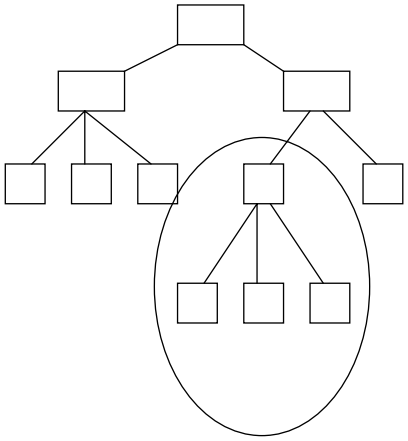
Kapitel 9.3 Datensubmodelle

Für Benutzer nur individualisierte Ausschnitte aus DB sichtbar, z. B. Werksarzt, Chef, selbst, Betriebsrat, ...

Datenschutz: bei Personaldaten soweit für Aufgabenbereich erforderlich

Logical DB: i.e. Auschnitt aus PDB:



Regel: *mit einem Knoten werden alle Nachfolger abgeschnitten.*

Regel: ***alle*** Segmente einer bestimmten Art zugänglich

oder ***keine***:

insert, **update**, **read**, **delete** (mit Nachfolgern)

Sonderfall: **delete** eines Segments mit unzugänglichen Nachfolgern verboten

$\Rightarrow$ covert channel!!

Regel: kein inhaltsabhängiges Ausblenden, nur abhängig von Schema!
in relationalen Systemen viel allgemeiner gehandhabt.

"Vereinbarung" einer LDB:

Inhalte der LDB: ***"sensitive** segments*"

Beschreibung über PCB: **p**rogram **c**ommunication **b**lock

zur Verbindung zwischen DBS und Anwendungs-Programm

Beispiel: für Sekretärin, die Anmeldung für Seminare aufnimmt:

```
1  PCB        TYPE = DB, DBDNAME = Sembank
2  SENSEG  NAME = Seminar, PROCOPT = G
3  SENSEG  NAME = Angeb, PARENT = Seminar,
                          PROCOPT = G
4  SENSEG  NAME = Teilnehmer, PARENT = Angeb,
                          PROCOPT = G, I, R, D
```


PROCOPT ≈ processing option (Berechtigung)

- G : GET
- L : Load (Initialisierung)
- I : Insert
- R : Replace
- D : Delete
- K : zum Ausblenden von Zwischenknoten in LDB

Operationen: i.e. Prozeduraufrufe aus Wirtssprache:

GET UNIQUE	≈	**GU**
GET NEXT	≈	**GN**
GET NEXT WITHIN PARENT	≈	**GNP**
INSERT	≈	**ISRT**
DELETE	≈	**DLET**
REPLACE	≈	**REPL**

GET HOLD UNIQUE
GET HOLD NEXT
GHNP
} mit exklusiver Sperre auf Segment-Ebene

Kap. 9.4 Datenmanipulation in IMS

1. Leseoperationen: für segmentweise (~ tupelweise) Verarbeitung von IMS - Datenbanken

GET UNIQUE ~ GU mit SSA

(SSA = **s**egment **s**earch **a**rgument für betroffene Segmente
~ **where**-Klausel in SQL)

2. Lesen mit Sperren: typisch für anschl. update

GHU GHN GHNP

3. Änderungsoperationen:

ISRT	SEQ-field bestimmt Ordnung
DLET	
REPL	SEQ-field darf nicht geändert werden

Beipiel 1: *"Teilnehmer an VDBS in Stuttgart mit Note ≤ 2"*

GU SEMINAR (TITEL = 'VDBS')
ANGEBOT (ORT = 'Stuttgart')
TEILNEHMER (Note ≤ 2)

⇒ Pfad durch Hierarchie!
liest Segment der Art TEILNEHMER,
liefert *erstes* Segment mit spezifizierten Eigenschaften

W: GNP TEILNEHMER (Note ≤ 2)
... Verarbeitung in Host-Programm,
d. h. Cobol, PL/I, Fortran.
Erfolgloses GNP durch Statusanzeige!
got to *W*

Beispiel 2: *"Liste aller Lehrer eines bestimmten Seminars"*

GU SEMINAR ... Segment geliefert, aber
nicht benötigt
V: GNP ANGEBOT
W: GNP LEHRER geht über PDB-record
Grenzen (Teilnehmer) hinweg
... Statusprüfung und Verarbeitung in
Host-Sprache ...
goto *W*
goto *V*

Beispiel 3: *"Änderung eines Seminarlehrers"*

```
GHU  SEMINAR   (TITEL = 'VDBS')
     ANGEBOT  (ORT = 'STUTTGART')
     LEHRER     {Navigationsposition durch
                GHU festgehalten;}

     Name:= 'Müller' im E/A-Bereich über PL/I oder COBOL,
            wie Host-Variable, siehe Kap. 8
     REPL
```

Anm: SEQ-Field (bestimmt physische Position) darf durch REPL nicht geändert werden, nur durch DLET oder ISRT

Beispiel 4: *"Ausfall eines Seminars"*

```
GHU  SEMINAR   (TITEL = 'VDBS')
     ANGEBOT (ORT = 'Hamburg'
                ∧ DATUM = 081003);

DLET {löscht alle Kind Segmente rekursiv, i.e.
      Unterbaum}
```

Anmerkung: Beachte die Trennung zwischen

- Positionierung auf ein bestimmtes Segment, Paradigma der Bandverarbeitung
- Ausführung der eigentlichen Änderungsoperation

Beispiel 5: *"Teilnehmer einschr. in 'VDBS' in Hamburg am 08.10.03"*

```
STUDENT
  A #   :=      16573
  NAME  :=      'Jung'
```

STUDENT ist eine Struktur, im E/A-Bereich im Programm in Host-Sprache vorbereitet

{hier kein GHU nötig, weil Segment noch nicht existiert}

```
ISRT   SEMINAR (TITEL = 'VDBS')
       ANGEBOT    (ORT = 'Hamburg'  ∧ DATUM = 081003)
                  {Bedingungen auf Feldern von Segment Angebot}
       STUDENT
```

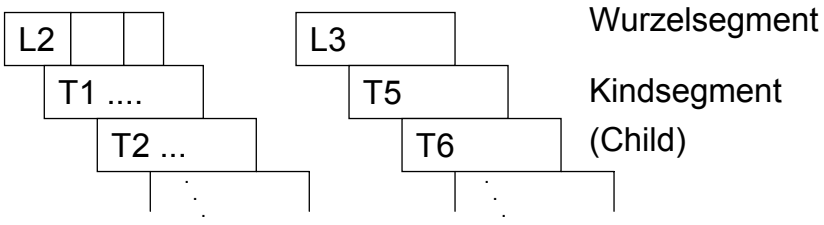
{Einfügung an richtiger Stelle entspricht SEQ-Field A# in Student}

Hinweis: Knoten, von denen im E/R Schema mehrere Links ausgehen, tauchen in Hierarchie mehrfach auf, i.e. Relationen für Relationships.

Logische Struktur und physische Speicherung

Jede Instantiierung der Hierarchie = 1 Satz, record*, physical DB-record*

IMS-DB = Folge von Sätzen



Speicherung: sequentiell

	L2	T1	T2	L3	T5	T6 ...

i.e. Baum der Records in Vorordnung, geeignet für Bandverarb, sofern Zugriffsstruktur des Anwendungsprogramms passend ist.

Relationship *"liefert"* wird implizit repräsentiert durch physische Positionierung vom T1 nach L2 bis zu nächstem Satz-Art Wechsel, hier von T2 zu L3. Satzart wird bei Operationen **GN** und GNP angegeben!

Query-Sprache: Prozedural mit Schleifen!

Beispiel 1: *"Finde T# für Teile, die von L2 geliefert werden"*

1. **Get unique** Lieferant mit L# = 'L2'
   {impliziert Positionierung des seqentiellen Speichers, d.h. des Bandes, heute über Index};
2. ***N*** : **Get next** *{nächstes physisches Segment}*
3. **if** Segmentart ≠ Teile **then** exit;
4. print (T#) *{i.a. bearbeite Teile-Segment}*
5. **go to *N***

Beispiel 2: *"Finde L# für Lieferanten, die T2 liefern"*

Beachte: Symmetrie der 2 natürlich-sprachlichen Queries in Beispiel 1 und Beispiel 2

∀ Lieferantensätze prüfe
  if ∃ Teil mit T# = 'T2'
    then bearbeite Lieferanten-Segment

⇒ *2 Suchschleifen wegen Dateiaufbau in Beispiel 2,*
nur 1 Schleife in Beispiel 1

L $\underbrace{\text{T ... T}}_{\geq 0}$ **L** T ... T2 ... **L** ... **EoF**


```
1. Anfangsposition der Satz-Datei
2. Get next              {suche nächstes Lieferanten-Segment}
3. if EoF then exit;
4. elseif Satzart = Lieferant
5. then  HL := Lieferantensegment;
6.       Get next    {suche nächstes Teile-Segment};
         if EoF then exit
         elseif Satzart = Lieferant then go to 5
         else {Satzart = Teil}
          if T# = 'T2' then      bearbeite HL;
                                 go to 2
          else go to 6        { innere Schleife}
7. else go to 2               { äußere Schleife}
```

Vergleiche SQL:

Beispiel 1: **select** T# **from** LT
where L# = 'L2'

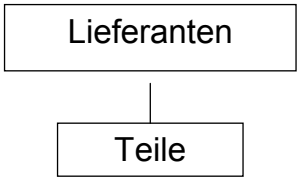
Beispiel 2: **select** L# **from** LT
where T# = 'T2'

Anm: *Symmetrie in Query-Formulierung wie bei natürlichsprachlicher Formulierung*

Warnung: *Auswertungseffizienz abhängig von Primärschlüssel-Wahl!*

Warum??

IMS



Unsymmetrie in DB-Schema
$\Rightarrow$ Unsymmetrie im Programm

Rel.-Schema: LT (L#, T#, A)

keine Unsymmetrie
$\Rightarrow$ Data-Independence! (von Queries)

Manipulation in IMS: *ähnliche Probleme, wie bei unnormalisierten Relationen:*

1. Information über neues Teil nur repräsentierbar, wenn es auch geliefert wird (child abhängig von parent-segment)
2. Mit letztem Lieferanten verschwindet auch Information über Teil
3. Information über Teil mehrfach gespeichert, redundant mit jedem Lieferanten
   $\Rightarrow$ Änderungen an vielen Stellen, durchsuchen der ganzen DB

Zusammenfassung IMS Begriffe:

PDB (physical database) =
geordnete Menge von Sätzen = 1 IMS-DB

PDR (physical DB-record) =
Folge von meist verschiedenartigen Segmenten,
i.e. Linearisierung einer Baumstruktur,
rekonstruierbar durch Segmentart.

Segment = Verbund mit fester Struktur u. fester Länge!!

Field = Komponente in Segment, feste Länge, mit frei gewähltem Selektor, nur einfache Arten!

Hinweis: Segmente in Satz (außer Wurzel) beliebig oft wiederholbar (≥ 0). Segmentname ist kein Selektor, d. h. kein Identifikator, sondern nur Segmentindikator (für Art)

Segment Identifizierung:

a) über Art-Indikator relativ zu anderen Segmenten und **next <Segmentart>**

b)Reihenfolge der Segmente für **next** nicht zufällig oder nach insert-Ordnung, sondern durch *"sequence-field"* SEQ

Gründe für Modell:

1. Für viele Anwendungen sehr natürliche Modellierung
2. Auf sequentiellen Medien (Bändern) gut implementierbar.