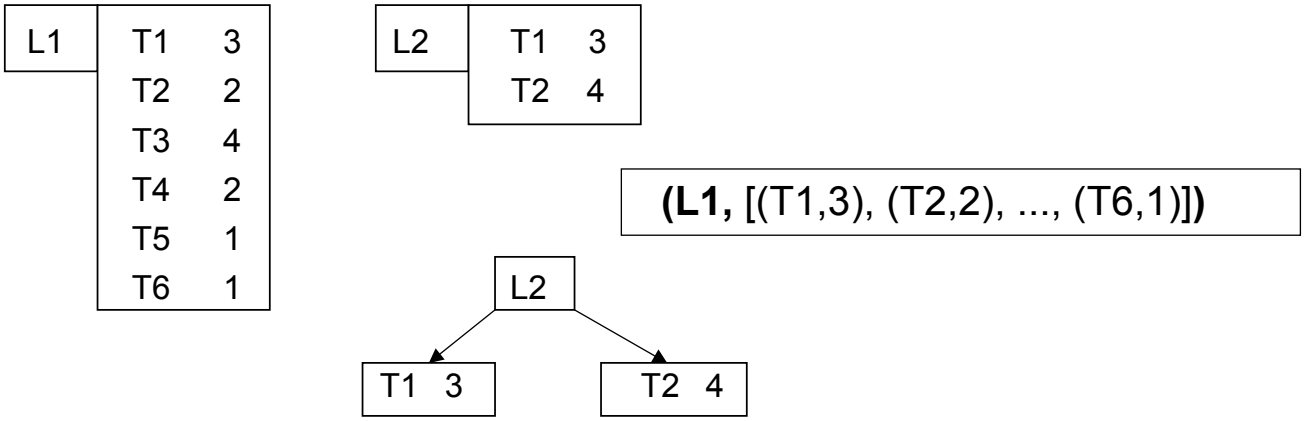
Kap. 9 Das hierarchische Datenmodell

Kap. 9.1 Grundbegriffe

Anlehnung am IMS

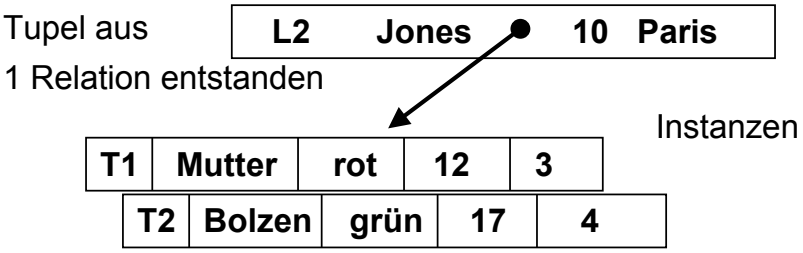
Information **M**anagement **S**ystem (IBM)

DB-Modell: Hierarchien, ähnlich zu nicht-normalisierten Relationen:



d.h. frühere LT-Relation durch Geflecht ersetzt, jetzt Hinzufügung der vollen Info über Lieferanten u. Teile!

Erweiterte Grundstruktur um Attribute:



T# TNAME TFARBE GEW Anzahl **Schema**

i.e. Schema u. Tupel aus 2 Relationen

(TEILE und LT)

Motivation: wie Gründe für Nicht-Normalisierung.

Info über Lieferanten u. Teile semantisch zusammengehörig u. meist gemeinsam benötigt.

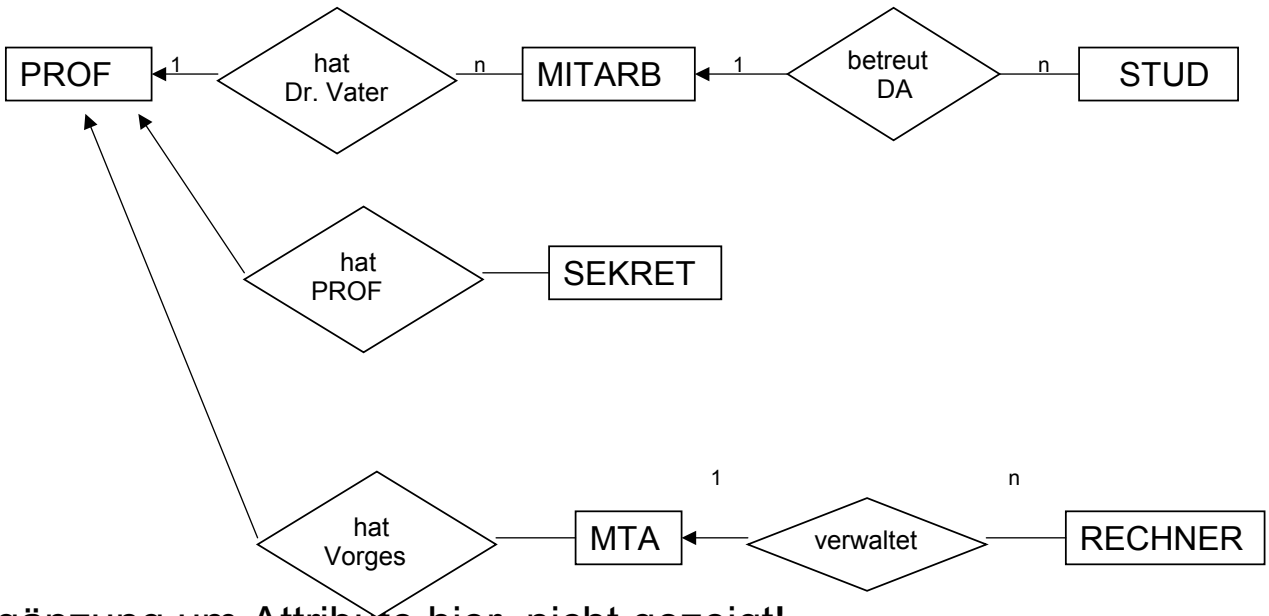
⇒ Zusammenfassung zu 1 Objekt = Satz

⇒ Info gemeinsam gespeichert, d.h. meistens effizient zugreifbar

⇒ Ausnahmefälle teuer, komplizierte Programmierung : Berücksichtigung der Zugriffsstruktur der Daten in den Programmen??

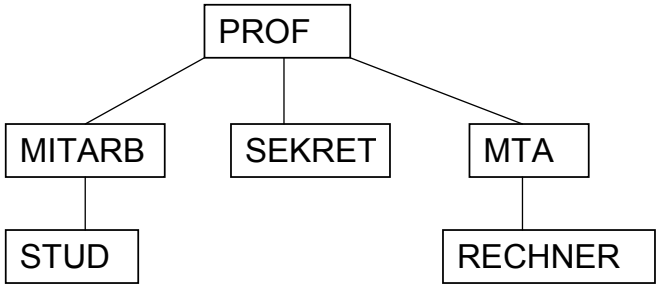
Systematische Modellierung:

natürlich für 1:n Beziehungen:

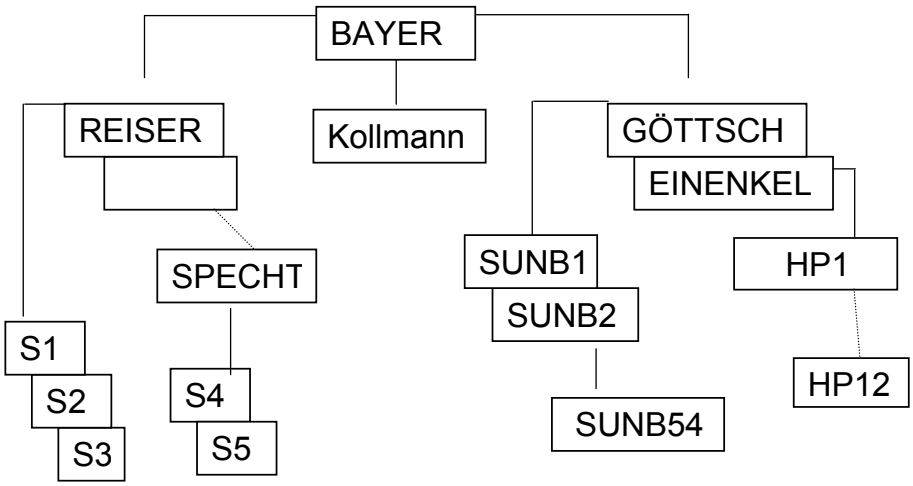


Ergänzung um Attribute hier nicht gezeigt!

Hierarchisches Schema: Bedeutung der Relationships wird nicht explizit angegeben, muß dem Verarbeitungsprogramm bekannt sein, abhängig von der Reihenfolge der Speicherung

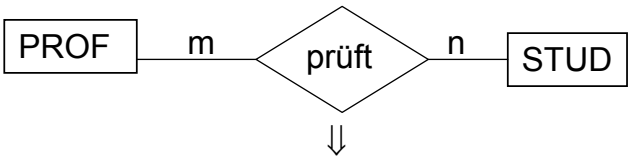


Hierarchische Instanzen mit Wiederholungen pro Ebene:



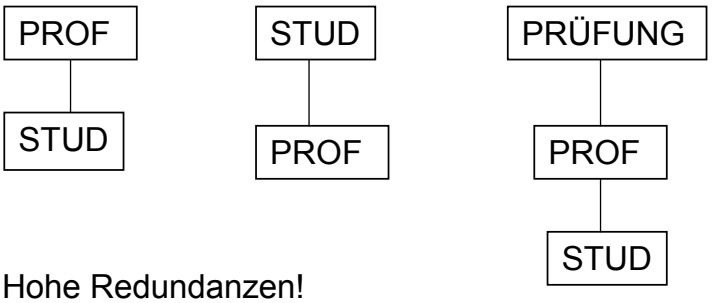
Hinweis: Beziehungsarten verschwunden, steckt in Programmlogik.

Modellierung von n:m Beziehungen



PRÜFUNG (Ort, Datum, Zeit, Fach, Note, P-Name, M#)

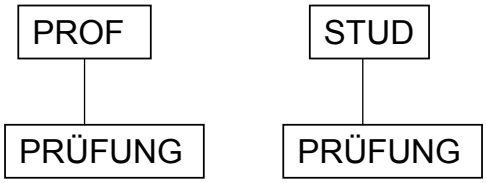
Keine natürliche Hierarchie!



Hohe Redundanzen!

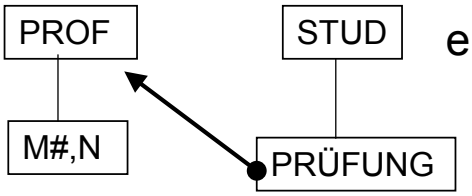
Relationales Schema:

Hierarchisches Schema?



mit voller Information über PRÜFUNG zu viel Redundanz,
alles 2 x gespeichert

Hierarchisches Schema mit „Fremdschlüssel“



enthält M# als Attribut

enthält P-Name als Attribut

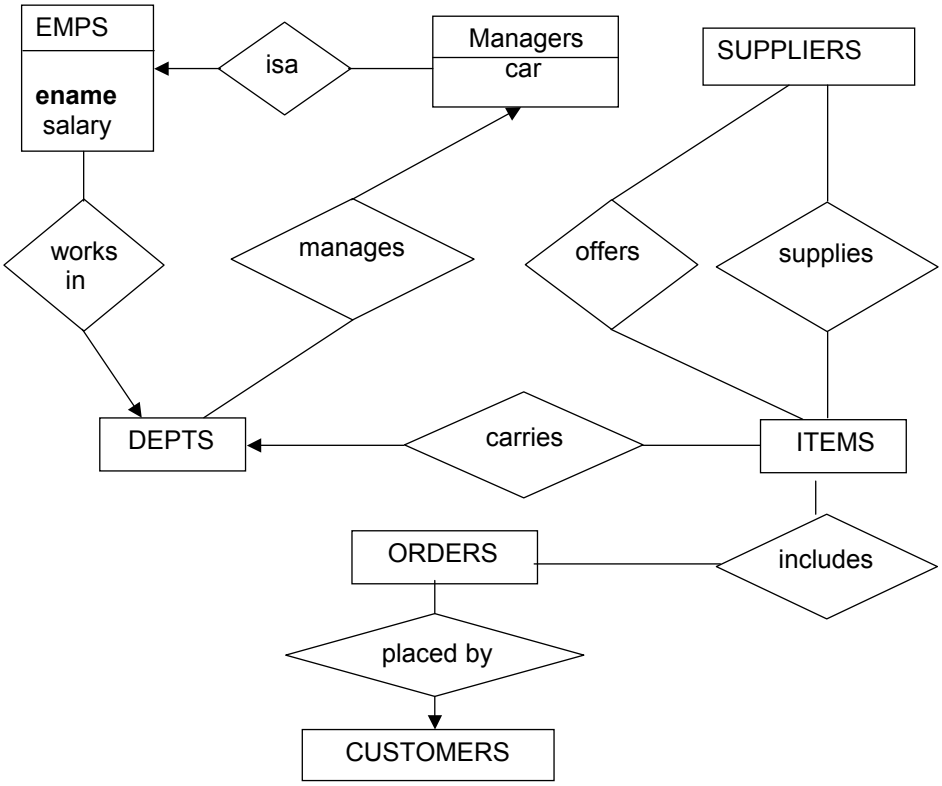
gut für Praxis: Prof hat Prüfungsliste
Stud hat Prüfungsliste

„Noten von Maier?“

Algorithmische Schritte:

1. suche Maier über Wurzel STUD;
2. finde alle Prüfungen von Maier, enthalten P-Name
3. suche P-Name über PROF
4. suche M#, Note von P-Name

Schema ohne Attribute:



Konvertierung E/R zu Hierarchie

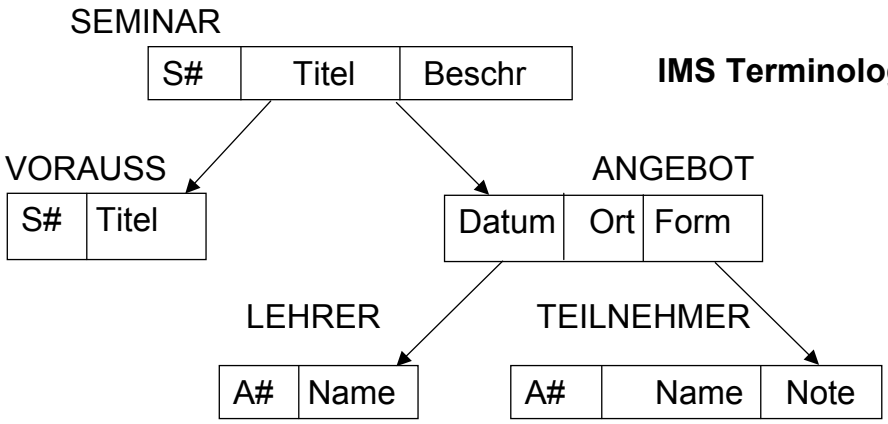
```
procedure    build-tree (n)
  begin  <markiere n>;
      for each Kante m ⟶ n in E/R Schema  do
      begin  <n wird Vaterknoten, ziehe Kante von n zu Sohnknoten m,
             m-Knoten werden bei Instanzen wiederholt>;
          if      ¬  <markiert m> then
                     build-tree (m)

          fi
      end
  end
```


Hauptprogramm: wähle Entität n mit vielen auf n zeigenden Links, ohne Links weg von n zu nicht markierten Knoten:

```
begin  <lösche alle Markierungen>
  while ¬<alle Knoten markiert> do
             wähle unmarkiertes n;
             build-tree (n)
          od
end
```

Kap. 9.2 Art-Vereinbarungen in IMS

Betriebsinterne Seminare“



IMS Terminologie: Tupel = ***Segment***
Attribut = ***Feld, field***

Hinweis: *hier sind nur die Feldnamen angegeben, die Datentypen fehlen noch, siehe später!*

Art von PDR (physical database record) als regulärer Ausdruck:

S V* (A L* T*) *

Art von PDB (physical data base) :

( )*

z.B. s v_1 v_2 a_1 l_{11} t_{11} t_{12} a_2 l_{21} t_{21} t_{22} t_{23} t_{24} a_3 t_{31} t_{32} . . .

d.h. Struktur von Satz wie regulärer Ausdruck,
in DB sequentiell gespeichert wie Wort des Sprachschatzes

Vereinbarungen:

- Segmente in Vorordnung des Baumes
- Segmentbezeichnungen (Indikatoren) bis 8 Zeichen mit Vorgänger (PARENT), Segment Länge in Bytes
- FIELD – Bezeichnungen mit Länge u. Startbyte innerhalb des Segments
- SQUENCE-FIELD: 1. Komponente im Segment, im Normalfall vorhanden, kein Schlüssel! (identische Werte erlaubt) (M für **m**ultiple Values)

Beispiel:

```
DBD     NAME = Sembank
SEGM    NAME = Seminar, BYTES = 256
  FIELD NAME = (S#, SEQ), BYTES = 3,
               START = 1
  FIELD NAME = Titel, BYTES = 33, START = 4
  FIELD NAME = BESCHR, Bytes = 220,
               START = 37
SEGM    Name = Vorauss, PARENT = Seminar,
               BYTES = 36
  FIELD Name = (S#, SEQ), BYTES = 3,
               START = 1
  FIELD Name = Titel, BYTES = 33, START = 4
```

```
SEGM    Name =  Angeb, PARENT = Seminar,
                BYTES . . .
 FIELD  Name =  (Datum, SEQ, M), BYTES = 6,
                START = 1
 FIELD  Name =  Ort . . .
SEGM    Name =  Lehrer, RARENT = Angeb,
                BYTES = . . .
```

Hinweis: keine Typisierung, alles in Bytes angegeben, Byte ist einziger Datentyp

War massives Jahr 2000 Problem: Wie findet man Segm bzw. Field mit Bedeutung "Jahr"? Erfordert Änderung der Programme!!!

Hinweis: Länge in Bytes **und** Start-Bytes angegeben, weil FIELDS im selben Segment sich überlappen können.

```
z.B.  SEGM     Datum     BYTES = 6 START = 1
        Field  Tag       BYTES = 2 START = 1
        Field  Monat     BYTES = 2 START = 3
        Field  Jahr      BYTES = 2 START = 5
```

⇒ Änderung von Jahr auf 4 Bytes schlägt auf Struktur der DB u. Programme durch.

Redundanz: im obigen Beispiel:

z.B S#, Titel
A#, Name
} i.a. vielfach vorhanden

Herausfaktorisieren im Prinzip, **aber** pro Manipulationsanweisung nur Bezugnahme auf **eine** DB möglich:

GU

GN

GNP . . .