständigkeit,
  Gewährleistung der funktionalen Anforderungen,
  Aufdeckung von Konflikten

3. Logischer Entwurf

- Umwandlung des konzeptuellen Schemas in ein logisches Datenmodell eines kommerziellen DBMS
- Beschreibung der Konstrukte und Strukturen auf logischer Ebene im Rahmen des logischen Datenmodells des DBMS

4. Physischer Entwurf

- Spezifikation der internen Speicherstrukturen und Zugriffspfade für die DB

Parallel dazu:
Entwurf der Anwenderprogramme und
Implementierung als DB-Transaktionen

Ausblick
auf weitere ausgewählte Kapitel

- **Integritätskontrolle**

  Gewährleistung der einfachen Widerspruchsfreiheit und weiterer Einschränkungen bzgl. der **Semantik** der Anwendung

  1. Funktionale und mehrwertige Abhängigkeiten
     $\Rightarrow$ Normalformen
  2. Arten von Integritätsverletzungen
     (Klassifizierung der Integritätsbedingungen)
     $\rightarrow$ *Beispiel*
  3. Integritätssicherung mit SQL

- **Datenschutz**

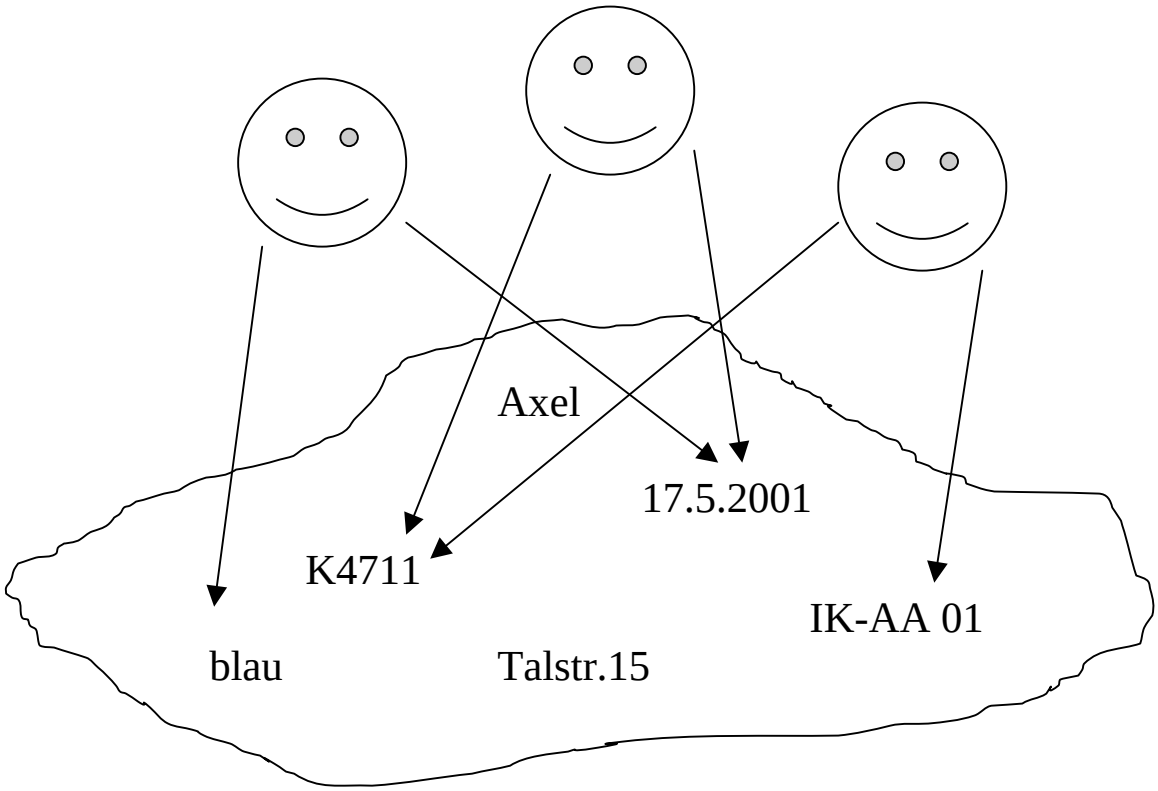
  Sicherstellung, dass die Daten in der DB
  - nur von den berechtigten (autorisierten) Nutzern
  - in der jeweils für sie zulässigen Form (select, insert, delete, update) bearbeitet
  - und nur auf den vorgesehenen Wegen transportiert und weiterverarbeitet

  werden.

  $\rightarrow$ *insbesondere statistische Datenbanken*

- **Transaktionsmanagement**

  Organisation des Mehrbenutzerbetriebes auf der DB



- **Recovery**

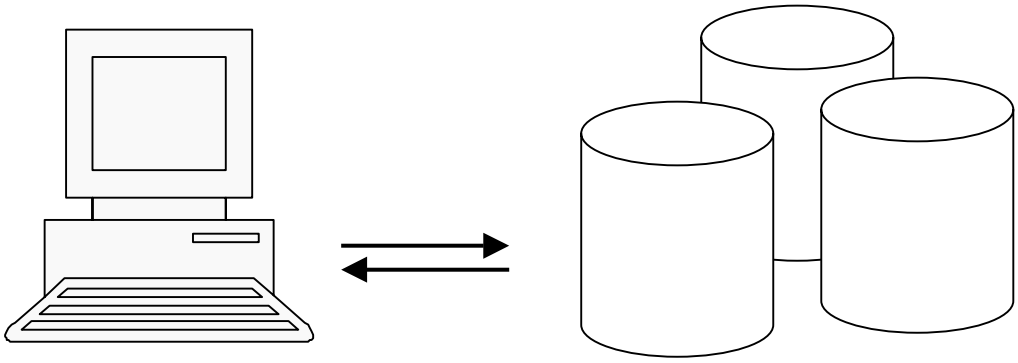
  Wiederherstellung der DB bei
  - Konflikten und Abbruch
  - Fehlern

- **Datensicherung**

  Maßnahmen zur Sicherung der gespeicherten Daten gegen Verfälschungen aller Art
  (fehlerhafte Eingaben, Programmfehler, Maschinenausfall, Datenträgerdefekte, ...)

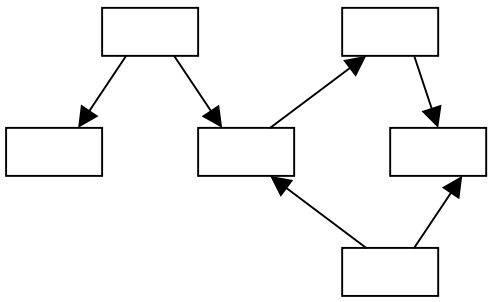
- **Interne Ebene**

  Speicherung der Daten auf einem
  Sekundärspeichermedium;
  Wiederauffinden;
  Übertragung und Verfügbarkeit für die verarbeitenden
  Programme

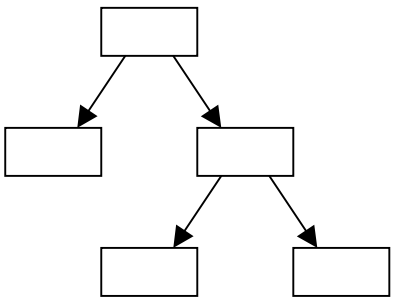


- **Weitere Datenmodelle (und DBMS)**

Netzwerk-Datenmodell



Hierarchisches Datenmodell



- **Nicht-Standard-Datenbanksysteme**

  und

- **Neuere Anwendungsgebiete**

6. Datenintegrität

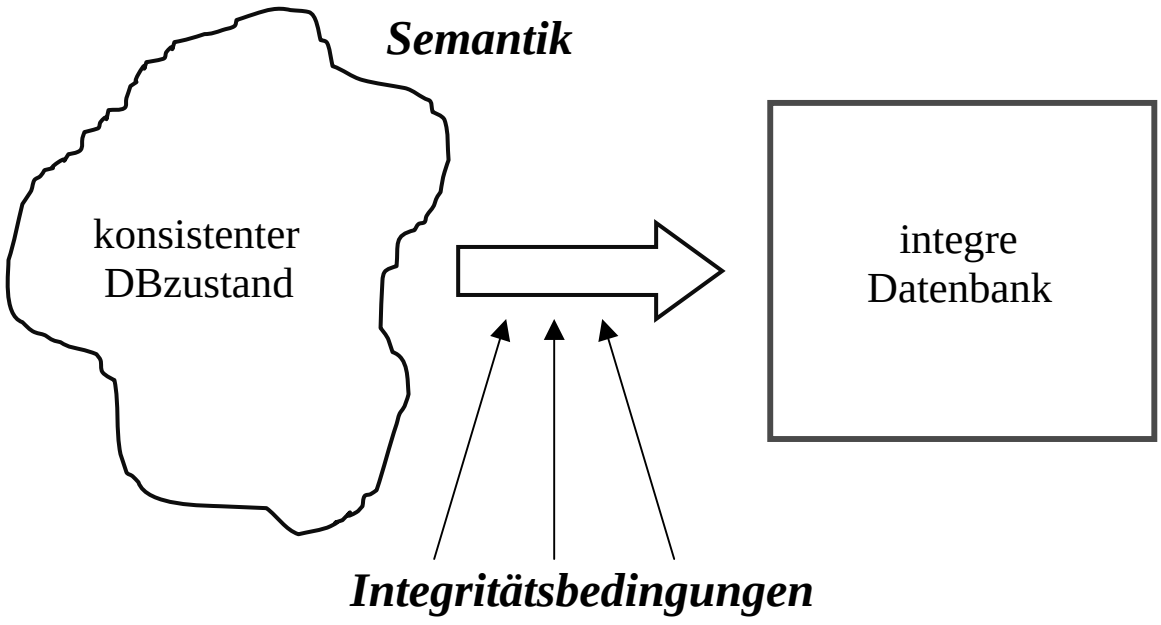
Gewährleistung der einfachen Widerspruchsfreiheit der Daten und weiterer Einschränkungen bzgl. der Semantik der Anwendungs“welt“

Konsistenz
ein korrekter, in sich widerspruchsfreier DBzustand

Integrität
Korrektheit der Daten auch noch bezüglich der Semantik der Anwendung

$\Rightarrow$

Integritätsbedingungen
Bedingungen, die den möglichen (konsistenten) DBzustand aufgrund der Semantik der Anwendung weiter einschränken



Arten von Integritätsbedingungen

1. statische

Einschränkung der möglichen DBzustände

0. Funktionale Abhängigkeiten
   Mehrwertige Abhängigkeiten
   ⇒ Normalformen

1. Attributbedingungen
   a) Ober- bzw. Untergrenzen
   b) Bestimmte Werte
   c) NOT NULL – Bedingung

2. Tupelbedingungen

3. Relationenbedingungen
   a) Schlüsselbedingung
   b) Aggregat-Bedingungen
   c) Rekursive Bedingung

4. Referentielle Bedingungen

2. transitionale

Einschränkung der möglichen Zustandsübergänge

3. dynamische

Einschränkung möglicher Zustandsfolgen

Formaler (relationaler) DBentwurf – Fortsetzung

Funktionale Abhängigkeit

Ein Attribut B ist funktional abhängig von einem Attribut A, wenn zu jedem Wert von A genau ein Wert von B existiert.
A ⟶ B

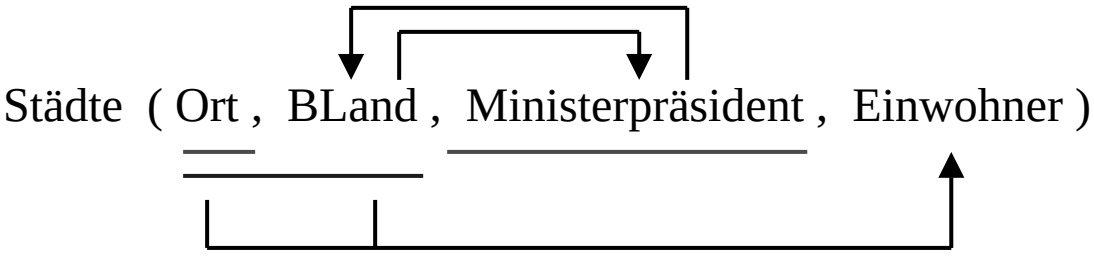
Dritte Normalform (3NF)

Eine Relation befindet sich in **3NF**, wenn sie in 1NF ist und jedes Nichtschlüsselattribut nicht transitiv abhängig vom Schlüssel ist.

Boyce-Codd-Normalform (BCNF)

Eine Relation befindet sich in **BCNF**, wenn sie in 1NF ist und jedes Attribut nicht transitiv abhängig vom Schlüssel ist.

Beispiel (aus Kemper/Eickler)



3NF erfüllt
BCNF nicht erfüllt

⇒ ***Zerlegung***

Regierungen (BLand , Ministerpräsident)

Städte (Ort , BLand , Einwohner)

Mehrwertige Abhängigkeiten

Ein Attribut C ist mehrwertig abhängig vom Attribut A, falls zu jeder Kombination eines bestimmten Wertes aus A mit einem beliebigen zugehörigen Wert aus B eine identische Menge von Werten aus C in den Tupeln der Relation existiert.

Bezeichnung: **A** $\rightarrow\!\!\!\!\rightarrow$ **C**

D.h.

Eine mehrwertige Abhängigkeit besteht, wenn zu einem Attribut A zwei mehrwertige (mengenwertige) Attribute B und C im Schema existieren und – nach der 1NF – alle Wertkombinationen (kartesisches Produkt) von zu einem bestimmten A-Wert gehörigen B- und C-Werten in den Tupeln vorkommen.

Vierte Normalform (4NF)

Eine Relation ist in **4NF**, wenn keine zwei echte und voneinander verschiedene mehrwertige Abhängigkeiten bestehen.

Auto

Marke	Typ	Farben
Skoda	Octavia	rot
	Fabia	grün
		blau
BMW	Z8	schwarz
	X5	rot

Auto

Marke	Typ	Farben
Skoda	Octavia	rot
Skoda	Octavia	grün
Skoda	Octavia	blau
Skoda	Fabia	rot
Skoda	Fabia	grün
Skoda	Fabia	blau
BMW	Z8	schwarz
BMW	Z8	rot
BMW	X5	schwarz
BMW	X5	rot

1NF
2NF
3NF

Autos

Marke	Typ
Skoda	Octavia
Skoda	Fabia
BMW	Z8
BMW	X5

Farbtab

Marke	Farbe
Skoda	rot
Skoda	grün
Skoda	blau
BMW	schwarz
BMW	rot

4NF

Statische Integritätsbedingungen

Beispiel

Prüfungen

Fach	Prüfer	Fachgebiet	StudNr	Name	Datum	Note
DVS	Apel	DB	23745	Schulz	3.2.2001	1,3
Logik	Schmidt	Theor.Inf.	2439	Meier	4.7.1899	7,0
Physik	Sauer	Allg. ET	NULL	NULL	2.3.2002	NULL

Fachbereiche

Fachgebiet	Leiter	Telefon	Budget	Ausgaben
Datenbanken	Bauer	4375	15000,00	17341,77
Theo Inf	Vogel	2143	10000,00	4750,00
Allg ET	Kühn	1289	25000,00	28893,15

⇒ Verletzung der Integrität

1. Attributbedingungen
   a) Ober- bzw. Untergrenzen
   b) Bestimmte Werte
   c) NOT NULL-Bedingung
2. Tupelbedingungen
3. Relationenbedingungen
   a) Schlüsselbedingung
   b) Aggregat-Bedingungen
4. Referentielle Bedingungen

Integritätssicherung in SQL (Auswahl)

Passive Integritätssicherung

- **... CHECK (attributbedingung)**
- **FOREIGN KEY (attr) REFERENCES relation (attr)**

Aktive Integritätssicherung

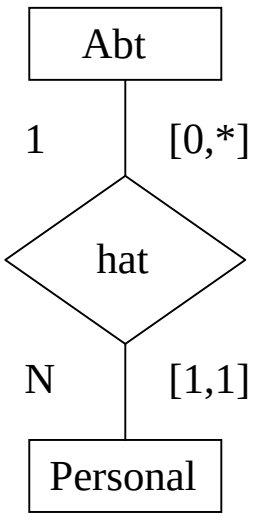
- **… REFERENCES relation (attribut)**

  ON { **DELETE** / **UPDATE** } { **CASCADE** / **RESTRICT** / **SET NULL** / **SET DEFAULT** }

Beispiel

```
CREATE  TABLE  Abt
( ANr  INTEGER  NOT NULL ,
  Bezeichnung  CHAR (15) ,
  …
  PRIMARY  KEY ( ANr ) ) ;

CREATE  TABLE  Personal
( PNr  INTEGER  NOT NULL ,
  Name  CHAR (30) ,
  ANr  INTEGER  NOT NULL ,
  …
  PRIMARY  KEY (PNr) ,
  FOREIGN  KEY  (ANr)  REFERENCES Abt (ANr)
           ON  DELETE  CASCADE
           ON  DELETE  DEFAULT
```



Tupel- und tabellenübergreifende, statische Integritätsbedingungen

- CREATE **ASSERTION** Budget
  CHECK ((SELECT SUM (Gehalt)
  FROM Professor) < 100000)

- CREATE TABLE Professor
  (…)

  CREATE TABLE Vorlesung
  (…
  CONSTRAINT Fremdschlüssel
  FOREIGN KEY PNr REFERENCES Professor (PNr)
  INITIALLY DEFERRED)

  INSERT INTO Vorlesung
  VALUES (…)

  INSERT INTO Professor
  VALUES (…)

  SET CONSTRAINT Fremdschlüssel IMMEDIATE

Trigger

- CREATE TRIGGER ReferentielleIntegrität
  AFTER INSERT
  ON Vorlesung
  WHEN PNr NOT IN (SELECT PNr
  FROM Professor)
  INSERT INTO Professor
  VALUES (PNr, 'NN.', NULL, NULL, NULL, NULL,
  NULL) ;

Die Tabellen-Definitionen lauteten (S. 33)

```
CREATE TABLE Professor
( PNr INTEGER PRIMARY KEY ,
  PName VARCHAR(20) NOT NULL ,
  Gehalt DECIMAL(7,2) ,
  Fachgebiet VARCHAR(20) ,
  Lehrsoll INTEGER CHECK ( Lehrsoll >= 0 ) ,
  Tel INTEGER UNIQUE ) ;

CREATE TABLE Vorlesung
( Vname VARCHAR(25) ,
  Studiengang VARCHAR(20) ,
  SWS INTEGER ,
  Semester CHAR(2)
           CHECK ( Semester IN ('WS','SS') ) ,
  PNr INTEGER ,
  PRIMARY KEY ( Vname , Studiengang ) ,
  FOREIGN KEY ( PNr ) REFERENCES Professor ) ;
```

⇒ ECA-Regeln
E - Event
C - Condition
A - Action ⇒ aktive Datenbanken

7. Datenschutz

(ausgewählte Aspekte)

Datenschutz

Maßnahmen zur Sicherstellung, dass alle Daten in der Datenbank

- nur von den berechtigten (autorisierten) Nutzern
- in der jeweils für sie zulässigen Form bearbeitet (select, delete, insert, update)
- und nur auf den vorgesehenen Wegen transportiert und weiterverarbeitet

werden.

→ *insbesondere statistische Datenbanken*

Datensicherheit

Maßnahmen, so dass die gespeicherten Daten gegen Verfälschungen aller Art

(z.B. fehlerhafte Eingaben,
Programmfehler,
Maschinenausfall,
Datenträgerdefekte, ...)

wirksam gesichert sind.

⇒ *beliebige Lebensdauer*
(so kurz wie möglich, aber so lang wie nötig)

Begriffsklärungen

Identifikation

Anmeldung eines Benutzers beim System unter Angabe einer Benutzerkennung

Authentisierung

bezeichnet den Prüfvorgang durch das System, mit dem sichergestellt werden soll, dass ein Benutzer, der sich mit einer bestimmten Kennung anmeldet, auch tatsächlich der zur Führung dieser Kennung Berechtigte ist

Autorisierung

Vergabe von Zugriffsrechten (oder allg.: Nutzungsrechten) an einen Benutzer.
Der autorisierende Benutzer muß die Rechte besitzen, die er weitergibt, und das Recht zu ihrer Weitergabe.

Anforderungen an den Zugriffsschutz

Für Datenbanksysteme sind spezifische Aspekte zu berücksichtigen.

1. abgestufte Schutzeinheiten
   - verschiedene Grade der Schutzbedürftigkeit
   - Objekte verschiedener Granularität
2. eindeutige Schnittstellen
3. dezentrale Autorisierung
   durch z.B. Fachabteilungen
4. dynamische Autorisierung
5. kooperative Autorisierung
6. verschiedenartige Entscheidungskriterien
   - Name und Inhalt des Objekts
   - Inhalt anderer Objekte in der DB
   - system- und nutzerspezifische Zustandsanzeiger
   - Art und Ergebnisse vorheriger DB-Zugriffe
7. Datenfluss- und Inferenz- Kontrolle
8. geringer Einfluss auf die Systemleistung
9. integrale Zugriffskontrolle im DBMS
10. Kernel-Architektur für die Schutzkomponente

⇒ *Schutzkonzepte*
1. , 3. , 4. , 6. , 7.

⇒ *Schutzmechanismen*
2. , 6. , 7. , 8. , 10.

⇒ *Beziehungen zum Betriebssystem und der übrigen Umgebung*
3. , 6. , 8. , 10.

Schutzkonzepte (Auswahl)

1. Isolation
   Keine gemeinsame Nutzung der Daten

2. unbeschränkte Zugriffskontrolle
   viele nutzen alle Daten gemeinsam unter Wahrung der Schutzbedürfnisse
   - Objekt-bezogene Kontrolle
   - Subjekt- und Objekt-bezogene Kontrolle
   - Objekt- und Operations-bezogene Kontrolle
   - Subjekt-, Objekt- und Operations-bezogene Kontrolle

   → Sicherheitsmatrix

3. Prinzip des kleinstmöglichen Privilegs
   - abgestufte Schutzgranulate
   - allgemeine wertabhängige Zugriffskontrolle

4. Ordnung der Nutzungsprivilegien

5. Ordnung der Zugriffsoperationen

6. Ordnung der Schutzbedürftigkeit der Objekte

⇒ Kein Nutzer darf Daten **lesen**, die einen höheren Geheimhaltungsgrad haben, als seine Berechtigung ausweist.

⇒ Kein Nutzer darf Daten mit einer niedrigeren Geheimhaltungsstufe **schreiben**, als seine Berechtigung ausweist.

Beispiel für eine wertabhängige Zugriffskontrolle

DBMS Ingres
Sprache: QUEL

Relation

Angestellte (Pers-Nr , Name , Gehalt , Abt , ...)

Schutz- und Integritätsbedingung:

```
RANGE OF X IS Angestellte
RESTRICT ACCESS FOR 'Schulze' TO Angestellte
WHERE X . Abt = 'A0815'
```

Anfrage:

```
RANGE OF X IS Angestellte
RETRIEVE INTO Liste ( X . Name )
WHERE X . Gehalt > 5000
    AND X . Abt = 'A0815'
```

⇒ ‚query modification'

Sichten (Views)

```
CREATE  VIEW  Abt-Angestellte
AS  SELECT  *
        FROM  Angestellte
      WHERE  Abt = ‘A0815’ ;
```

Problembereiche bei Änderungen

- Verletzung der Schemadefinition
  d.h. Vermeidung von **Integritätsverletzungen**

- Vermeidung von Seiteneffekten auf den nicht-sichtbaren Teil der DB aus **Datenschutz**gründen

- Lösung des Auswahlproblems bei **mehreren Transformationsmöglichkeiten**

- In vielen Fällen **keine sinnvolle Transformation** möglich

- Forderung:
  elementare Änderung auf der Sicht entspricht genau einer atomaren Änderung auf der Basisrelation
  d.h.
  1:1-Beziehung zwischen Sichttupeln und Basistupeln

Beispiel (aus Biber-Buch)

MGA (Mitarbeiter , Gehalt , Abteilung)
AL (Abteilung , Leiter)

- Projektionssicht

```
CREATE  VIEW  MA
AS  SELECT  Mitarbeiter ,  Abteilung
      FROM  MGA ;
```

?
```
INSERT  INTO  MA
VALUES ( 'Zuse' ,  'Info' ) ;
```

- Selektionssicht

```
CREATE  VIEW  MG
AS  SELECT  Mitarbeiter ,  Gehalt
      FROM  MGA
     WHERE  Gehalt > 20 ;
```

?
```
UPDATE  MG
SET  Gehalt = 15
WHERE  Mitarbeiter = 'Zuse' ;
```

Beispiel weiter

MGA (Mitarbeiter , Gehalt , Abteilung)
AL (Abteilung , Leiter)

- Verbundsichten

```
CREATE  VIEW  MGAL
AS  SELECT  Mitarbeiter , Gehalt ,
                  MGA . Abteilung , Leiter
       FROM  MGA , AL
     WHERE  MGA . Abteilung = AL . Abteilung ;
```

?
```
INSERT  INTO  MGAL
VALUES  ( 'Turing' , 30 , 'Info' , 'Zuse' ) ;
```

- Aggregierungssichten

```
CREATE  VIEW  ASG ( Abteilung , SummeGehalt )
AS  SELECT  Abteilung , SUM ( Gehalt )
       FROM  MGA
    GROUP  BY  Abteilung ;
```

?
```
UPDATE  ASG
SET  SummeGehalt = SummeGehalt + 1000
WHERE  Abteilung = 'Info'
```

Schutzprobleme bei statistischen Datenbanken

- Datenbanksysteme mit ihrer Funktionalität erlauben fundierte (statistische und auf die Erkennung von Zusammenhängen ausgerichtete) Untersuchungen auf großen Datenbeständen.

- **Gefahr**,
  wenn Datenbanken schutzwürdige Informationen
  z.B. über Personen enthalten
  und
  Nutzer zwar keinen Zugang zu den einzelnen Personaldatensätzen haben, wohl aber Häufigkeiten, Mittelwerte usw. von Attributwerten anfordern dürfen.

Beispiele

  - Statistische Untersuchungen an Patientendaten zur Ermittlung von Zusammenhängen zwischen bestimmten Krankheitsbildern und anderen persönlichen Merkmalen

  - Soziologische Untersuchungen verschiedenster Art an Daten des Einwohnermeldeamtes, Arbeitsamtes o.ä.

Angestellte
(PNr , Name , Gehalt , Abt , EinstDat , Tätigkeit , FamStand)

⇒ **Sicht** StatAng
(Gehalt , Abt , EinstDat , FamStand)

Statistiker sei autorisiert für Häufigkeitsanfragen über alle Attributwert-Kombinationen:

- SELECT COUNT (*)
  FROM StatAng
  WHERE EinstDat < 71-01-01 ; ⇒ 2 Tupel

- …
  WHERE EinstDat < 71-01-01 ;
  AND FamStand = ‘ verheiratet ’ ; ⇒ 1 Tupel

- …
  WHERE EinstDat < 71-01-01 ;
  AND FamStand = ‘ verheiratet ’
  AND Gehalt > 120000 ; ⇒ 1 oder 0

Was tun ?

- Anfragen mit Ergebnismengen < k Tupeln abweisen

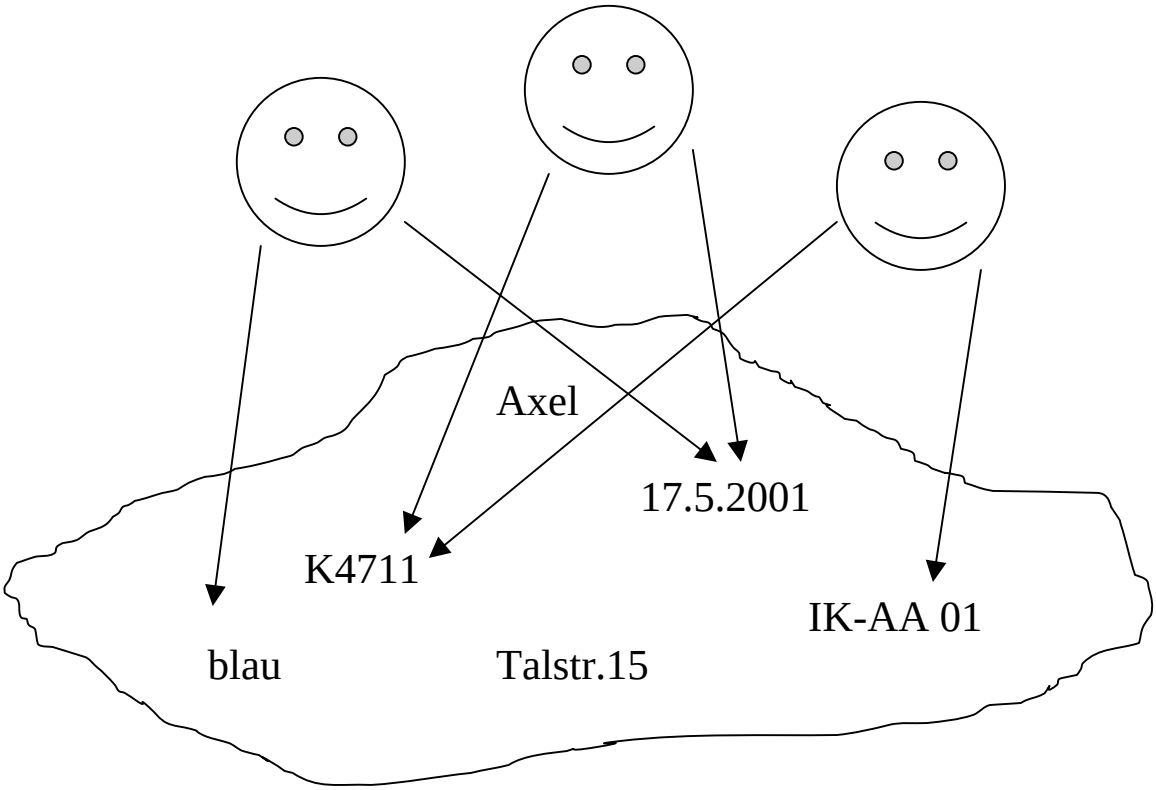
- keine Anfragen, die paarweise einen Durchschnitt von > m vorgegebenen Tupeln betreffen (Überlappung der Ergebnismengen)

- Kontrolle auf „Tracker“-Algorithmus

- Zufällige kleine Fehler

8. Transaktionsmanagement

(Mehrbenutzerverwaltung)



Organisation des Mehrbenutzerbetriebes
und
Synchronisation der Zugriffe auf Einzel“informationen“

⇒

- sehr kritisch
  viele Nutzer wollen häufig und gleichzeitig Daten abrufen und Änderungen auf der DB ausführen

- weniger kritisch
  viele Nutzer wollen häufig und gleichzeitig Daten abrufen

Transaktion

Die **Transaktion** ist eine **ununterbrechbare Abfolge** von einzelnen Verarbeitungsschritten und **Zugriffsoperationen** eines Nutzers **auf der Datenbank** im Rahmen einer kleinen Aufgabenstellung bzw. Anfrage, die die Datenbank von einem **konsistenten** Zustand in einen (nicht unbedingt verschiedenen) **konsistenten** Zustand überführt.

Hier: Konsistenz auch im Sinne der Einhaltung aller semantischen Integritätsbedingungen

Eigenschaften einer Transaktion
⇒

ACID – Prinzip

Atomicity - atomar
ununterbrechbare Abfolge

Consistency - konsistent
Erhaltung der Korrektheit

Isolation - isoliert
isolierter Ablauf bzgl. anderer Transaktionen

Durability - dauerhaft
korrekte Ergebnisse (Daten) wieder dauerhaft in der Datenbank

NUN

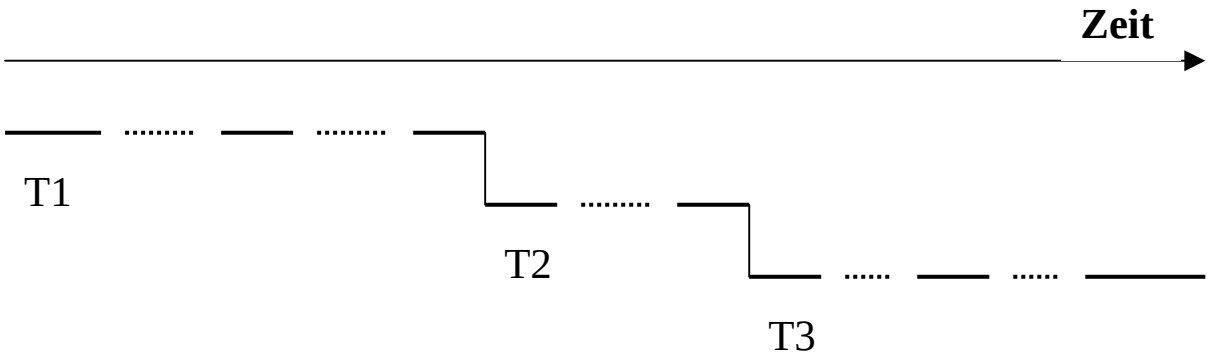
viele Transaktionen gleichzeitg

DAZU

Insbesondere DBanwendungen müssen oft auf

- langsame Betriebsmittel
  (z.B. Sekundärspeicher)
- interaktive Benutzereingaben

warten.



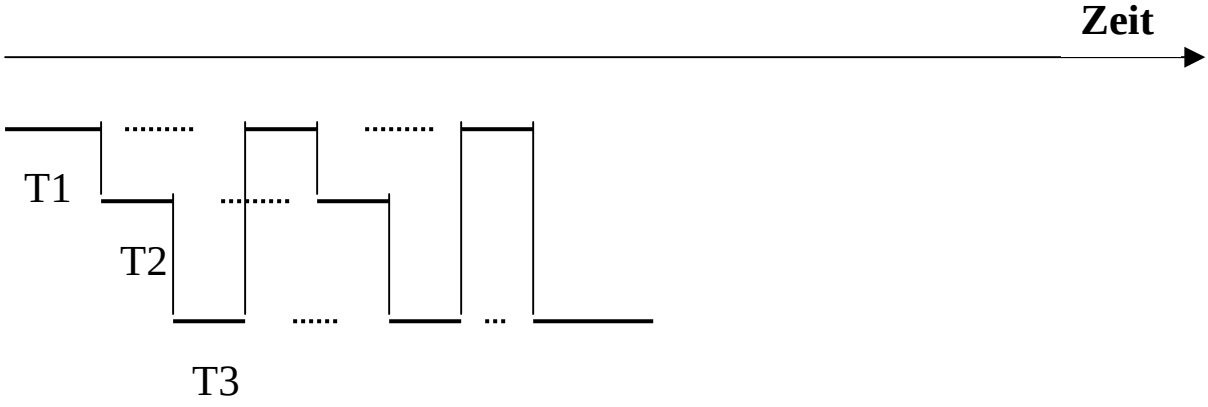
⇒

BESSERE

Auslastung des Computersystems

⇒

verzahnte Ausführung der Transaktionen unter Wahrung aller sonstigen Bedingungen



Synchronisation der Zugriffe

- Zugriffe
  read - lesen
  write - schreiben

  bringen Transaktion mit DB in „Beziehung“
  und
  in Konkurrenz mit anderen Transaktionen

  Verarbeitungsschritte dazwischen, z.B. Erzeugung neuer Werte aus den gelesenen Daten, stellen keine unmittelbare Verbindung zwischen Nutzer und DB dar.

  ABER

  Reihenfolge der Verarbeitungsschritte (Algorithmus) innerhalb einer Transaktion muss auch eingehalten werden.

- Weitere Aktionen im Zusammenhang mit Transaktionen:

  abort - Abbruch
  mit Zurücksetzen der Transaktion

  commit - erfolgreicher Abschluss der Transaktion

  running - Transaktion ist in Bearbeitung

  sleeping - Transaktion ist im Warte-Zustand

  begin - Beginn der Transaktion

Der Transaktionsmanager

Bestandteile

- **Scheduler**
  - Verzahnung der Einzelaktionen verschiedener Nutzer und Erstellung eines globalen Ausführungsplanes (unter Einhaltung des Algorithmus innerhalb der Transaktion)
  - Feststellung unvereinbarer Anforderungen
    → Abbruch und Übergabe an Recovery-Manager

in Zusammenarbeit mit

- Data-Manager
  holt dazu benötigte Daten aus der bzw. bringt (neue oder geänderte) Daten in die DB

- Recovery-Manager
  Zurücksetzen abgebrochener Transaktionen schrittweise bis zur Ausgangssituation

Typische Probleme paralleler Transaktionen

1. Lost Update

T1	Zeitpunkt	T2
BOT(T1)	1	
r(x)	2	
	3	BOT(T2)
	4	r(x)
u(x)	5	
	6	u(x)
w(x)	7	
EOT(T1)	8	
	9	w(x)
	10	EOT(T2)

2. Dirty Read

T1	Zeitpunkt	T2
BOT(T1)	1	
r(x)	2	
u(x)	3	
w(x)	4	
	5	BOT(T2)
	6	r(x)
	7	u(x)
ABORT	8	
	9	w(x)
...	10	**...**

3. Unrepeatable Read (Phantom-Problem)

T1	Zeitpunkt	T2
BOT(T1)	1	
sum:= 0	2	
r(x)	3	
r(y)	4	
sum:= sum + x	5	
sum:= sum + y	6	
	7	BOT(T2)
	8	r(z)
	9	z := z – 100
	10	w(z)
	11	r(x)
	12	x := x + 100
	13	w(x)
	14	EOT(T2)
r(z)	15	
sum:= sum + z	16	
EOT(T1)	17	

Schedules und deren Korrektheit

Log

Graph G_T (A,P) - Knoten A
Aktionen read und write
Kanten P
$P \subset A \times A$
Reihenfolge von Aktionen w und r auf dem gleichen Objekt ist geordnet

Schedule

absolute (verzahnte) Ausführungsreihenfolge **aller** einzelnen Aktionen der zu diesem Zeitpunkt anliegenden Transaktionen

Korrekt ?

Serialisierbarkeit

Zu dem Schedule gibt es **einen** seriellen Schedule (strikte Nacheinanderausführung der einzelnen Transaktionen als Ganzes), der den **gleichen** DBzustand erzeugt.

Kontrolle ?

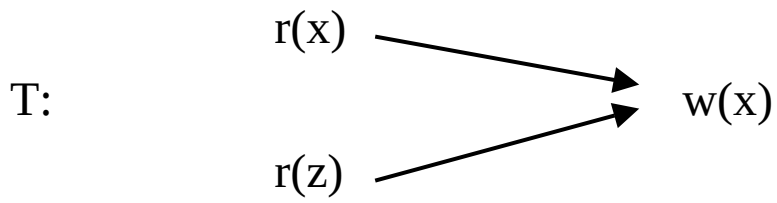
Serialisierbarkeitsgraph

ohne Zyklen $\rightarrow$ serialisierbar
mit Zyklen $\rightarrow$ nicht serialisierbar

$\Rightarrow$ **mühsam**

Beispiel

Transaktions-Log



G = (A , P)
wobei
A = { r(x) , r(z) , w(x) }
P = { (r(x) , w(x)) , (r(z) , w(x)) }

Forderung: Reads und Writes auf demselben Objekt sind geordnet

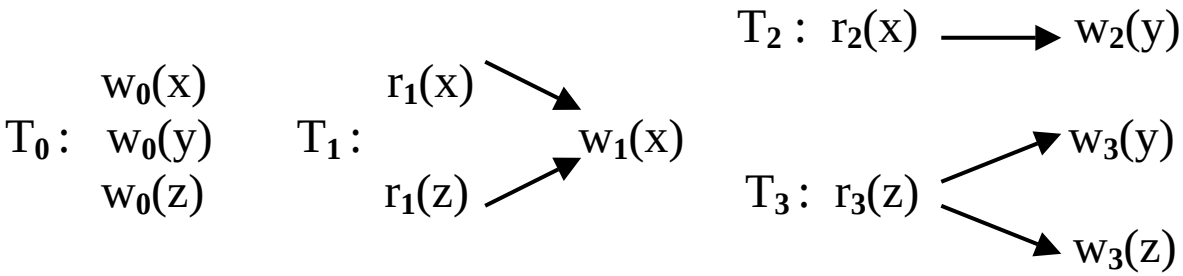
Schedule $\rightarrow$ Totalordnung für P

Schedule

Mögliche Schedules für T sind:

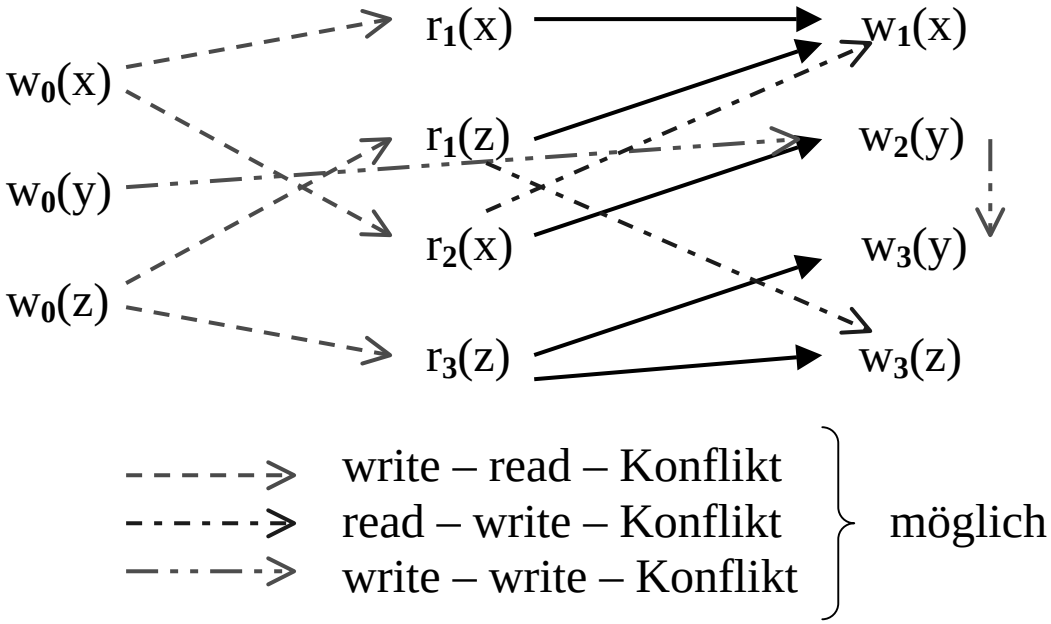
r(x) , r(z) , w(x)
oder
r(z) , r(x) , w(x)

Beispiel



$T = \{ T_0 , T_1 , T_2 , T_3 \}$

Log L



Schedules

- L1 = { $w_0(x) , w_0(y) , w_0(z) , \quad r_1(x) , r_1(z) , r_2(x) , r_3(z) , w_1(x) , w_2(y) , w_3(y) , w_3(z)$ }

- L2 = { $w_0(x) , w_0(y) , w_0(z) , \quad r_2(x) , w_2(y) , r_1(x) , r_1(z) , w_1(x) , \quad r_3(z) , w_3(y) , w_3(z)$ }

  $\Rightarrow$ serieller Schedule

Zwei-Phasen-Sperrprotokoll

(Two-Phase-Locking → 2PL)

Ansatz: Zugriff auf ein DB-Objekt x durch verschiedene Transaktionen in sich gegenseitig ausschließender Weise

⇒ Sperren - lock (x)
Entsperren - unlock (x)

- Sperren
  zum Lesen ⇒ rlock (x)
  zum Schreiben ⇒ wlock (x)
- Entsperren
  ⇒ unlock (x)

Regeln für die Verwendung von Sperren

1. Falls T_i eine Aktion $r_i(x)$ bzw. $w_i(x)$ enthält, so ist das Objekt x vor der Ausführung der Aktion durch $rlock_i(x)$ bzw. $wlock_i(x)$ zu sperren. Die Sperre ist für die Dauer der Ausführung zu halten.
2. Die gesetzte Sperre wird später durch $unlock_i(x)$ aufgehoben.
3. Sperren derselben Art auf ein Objekt x werden innerhalb einer Transaktion T_i nur einmal gesetzt.

Verträglichkeit

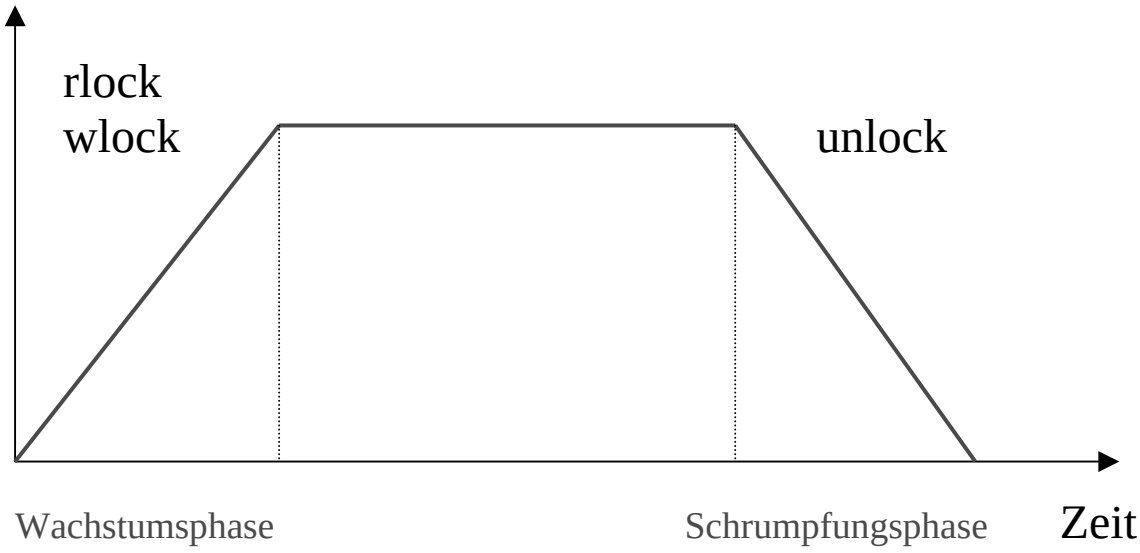
	$rlock_i$	$wlock_i$
$rlock_j$	√	–
$wlock_j$	–	–

Sperrprotokoll

I. Ein Scheduler arbeitet nach einem Sperrprotokoll, falls in jedem Schedule für einen von ihm erzeugten Log für $T=\{T_0, \ldots, T_n\}$ gilt:
   a) Jede Transaktion T_i, i = 0 (1) n , genügt den Regeln 1.-3.
   b) Ist x durch T_i und T_j gesperrt, so sind diese beiden Sperren kompatibel.

Zwei-Phasen-Sperrprotokoll

II. Ein Sperrprotokoll heißt 2-Phasen-Sperrprotokoll, wenn für alle $T_i \in T$ gilt: Nach der ersten *unlock*-Aktion folgen keine weiteren *rlock*- bzw. *wlock*-Aktionen.



Variationen möglich !

Satz: Es sei L ein Schedule, der von einem 2PL-Scheduler erzeugt wurde. Dann ist L serialisierbar.

⇒

Vorteil

Die verzahnte Ausführung mehrerer Transaktionen nach einem Zwei-Phasen-Sperrprotokoll ist korrekt (serialisierbar).

Nachteil

Es können sogenannte Deadlock-Situationen entstehen.

T1	Zeit	T2
. . .	1	. . .
$rlock_1(x)$	2	
	3	$rlock_2(y)$
$wlock_1(y)$	4	
	5	$wlock_2(x)$
. . .	6	. . .

⇒

Auswege

1. Vergabe von Zeitmarken
   → Abbruch, wenn Zeit überschritten
2. Wartegraphen protokollieren
   Knoten: aktuell laufende Transaktionen
   Kanten: Transaktion wartet auf unlock einer anderen Transaktion

   Zyklen im Warte-Graphen bedeuten eine Deadlock- Situation
   → Abbruch wenigstens einer Transaktion

Achtung! Performance

Beispiel Phantom-Problem

T1	Zeitpunkt	T2
BOT(T1)	1	
	2	BOT(T2)
sum:= 0	3	
rlock(y)	4	
r(y)	5	
sum:= sum + y	6	
	7	wlock(x)
	8	wlock(z)
	9	r(x)
	10	r(z)
	11	z := z – 100
	12	x := x + 100
	13	w(z)
	14	unlock(z)
rlock(z)	15	
r(z)	16	
sum:= sum + z	17	
	18	w(x)
	19	unlock(x)
rlock(x)	20	
r(x)	21	
sum:= sum + x	22	
unlock(x)	23	
unlock(y)	24	
unlock(z)	25	
	26	EOT(T2)
EOT(T1)	27	

Fehlerbehandlung und Recovery

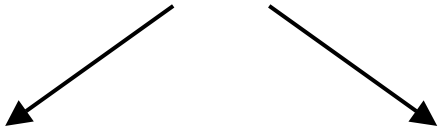
1. Transaktionsfehler
   Transaktion erreicht nicht ihr reguläres Ende
   (EOT und Zustand „committed“, weil z.B.
   - Inkonsistenzen auftraten und erkannt wurden
   - Abbruch nach Verwicklung in einen Deadlock

2. Systemfehler
   „Absturz“ einer Transaktion durch z.B.
   - Fehler im DBMS-Code
   - Betriebssystem-Fehler
   - Fehler in der Rechner-Hardware

3. Sekundärspeicherfehler
   Abbruch der Transaktion aus „externen“ Gründen
   - Soft- oder Hardware-Fehler bei einem Plattenzugriff
   - Head-Crash auf einer Magnetplatte

⇒ **Recovery-Manager**

Aufgabe: Rekonstruktion eines konsistenten Zustandes der Datenbank



Redo (2) u. (3)

Rollback bzw. Undo (1)

dazu
in bestimmten Zeitabständen Sicherungskopien des gesamten Datenbank-Zustandes („Dump“)

Recovery-Techniken

Log-Buch

Before-Image-Eintrag
- Identifikation der Transaktion T
- Identifikation für Objekt x
- alter Wert von x

Rollback
Log-Buch-Eintragungen für T von hinten beginnend fortlaufend lesen und für jedes Before-Image alten Wert des Objektes in die DB zurückschreiben

After-Image-Eintrag
- Identifikation der (schreibenden) Transaktion T
- Identifikation des Objektes x
- neuer Wert von x

Zwei-Phasen-Commit-Strategie
1. After-Images in Log-Buch eintragen
   → Transaktion T „committed“
2. Schreiben der Daten in die DB

(→ Zwei-Phasen-Sperrprotokoll)
unlock erst nach „committed“

Redo
alle „committed“ Transaktionen, die ins Log-Buch, aber noch nicht in DB geschrieben sind

Undo
alle übrigen

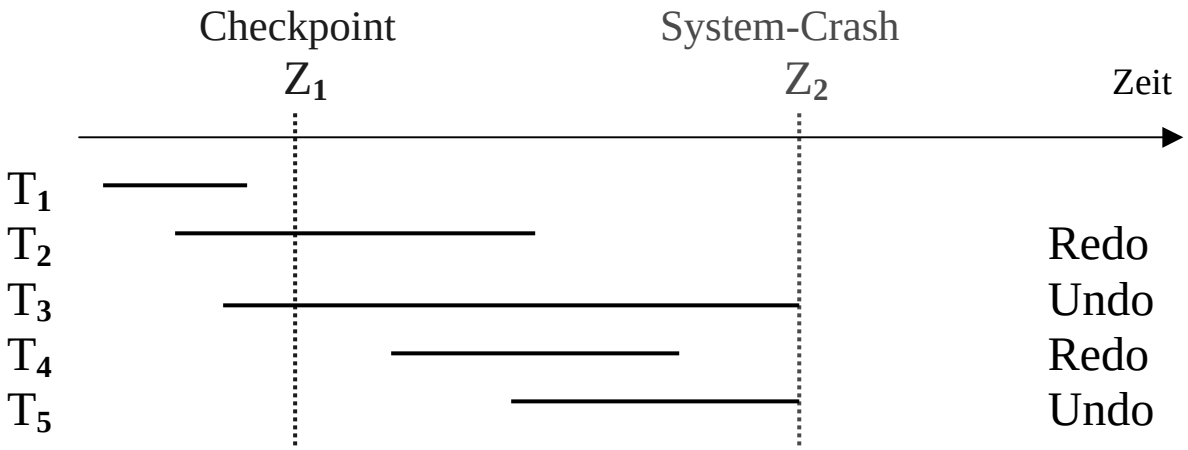
System-Crash $\Rightarrow$ Restart

- Puffer-Inhalte verlorengegangen
- Inkonsistenter DB-Zustand

$\rightarrow$ in festen Zeitintervallen „Checkpoints“ „setzen“, d.h.

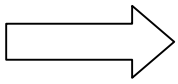
- Inhalt des Log-Puffers in Log-Buch übertragen
- „Checkpoint“-Eintrag im Log-Buch
- Inhalt des DB-Puffers in DB schreiben
- Adresse des „Checkpoint“-Eintrags im Log-Buch in einem „Restart-File“ vermerken

Checkpoint-Eintrag
Liste der gerade aktiven Transaktionen
(aus Log-Einträgen für BOT(T_i) und EOT(T_i))



9. Physische Datenorganisation

(Interne Ebene)

Speicherungsformen und Suchalgorithmen  Datenunabhängigkeit und Effizienz

Datenbanken: **großes** Datenvolumen

Externspeicher

⇒ **Externspeicherorganisation und -verwaltung**

Anzahl der Zugriffe auf Externspeicher ist i.a. die dominierende Einflussgröße für das **Antwortzeitverhalten** von DBsystemen

⇒ Abschätzung der Anzahl der Zugriffe wichtig für

Auswahl von Verfahren bzw.

Ausführungsalternativen

Stufen von Speichermedien

1. *Primärspeicher* → Hauptspeicher des Rechners
   - sehr teuer
   - sehr schnell
   - eher klein
   - Granularität sehr fein
     → alle Operationen auf Daten im HS ausführen
   - gegen Systemausfälle nicht gesichert

   → für DBsysteme Pufferfunktion

2. *Sekundärspeicher* → Festplatte
   - Zugriff auf Daten um Faktor 10^5 langsamer
   - viel Platz (mehr als HS)
   - relativ ausfallsicher
   - günstiger im Preis
   - Granularität grob
     kleinste Einheit für Zugriff: Block

   → für DBsysteme kleinste Einheit: Seite
   (d.h. mehrere in einer Spur liegende Blöcke)

3. *Archivspeicher* → Magnetbänder
   - sehr billig (Pfennige pro MB)
   - nahezu Kapazität der Festplatte
   - nur sequentiell lesen und beschreiben –
     damit Zugriffszeit nicht vergleichbar
   - ausfallsicher

   → für DBsysteme zur Protokollierung von
   Operationen

Physischer Entwurf

effektive Organisation der Daten und des Zugriffs auf den Sekundärspeicherspeicher

maßgebliche Faktoren beim physischen Entwurf

- **Zugriffszeit**
- Aufwand für die Wartung
- Platzbedarf der Daten

⇒ Operationen
- Suchen
- Einfügen
- Ändern
- Löschen

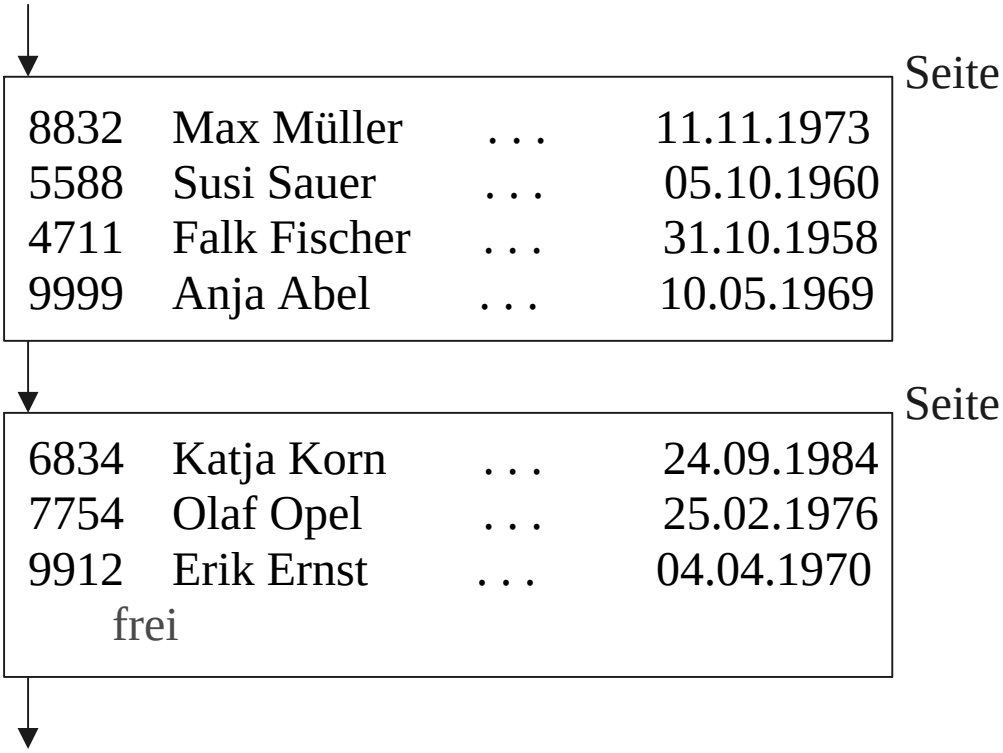
➢ Datensätze ohne Zusatzinformation in den Dateien gespeichert
→ ganze Datei durchsuchen

➢ *Index-Strukturen*
ermöglichen, die zu einem gegebenen Suchkriterium passenden Datensätze anzugeben
(unter Ausnutzung der Direktzugriffsmöglichkeiten des Sekundärspeichers)

Primärindex
legt physische Anordnung der indizierten Daten fest
→ für jede Datei nur **ein** Primärindex
(meist Primärschlüssel)

Sekundärindex
mehrere möglich (für andere Attribute)

Heap-Datei



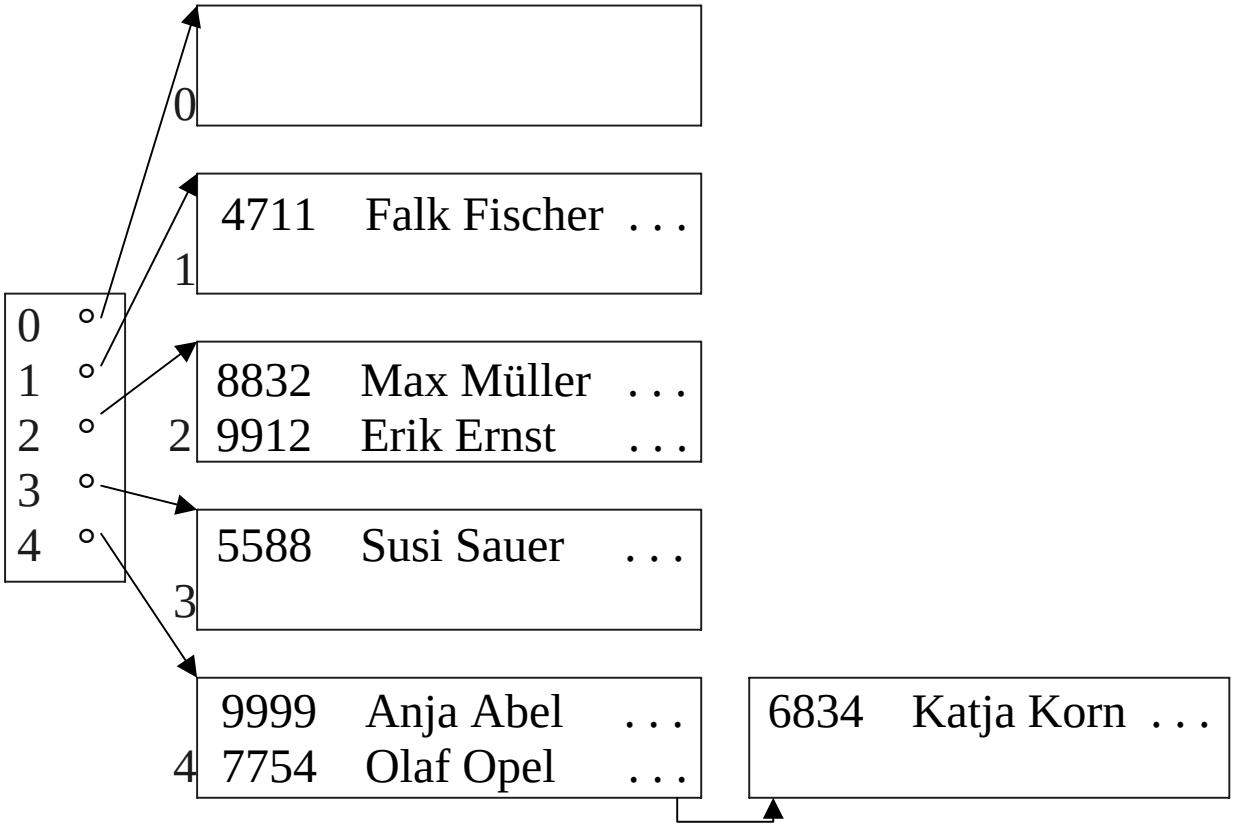
Operationen:

- Suchen
- Einfügen
- Löschen
- Ändern

Anfragen:

```
SELECT  Name
  FROM  Personal
WHERE  PNr = 8832 ;
. . .
WHERE  PNr > 7000 ;
. . .
WHERE  GebDat > 01.01.1975 ;
```

Hash-Verfahren



Liste der Segmente — Segmente

Hash-Funktion (z.B.) h(k) := k mod p

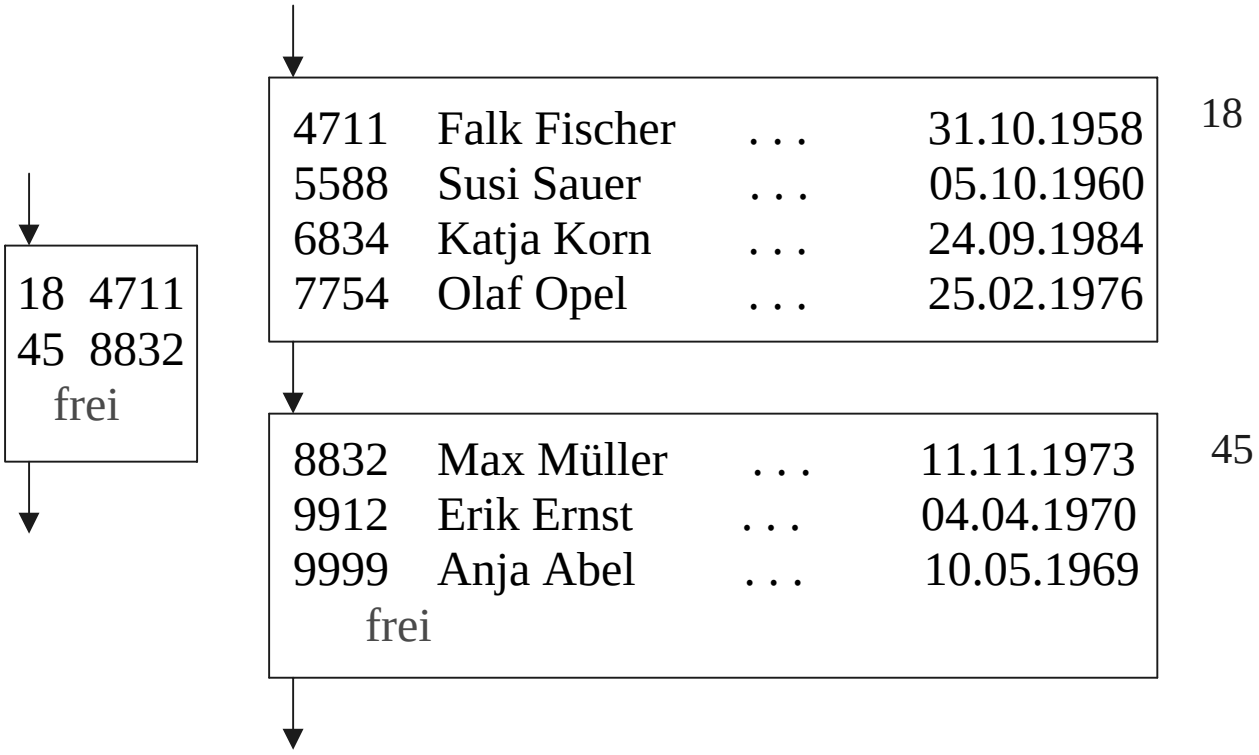
Operationen:

- Suchen
- Einfügen
- Löschen
- Ändern

Anfragen:

```
SELECT  Name
  FROM  Personal
WHERE  PNr = 8832 ;
. . .
WHERE  PNr > 7000 ;
. . .
WHERE  GebDat > 01.01.1975 ;
```

Indexsequentielle Dateiorganisation



- Sortierte Datei mit nichtverzeigerten Sätzen
- Sortierte Datei mit verzeigerten Sätzen

Operationen:

- Suchen
- Einfügen
- Löschen
- Ändern

Anfragen:

```
SELECT  Name
  FROM  Personal
WHERE  PNr = 8832 ;
. . .
WHERE  PNr > 7000 ;
. . .
WHERE  GebDat > 01.01.1975 ;
```

B-Bäume

Grundidee: Index-Hierarchie (Baum);
Bildung eines Index über dem Index, davon wiederum Index usw., bis Index in einem Satz Platz findet (Wurzel des Baumes)

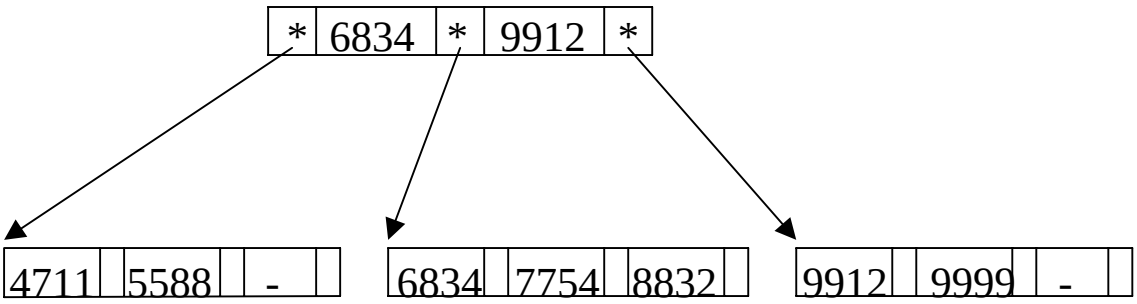
- Binär-Bäume: immer jeweils zwei Söhne
- B-Baum: Mehrwegbaum, d.h. mehrere Söhne
  → für Sekundär-Index geeignet
- B^*-Baum: Datensätze sind Teil des Baumes (in den Blättern)
  → für Primär-Index geeignet
- völlig ausgeglichen:
  Weg von der Wurzel bis zu einem beliebigen Blatt ist gleich lang **und**
  jeder Knoten hat gleich viele Indexeinträge
  → schwierig bei Änderungen

Ein Indexbaum ist ein B-Baum der Ordnung *m*,
wenn gilt:

1. jede Seite hat höchstens *2m* Elemente
2. jede Seite – außer der Wurzel – hat mindestens *m* Elemente
3. jede Seite ist entweder ein Blatt ohne Nachfolger oder hat *i+1* Nachfolger, falls *i* die Anzahl ihrer Elemente ist
4. alle Blattseiten liegen auf der gleichen Stufe

Beispiel

B-Baum der Ordnung *2* mit *2* Stufen



Operationen:

- Suchen
- Einfügen
- Löschen
- Ändern

Anfragen:

```
SELECT  Name
  FROM  Personal
WHERE  PNr = 8832 ;
. . .
WHERE  PNr > 7000 ;
. . .
WHERE  GebDat > 01.01.1975 ;
```

10. Andere Datenmodelle

Das Netzwerk-Datenmodell

- original Netzwerkmodell und –sprache präsentiert 1971 CODASYL Database Task Group (DBTG)

- CODASYL - **Co**nference on **Da**ta **Sy**stems **L**anguages

- (auch CODASYL-Modell oder DBTG-Modell)

- repräsentiert die Datenbank als ein Netzwerk aus
  - Knoten: Satz-Typen (record types)
  - Kanten: Beziehungstypen (set types)

- satzorientiertes Datenmodell

- Einschränkung
  Beziehungstypen sind **binär, 1:N, ohne Attribute**

- Netzwerkschema kann ebenfalls aus ERM bzw. ERD generiert werden

- Schwächen im Bereich der
  - Datenunabhängigkeit
  - Abstraktion von konkreten Speicherstrukturen

Modellierungskonzepte des NDM

- **record**

  zur Speicherung von Objekt-Daten;
  enthält eine Liste zusammengehöriger (Attribut-)Werte
  zur Beschreibung eines Objekts

- **record type**

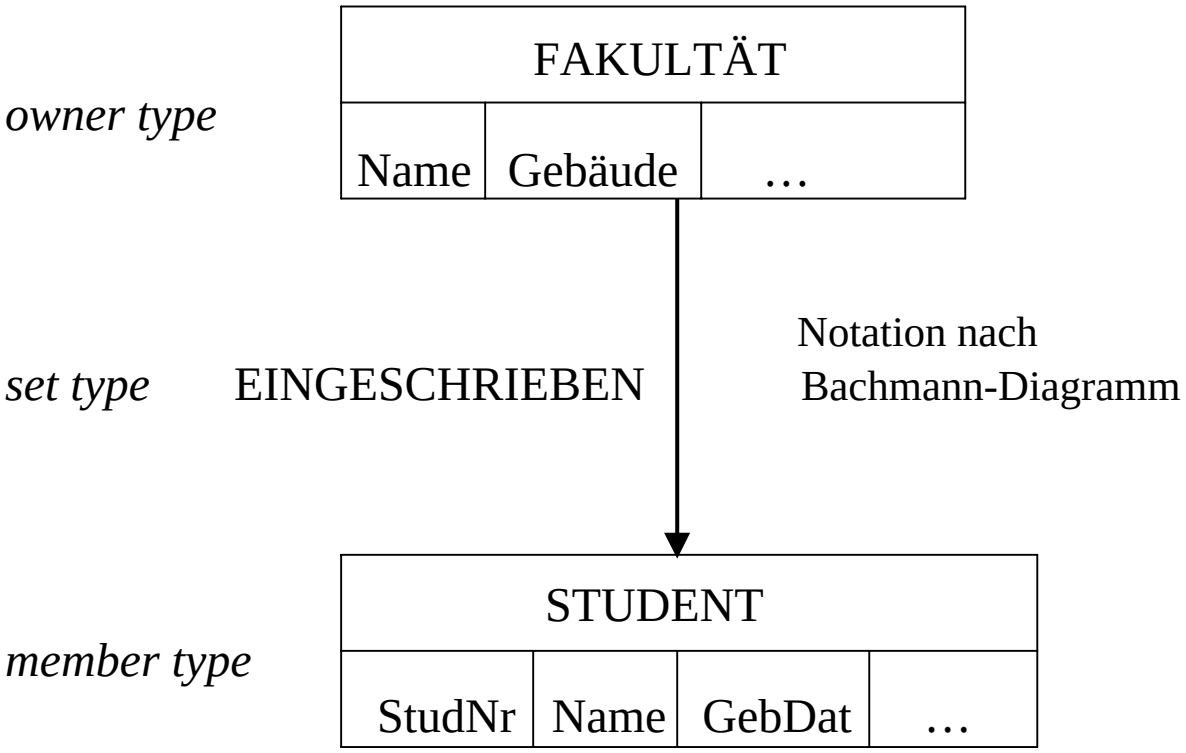
  zur Klassifizierung von *records* ;
  beschreibt die Struktur einer Menge von *records*, die den gleichen Typ von Daten (Objekten) speichern;
  Definition enthält:
  - Name
  - *data items*

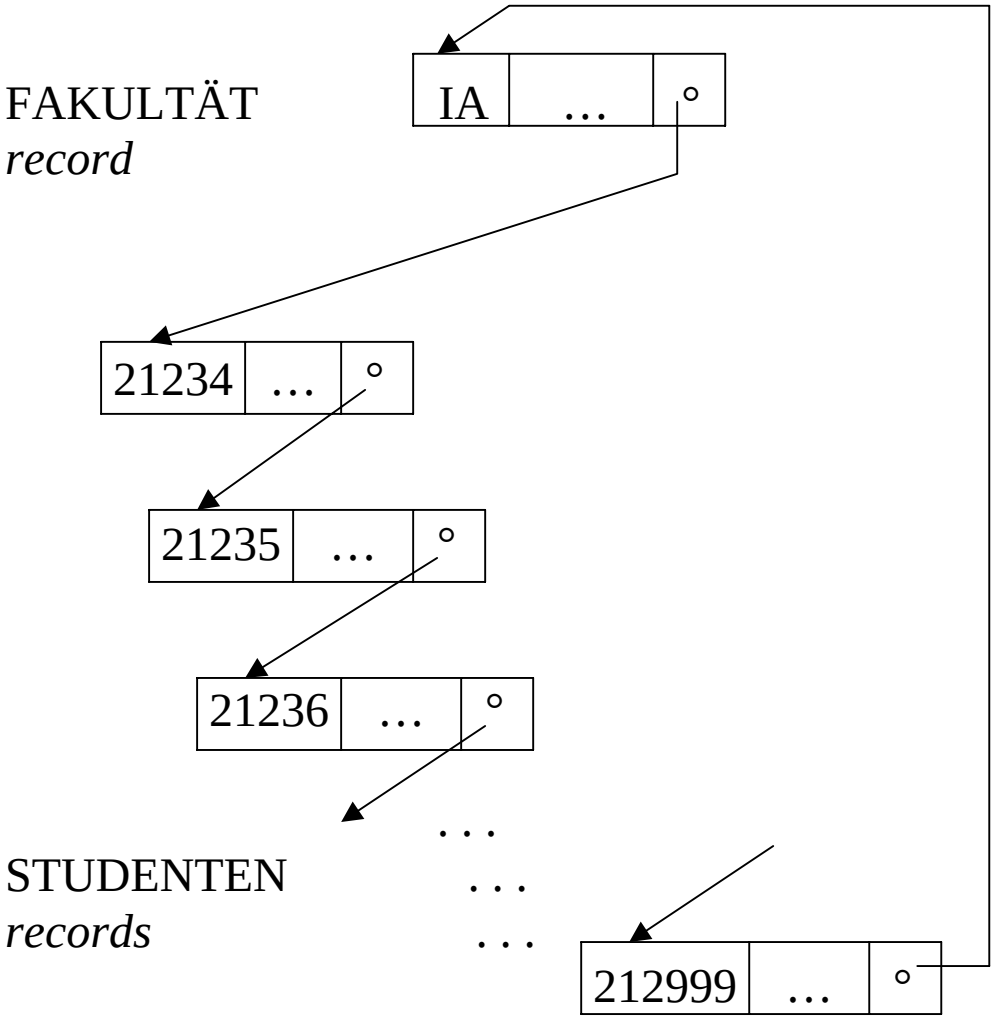
- **data item (attribute)**

  Bestandteil eines *record types*;
  Definition enthält:
  - Name
  - Format (Datentyp)

- **set type (link)**

  binärer, 1:N-Beziehungstyp ohne Attribute
  Definition enthält:
  - Name
  - *owner record type*
  - *member record type*


FAKULTÄT
Name
Gebäude
…
owner type
set type
EINGESCHRIEBEN
Notation nach
Bachmann-Diagramm
STUDENT
StudNr
Name
GebDat
…
member type


FAKULTÄT
record
IA
…
21234
21235
21236
STUDENTEN
records
212999

- **set occurrences (set instances)**

  Ausprägungen eines *set type*.

  Jede Beziehung (Instanz) besteht aus:
  1. einem *owner record* vom *owner record type*
  2. einer Anzahl (null oder mehr) dazugehöriger *member records* vom *member record type*

  Ein *record* vom *member record type* kann nur zu höchstens einer *set occurrence* eines speziellen *set type* gehören. (wegen 1:N)

  Eine *set occurrence* kann über den *owner record* oder über einen der *member records* identifiziert werden.

  Besonderheiten:
  Eine Ausprägung hat ein herausgehobenes Element – den *owner record*.
  Die *member records* sind geordnet.

Speicherung von *set instances*

Z.B. als Ring – ringförmig verbundene Liste

→ Repräsentation symmetrisch

- *type field*
- *pointer fields*
  NEXT-*pointer*
  FIRST-*pointer*
  NIL-*pointer*

Operationen (allgemein)

Prinzipiell müssen folgende Operationen ausführbar sein:

- *owner record*
  → all member records of the set occurrence

- *owner record*
  → 1. , i. , or last member record of the set occurrence
  (if no such record exists, give indication)

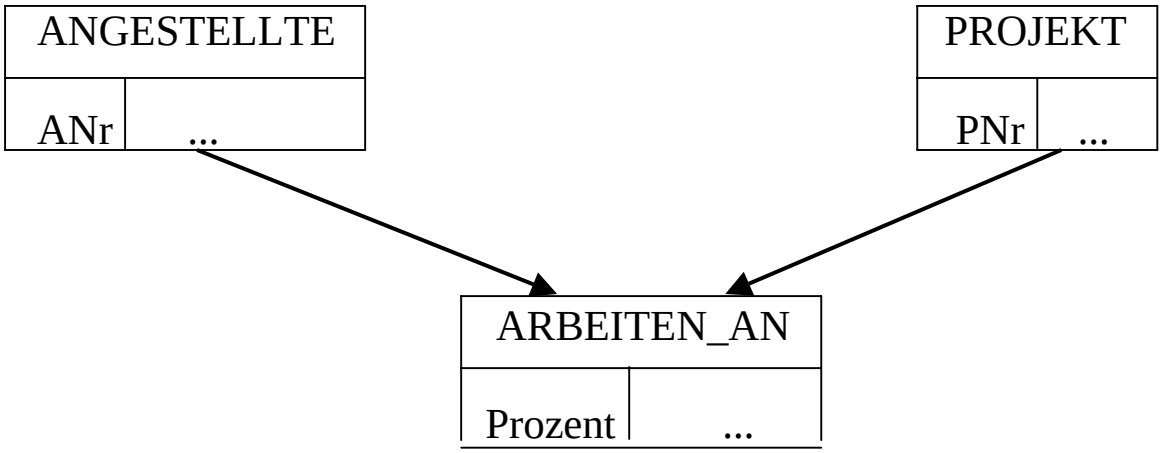
- *member record*
  → next (previous) member record of the set occurrence

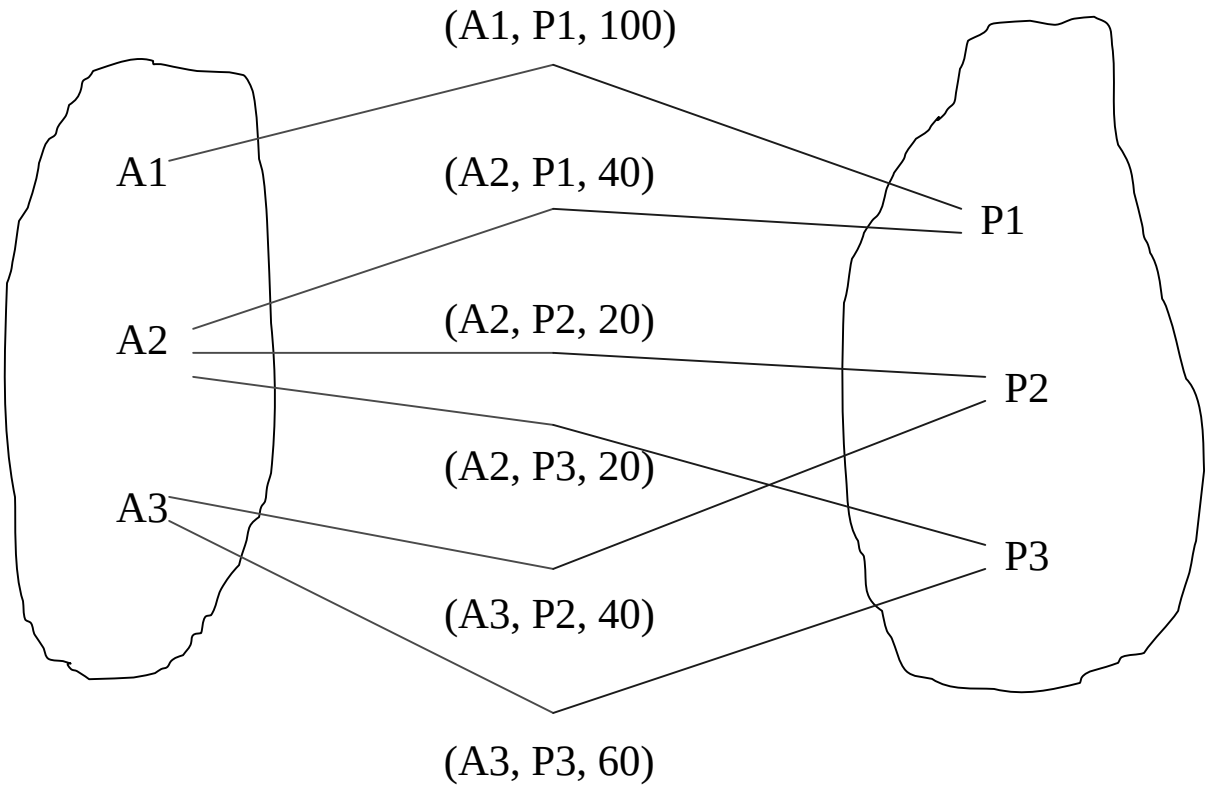
- *member record*
  → owner record of the set occurrence

Darstellung von binären 1:1- und M:N-Beziehungen

- **1:1-Beziehungen**
  Jede *set occurrence* hat nur **einen** *member record.*
  (Nutzer muß das selbst überwachen.)

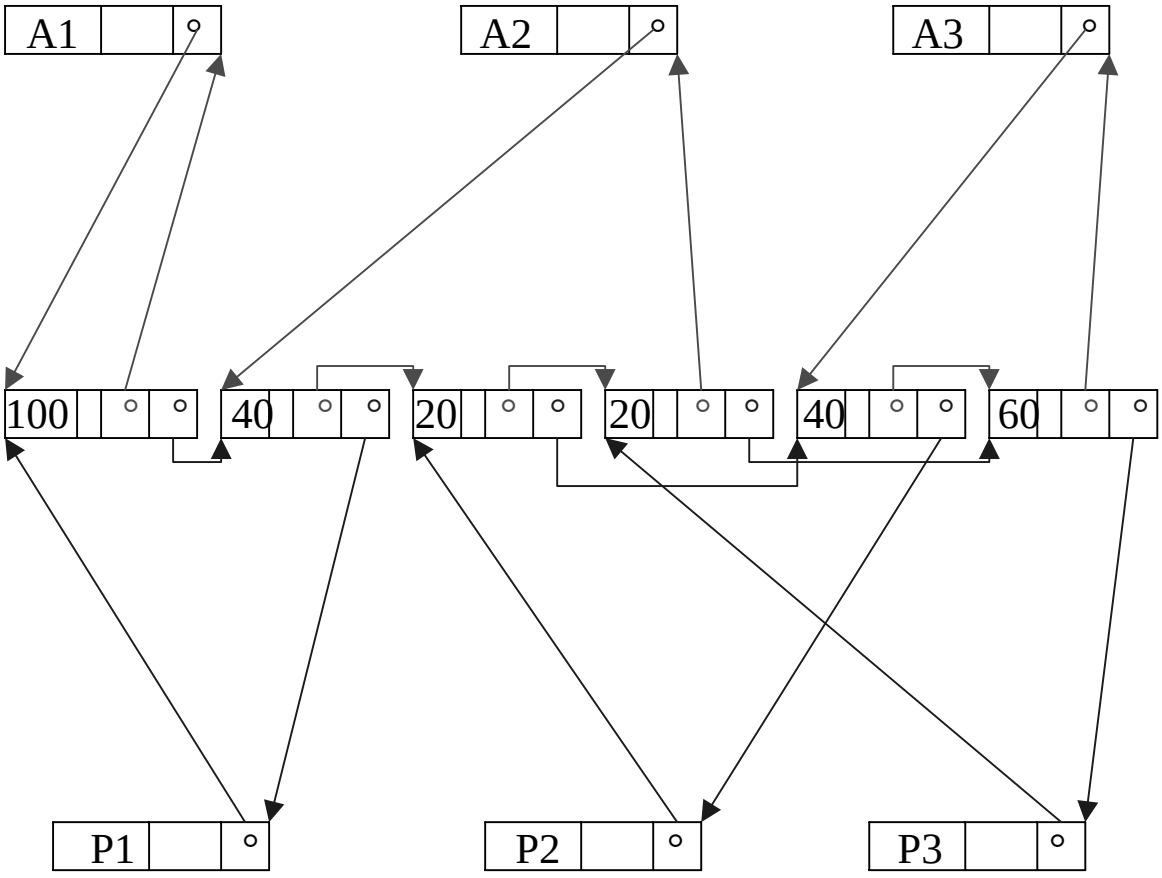
- **M:N-Beziehungen**
  Darstellung mit Hilfe von zwei *set types* und
  linking (or dummy) record type


ANGESTELLTE
ANr
...
PROJEKT
PNr
...
ARBEITEN_AN
Prozent
...


(A1, P1, 100)
A1
(A2, P1, 40)
P1
A2
(A2, P2, 20)
P2
(A2, P3, 20)
A3
P3
(A3, P2, 40)
(A3, P3, 60)

ARBEITEN_AN

ANr	PNr	Prozent
A1	P1	100
A2	P1	40
A2	P2	20
A2	P3	20
A3	P2	40
A3	P3	60



Bedingungen und Restriktionen des NDM

zur Spezifikation von *set*-Mitgliedschaften
1. beim Einfügen
2. für die Lebensdauer in der DB

- **Insertion options (constraints) on sets**
  Einfügen eines neuen *record* eines *member record type* mit **STORE**-Kommando
  ⇒
  1. **AUTOMATIC**
     Der neue *member record* wird automatisch in die *set occurrence* eingebunden.
  2. **MANUAL**
     Der Nutzer kann explizit (manuell) den *record* mit einer *set occurrence* verbinden.
     (mit **CONNECT**-Kommando)

- **Retention options (constraints) on sets**
  Spezifikation der Existenzfähigkeit für *records* in der DB
  ⇒
  1. **OPTIONAL**
     kann existieren ohne *member* einer *occurrence* zu sein
     (für **CONNECT**- , **DISCONNECT**-Kommando)
  2. **MANDATORY**
     muß immer Mitglied einer *occurrence* des *set type* sein
     (kann mit **RECONNECT** auch „umgehängt“ werden)
  3. **FIXED**
     wie MANDATORY und immer in dieser *occurrence*

Das hierarchische Datenmodell

- entwickelt unter Ausnutzung hierarchischer Speicherstrukturen für hierarchische Strukturen der realen Welt

- keine Standardisierung / Festschreibung

- kommerzielle Systeme
  z.B. 1969 von IBM
  IMS – **I**nformation **M**anagement **S**ystem
  (kommerziell erfolgreichstes DBMS der ersten Generation)

- repräsentiert die Datenbank als eine Hierarchie
  d.h.
  ein Netzwerkschema, das ein Wald ist
  $\Rightarrow$ Menge von Bäumen

- Einschränkung
  Links sind alle **in eine Richtung** ausgerichtet

Modellierungskonzepte des HDM

hauptsächlich zwei Struktur-Konzepte:

1. *records*
2. *parent-child-relationships*

- **record**

- **record type**

- **parent-child-relationship type (PCR)**
  binärer, 1:N-Beziehungstyp
  parent record type → *record type* der 1-Seite
  child record type → *record type* der N-Seite

- **occurrence (instance)**
  Jede Instanz (Beziehung) besteht aus:
  1. einem *record* vom *parent record type*
  2. einer Anzahl (null oder mehr) dazugehöriger *records* vom *child record type*

- **hierarchical database schema**
  Menge von *hierarchical schemas*

- **hierarchical schema (hierarchy)**
  Menge von
  - *record types*
  - *parent-child-relationship types*

Eigenschaften von Hierarchien

1. Ein *record type* ist kein *child record type* in einem *PCR type*. → Wurzel

2. Jeder *record type* (außer der Wurzel) ist *child record type* in genau einem *PCR type*

3. Jeder *record type* kann als *parent record type* in mehreren *PCR types* auftreten.

4. Ein *record type*, der nicht als *parent record type* in einem *PCR type* auftritt, heißt Blatt.

5. Ist ein *record type* in mehr als einem *PCR type* als *parent*, dann sind seine *child record types* geordnet (im Diagramm von links nach rechts)

⇒

Baum-Struktur

hierarchical occurrence trees

UND

Notwendigkeit von
virtual parent-child-relationships

Virtuelle Beziehungstypen

- **virtual (or pointer) record type**
  ein *record type* mit der Eigenschaft, daß seine *records* Zeiger auf *records* eines anderen *record type* enthalten
  →
  VC - *virtual child*
  VP - *virtual parent* *in a virtual PCR*

- **virtual parent-child-relationships (VPCR)**
  für folgende Situationen:
  1. M:N-Beziehungstypen
  2. wenn ein *record type* in mehr als einem *PCR type* vorkommt
  3. Beziehungstypen mit mehr als zwei beteiligten *record types*

Speicherung

hierarchisch aufgebaute Dateien
oder
Bäume, in denen die Söhne jeweils sequentiell verzeigert sind

Fazit

Das hierarchische Datenmodell hat die gleichen Modellierungsmöglichkeiten wie das RDM oder NDM unter Beachtung der Besonderheiten und Einschränkungen.

11. Nicht-Standard-Datenbanken

Klassische Datenbanksysteme

sind gut geeignet für

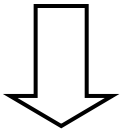
konventionelle Anwendungen
Personalwesen
Buchungs- und Abrechnungswesen
Versicherungswesen
Produktionsplanung (und –steuerung)
Finanz- und Investitionsplanung
u.v.a.m.

mit

- einfacher Semantik der Anwendungswelt
- einfachen, elementaren Datentypen
- flachen Strukturen
- einfachen Beziehungen
- entsprechend einfachen Operationen
- kurzen Bearbeitungszeiten (Transaktionen)
- einfachen Integritätsbedingungen

ABER Grenzen
bezüglich erweiterter **Anforderungen** in

neuen Anwendungsgebieten



Wo

- Entwurf / Konstruktion
- Büroinformation
- wissensbasierte Systeme
- Prozess- und Fertigungssteuerung
- Workflow-Modellierung
- geometrische Datenverarbeitung
- multimediale Anwendungen
- wissenschaftliche und statistische Anwendungen
- Echtzeit-Systeme
- Data Mining / Knowledge Discovery
- …

Anforderungen

- neue Datentypen für komplexe Werte und Objekte
- neue Datenmodelle mit mehr Semantik
- neue Operationen
  → Anfragesprache
- neue Speicherstrukturen für Externspeicher
- komplexe Integritätsbedingungen
- lange Transaktionen
- Zeitachse
- Versionen, Alternativen, Konfigurationen
- Erweiterbarkeit
- Archivierung
- ...

Wie

- Erweiterungen und Verallgemeinerungen relationaler DBsysteme
- Objektrelationale DBs
- Objektorientierte DBs
- Erweiterbare DBs
  (DBkern, Baukasten, Generator)
- Deduktive DBsysteme
- Aktive DBsysteme
- Temporale DBsysteme
  Echtzeit-DBsysteme
- GIS
- MMDBsysteme
- Workflow-Management-Systeme
- Data Warehouse
- Verteilte Datenbanken
- Föderierte Datenbanken
- Parallele DBsysteme