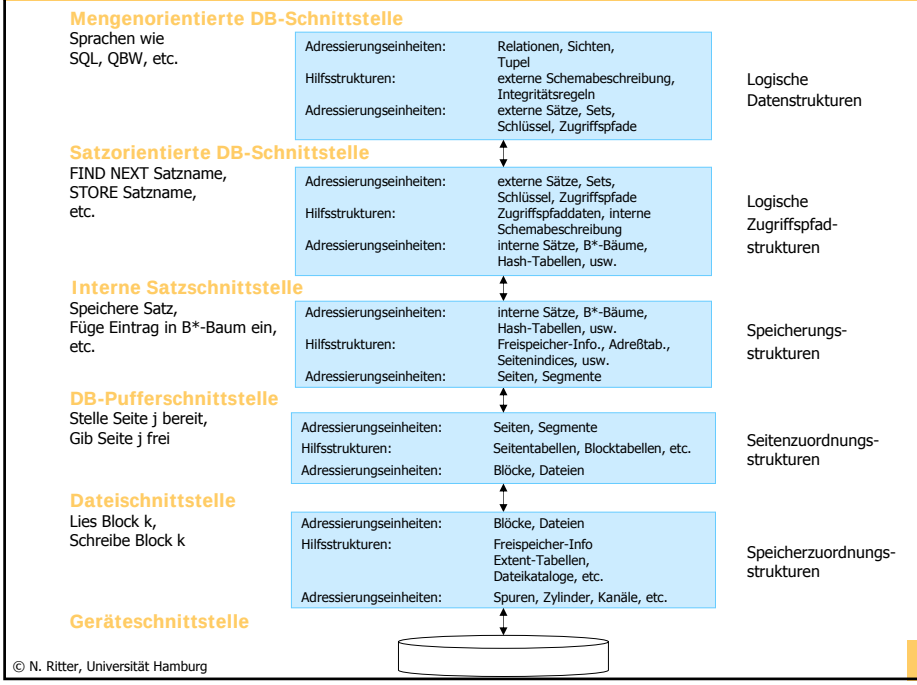
Implementierung von Datenbanksystemen

Kapitel 4: Speicherungsstrukturen

Kapitel 4: Speicherungsstrukturen

- Ziel: Entwurf von
  - Speicherungsstrukturen für Sätze und komplexe Objekte
  - Hilfsstrukturen wie Freispeicherverwaltung, Adressierung usw.
- Freispeicherverwaltung
  - im Segment
  - in der Seite
- Externspeicherbasierte Satzadressierung
  - TID
  - Zuordnungstabelle
  - Indexierung von Tabellen (Satzmengen)
- Hauptspeicherbasierte Satzadressierung
  - Klassifikation der Lösungskonzepte
  - Pointer-Swizzling-Verfahren
- Abbildung von Sätzen
  - feste/variable Felder
  - Partitionierung
- Speicherungsstrukturen für komplexe Objekte
  - Listen- und Mengenkonstruktoren
  - Tupelkonstruktoren
- Darstellung langer Felder
  - Abbildung auf Segmente
  - Zugriffsstruktur des Objekts



Speicherungsstrukturen

- Operationen an der oberen Schnittstelle:

```
insert    <record> at <location> with <database-key>
retrieve  <record> with <database-key>
add       <entry> to <B*-tree>
retrieve  <address-list> from <B*-tree> for <value>
```

- Abbildungsfunktion:

```
Satz-Identifikator   ↔ address
Attributwert         ↔ record-id.list
Satz-Identifikator   ↔ record-id.list
Adresse              ↔ {occupied, free}
```

- Operationen an der unteren Schnittstelle:

FIX P_i, FIX P_j, UNFIX P_j, FIX P_k, UNFIX P_i, ...

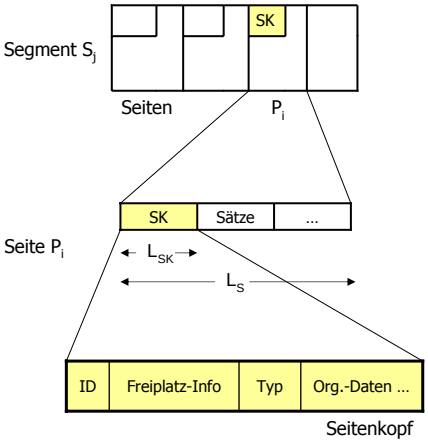
- Eigenschaften der oberen Schnittstelle:
  - Nicht-flüchtiger Speicher mit Adressierungshilfen
  - Freispeicherverwaltung
  - Adressierungsverfahren von und zwischen physischen Sätzen
  - Zugriffspfade zur Realisierung von Inhaltsadressierbarkeit

Record-Manager

Zugriffspfad-verwaltung

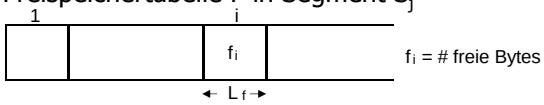
Freispeicherverwaltung

- Freispeicherverwaltung (FPA) für
  - Externspeicher
    - Allokation von Dateien
  - Segmente
    - Allokation von internen Sätzen
  - Seiten
    - Verwaltung von belegten/freien Einträgen
- Für alle Seiten eines Segmentes:
  - Einfügen/Ändern
    → Suche nach n freien Bytes
  - Löschen/Ändern
    → Freigabe oder Markierung von Speicherplatz
  - Allgemein:
    Suche, Belegung und Freigabe von Speicherplatz in S_j



Freispeicherverwaltung in Segmenten

- Freispeichertabelle F in Segment S_j



f_i = # freie Bytes

- Größe von F
  - Einträge pro Seite der Länge L_S: $k = \left\lfloor \frac{L_s - L_{SK}}{L_f} \right\rfloor$
  - Seiten für F: $n = \left\lceil \frac{s}{k} \right\rceil$

    mit s = #Seiten im Segment
- Lage von F
  - Segmentanfang
  - Segmentende
  - äquidistante Verteilung auf die Seiten i * k+1 (i=0,1,2,...)

Freispeicherverwaltung in Segmenten

- Art der FPA:
  - exakt:
    - typisch: L_f = 2 Bytes
  - unscharf:

L_f = 1 Byte

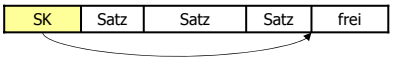
Einheiten von f_i: Vielfache von $\left\lceil \frac{L_S}{256} \right\rceil$

bei L_S = 4 KB: 16 Bytes

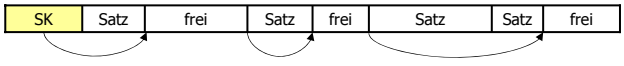
L_f = 2 Bits
'00': weniger als 25% sind belegt
'10': zwischen 50% und 75% sind belegt
...

Freispeicherverwaltung in Seiten

- FPA innerhalb Seite P_i
  - exaktes f_i in SK (Freiplatz-Info)
  - zusammenhängende Verwaltung
    - Verschiebungen!



  - unzusammenhängende Verwaltung
    - Freispeicherkette (best-fit / first-fit)



Kapitel 4: Speicherungsstrukturen

- Ziel: Entwurf von
  - Speicherungsstrukturen für Sätze und komplexe Objekte
  - Hilfsstrukturen wie Freispeicherverwaltung, Adressierung usw.
- Freispeicherverwaltung
  - im Segment
  - in der Seite
- Externspeicherbasierte Satzadressierung
  - TID
  - Zuordnungstabelle
  - Indexierung von Tabellen (Satzmengen)
- Hauptspeicherbasierte Satzadressierung
  - Klassifikation der Lösungskonzepte
  - Pointer-Swizzling-Verfahren
- Abbildung von Sätzen
  - feste/variable Felder
  - Partitionierung
- Speicherungsstrukturen für komplexe Objekte
  - Listen- und Mengenkonstruktoren
  - Tupelkonstruktoren
- Darstellung langer Felder
  - Abbildung auf Segmente
  - Zugriffsstruktur des Objekts

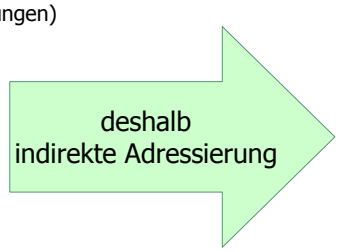
Externspeicherbasierte Satzadressierung

- Problemstellung
  - langfristige Speicherung der Datensätze
  - Vermeiden von „Technologieabhängigkeiten"
  - Unterstützung von Migration u. a.
- Allgemeine Form einer Satzadresse

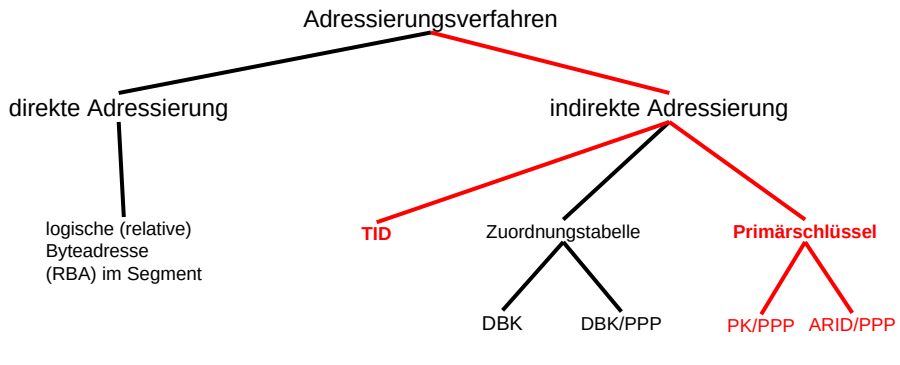
  - DBID: Datenbank-Identifikator
  - SID: Segment-Identifikator
  - TID: Tupel-Identifikator
  - RID: Relationen-Identifikator
  - Relationen vollständig in einem Segment gespeichert: TID (DBID, SID im DB-Katalog)
  - Relationen in mehreren Segmenten: SID, TID

Externspeicherbasierte Satzadressierung

- Ziele der Adressierungstechnik:
  - schneller, möglichst direkter Satzzugriff
  - hinreichend stabil gegen geringfügige Verschiebungen (Verschiebungen innerhalb einer Seite ohne Auswirkungen)
  - seltene oder keine Reorganisationen
- Direkte Adressierung in Segmenten
  - logisch zusammenhängender Adressraum
  - direkte Adressierung (logische Byte-Adresse, RBA)
  - instabil bei Verschiebungen

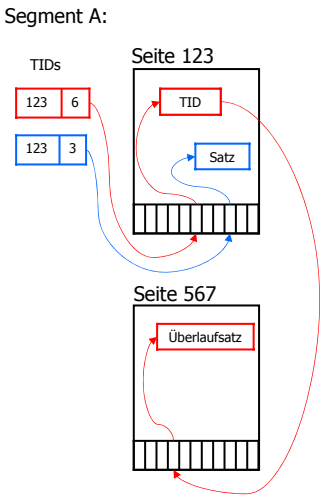


Techniken zur externspeicherbasierten Satzadressierung



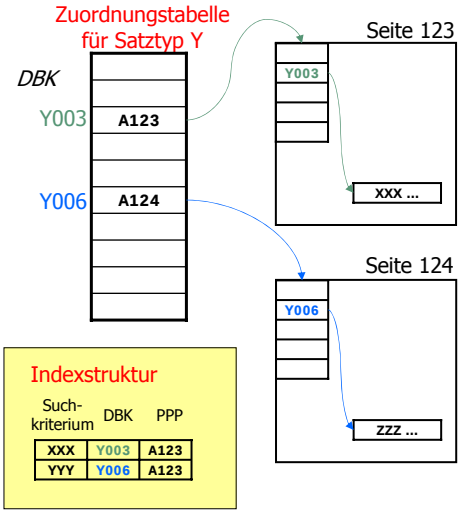
Satzadressierung: TID-Konzept

- TID (tuple identifier) besteht aus zwei Komponenten:
  - Seitennummer (3 B)
  - relative Indexposition innerhalb der Seite (1 B)
  - dient zur Adressierung in einem Segment (z. B. SID = A)
- Migration eines Satzes in andere Seite
  - ohne TID-Änderung möglich
  - Einrichten eines Stellvertreter-TID in Primärseite
  - Überlaufkette: Länge <= 1



Satzadressierung über Zuordnungstabelle

- Jeder Satz erhält eindeutigen logischen Identifikator:
  - Datenbankschlüssel (DBK)
  - Vergabe der DBK erfolgt i.A. durch DBVS
  - systeminterne Verweise auf Sätze erfolgen ausschließlich über den DBK
- Zuordnungstabelle enthält pro DBK zugehörigen PP
  - SID (1B)
  - Seitennummer (3B)
- Hybrides Verfahren:
  - Verwendung von 'probable page pointers' (PPP) in Zugriffspfaden erspart u. U. Zugriff auf Zuordnungstabelle

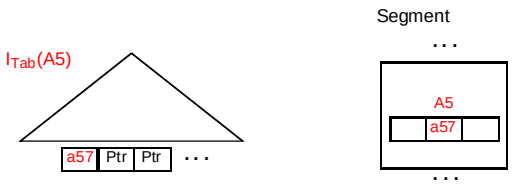


Indexierung von Tabellen

- **Speicherung von Tabellen**
  - **ungeordnete Tabelle**:
    Sätze (Zeilen) sind im Segment verstreut (Heap)
  - **geordnete Tabelle**:
    Sätze sind in B*-Baum eingebettet (key-sequenced table);
    es wird dadurch eine Clusterbildung erzielt
  - Wir bezeichnen eine solche Tabelle als Index-organisierte Tabelle (IT)
- **Indexierung von Tabellen**
  - mit sekundären Indexen für Spalten Ai: $I_{Tab}(Ai)$
  - Nutzung verschiedener Adressierungsverfahren
    - TID (physisch)
    - DBK (indirekt: logisch/physisch)
    - PK (Primärschlüssel: logisch)
    - hybride Verfahren

Indexierung von Tabellen

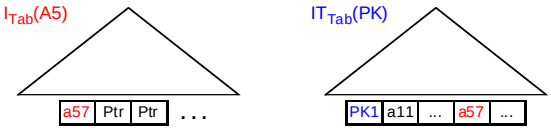
- **Wie spielen Adressierung und Tabellenspeicherung zusammen?**



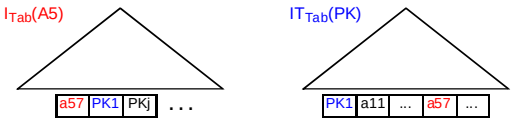
- **Ungeordnete Tabelle**
  - Sätze verschieben sich bei Einfügung (Aktualisierung) nicht (kaum)
  - Adressierungsverfahren (Ptr):
    Es kommen TID, DBK und DBK/PPP in Frage
  - Indexunterstützung für ungeordnete Tabellen in DB2, Sybase, MS SQL-Server, Oracle, ...

Indexierung von Tabellen

- **Index-organisierte Tabelle**



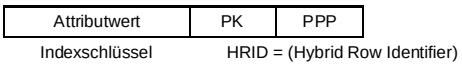
  - Split in IT_{Tab} erfordert viele Adressanpassungen in I_{Tab}(Ai)
    - bei TID
    - bei DBK
    - bei DBK/PPP
  - Verbesserung: logische Adressierung



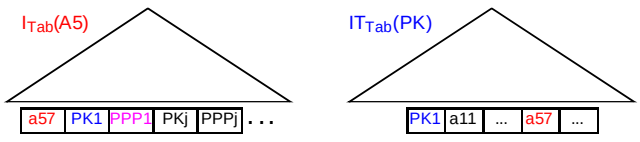
  - keine Wartung der I_{Tab}(Ai) bei Split/Verschiebungen in IT_{Tab} nötig
  - **aber:** höhere Zugriffskosten bei Index-Scan usw.

Indexierung von Tabellen

- **Nutzung einer hybriden Adressierungstechnik**
  - Verweis hat zwei Komponenten
    - logischer Verweis: PK
    - physischer Verweis: wahrscheinliche DB-Seite (PPP, Guess-DBA)
  - Eintrag in Index:



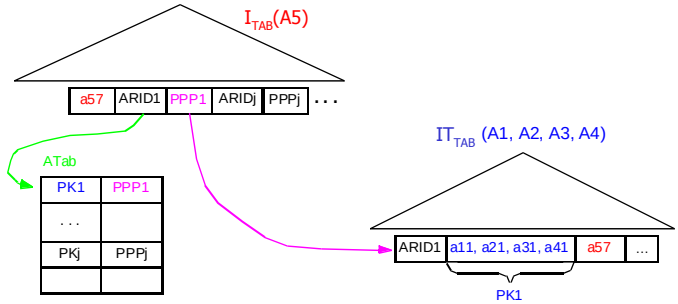
  - Index:



  - Vereinigung der Vorteile beider Verfahren
  - Was passiert bei langen Primärschlüsseln?

Indexierung von Tabellen

- **Optimierung bei langen Primärschlüsseln**
  - Beispiel: TAB (A1, A2, A3, A4, A5, ...)
  - Vermeidung der PK-Speicherung im Index (Oracle-Lösung)
    - Nutzung einer Abbildungstabelle ATab
    - Verweis auf ATab durch ARID



  - Falls der Zugriff über PPP fehlschlägt, wird mit Hilfe von ARID ATab aufgesucht
  - Alle I_{TAB}(Ai) benutzen ATab
  - Von dort kann über PPP oder über PK auf IT_{TAB} zugegriffen werden

Kapitel 4: Speicherungsstrukturen

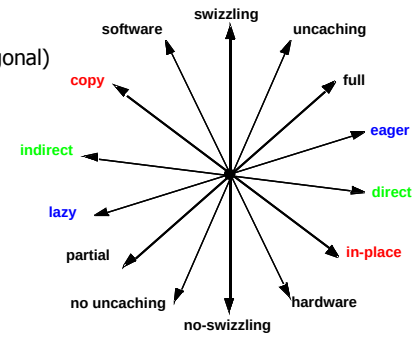
- Ziel: Entwurf von
  - Speicherungsstrukturen für Sätze und komplexe Objekte
  - Hilfsstrukturen wie Freispeicherverwaltung, Adressierung usw.
- Freispeicherverwaltung
  - im Segment
  - in der Seite
- Externspeicherbasierte Satzadressierung
  - TID
  - Zuordnungstabelle
  - Indexierung von Tabellen (Satzmengen)
- Hauptspeicherbasierte Satzadressierung
  - Klassifikation der Lösungskonzepte
  - Pointer-Swizzling-Verfahren
- Abbildung von Sätzen
  - feste/variable Felder
  - Partitionierung
- Speicherungsstrukturen für komplexe Objekte
  - Listen- und Mengenkonstruktoren
  - Tupelkonstruktoren
- Darstellung langer Felder
  - Abbildung auf Segmente
  - Zugriffsstruktur des Objekts

Hauptspeicherbasierte Adressierung

- Programme sollen im HSP transiente und persistente Datenobjekte transparent verarbeiten können.
  - Ausschließliche Nutzung von direkten Adressen im HSP (Virtuelle Adressierung), d.h., Zugriff auf persistente Objekte ist im HSP genauso effizient wie auf transiente Objekte.
  - Keine Mehrkosten für Programme, die nur auf transiente Objekte zugreifen
  - Abbildungskosten für persistente Objekte sollen nicht bei jedem Zugriff anfallen
- Abbildung von persistenten Objekten auf Externspeichern (ES) auf solche in Virtuellen Speichern (VS)
  - Persistente Adressen (z.B. SID, RID, TID) sind lang (z.B. 64 Bit), Virtuelle Adressen dagegen kürzer (z.B. 32 Bit)
  - Übersetzung der Zeiger (pointer swizzling) vom langen Format mit indirekter Adressierung ins kürzere Format mit möglichst direkter Adressierung
- Schnelle Verarbeitung von Pointerfolgen im VS - z.B. 10^5 Refs/sec
  - Direkter Zugriff im HSP ist wesentlich billiger als Zugriff über persistente Adresse (Lokalisierung einer Seite im DB-Puffer und Suche des Objektes in der Seite)
  - Objektverarbeitung: Traversierung von Referenzfolgen und Navigation in vernetzten Objektstrukturen
  - ggf. zusätzliche Zugriffspfade zur Suche im HSP: B*-Baumzugriff erfordert h+1 direkte Pointerreferenzen

Pointer Swizzling

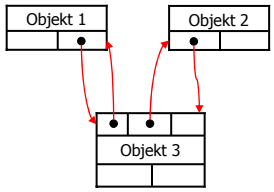
- **Dimensionen von Pointer Swizzling**
- **Klassifikation von Swizzling-Verfahren**
  wichtigste Kriterien: Ort, Zeitpunkt und Art (orthogonal)
  - **Ort:**
    - *In-Place Swizzling:* Beibehaltung der Objektformate und der Seitenstrukturen
    - *Copy Swizzling:* Kopieren der Objekte in einen Puffer und Umstellen der Zeiger in den Kopien
  - **Zeitpunkt:**
    - *Eager Swizzling:* Umstellen aller Zeiger, sobald die Objekte in den Hauptspeicher gebracht werden
    - *Lazy Swizzling:* Umstellen der Zeiger bei Erstreferenz oder später (nach beliebigen Kriterien — magische Zahl 3)
  - **Art:**
    - *Direct Swizzling:* Nutzung der Virtuellen Adresse des Objektes, dadurch kann die Ersetzung von Objekten während der Verarbeitung sehr schwierig oder gar unmöglich werden
    - *Indirect Swizzling:* Nutzung der Virtuellen Adresse von Objekt-Deskriptoren



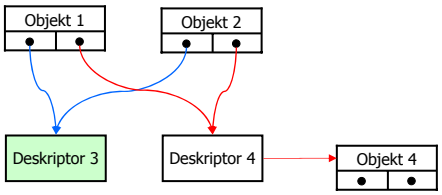
Quelle: [WW95]

Direktes und indirektes Swizzling

- **Prinzip**



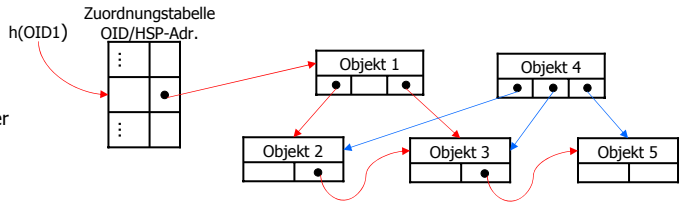
a) Symmetrische Referenzen



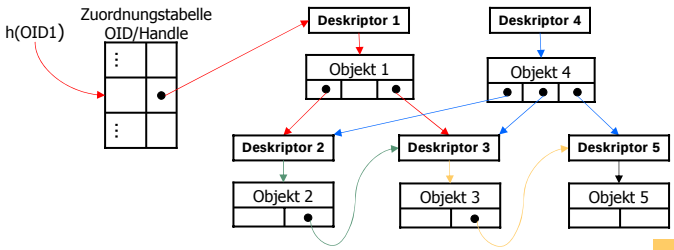
b) Referenzierung von Deskriptoren

Direkte und indirekte Variante beim Copy Swizzling

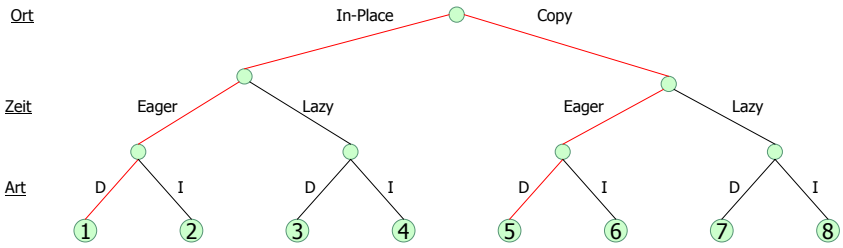
a) Direktes Swizzling in einem Objektpuffer



b) Indirektes Swizzling in einem Objektpuffer



Pointer Swizzling



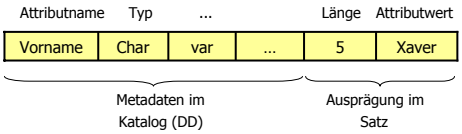
- Bemerkungen:
  - 1 + 5 : Swizzling von allen Seiten/Objekten bei Checkout, kein Ersetzen (no uncaching)
  - 2 + 4 : Umständliche Organisation
- Fragen:
  - Welche Verfahren sind bei der Verarbeitung (beim Swizzling) am schnellsten?
  - Welche Verfahren erlauben Objektersetzung (uncaching)?
  - Wie kann Lazy/Direct (3 + 7) realisiert werden?

Kapitel 4: Speicherungsstrukturen

- Ziel: Entwurf von
  - Speicherungsstrukturen für Sätze und komplexe Objekte
  - Hilfsstrukturen wie Freispeicherverwaltung, Adressierung usw.
- Freispeicherverwaltung
  - im Segment
  - in der Seite
- Externspeicherbasierte Satzadressierung
  - TID
  - Zuordnungstabelle
  - Indexierung von Tabellen (Satzmengen)
- Hauptspeicherbasierte Satzadressierung
  - Klassifikation der Lösungskonzepte
  - Pointer-Swizzling-Verfahren
- Abbildung von Sätzen
  - feste/variable Felder
  - Partitionierung
- Speicherungsstrukturen für komplexe Objekte
  - Listen- und Mengenkonstruktoren
  - Tupelkonstruktoren
- Darstellung langer Felder
  - Abbildung auf Segmente
  - Zugriffsstruktur des Objekts

Abbildung von Sätzen

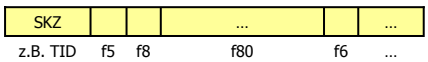
- Record-Manager:
  - physische Abspeicherung von Sätzen in Seiten
  - Operationen: Lesen, Einfügen, Modifizieren, Löschen
- Satzbeschreibung
  - pro Attribut:



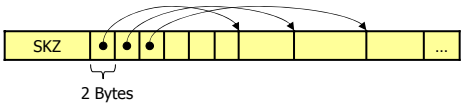
  - Satz- und Zugriffspfadbeschreibung im Katalog
  - besondere Methoden der Speicherung
    - Blank-/Nullunterdrückung
    - Zeichenverdichtung
    - kryptographische Verschlüsselung
    - Symbol für undefinierte Werte
  - Tabellenersetzung für Werte: KL = Kaiserslautern
- Organisation
  - n Satztypen pro Segment
  - m Sätze verschiedenen Typs pro Seite
  - Satzlänge < Seitenlänge: $S_L \leq L_S - L_{SK}$

Speicherungsstrukturen für Sätze

- Entwurfsziele:
  - Speicherplatzökonomie
  - schnelles Aufsuchen des i-ten Feldes (weitgehende Berechnung aus Kataloginformation)
  - dynamische Erweiterung (ALTER TABLE ...)
- Konkatenation von Feldern fester Länge
  Katalog: f5 | f8 | f80 | f6 | ... |



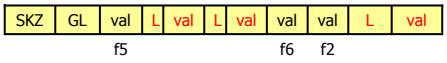
  - speicheraufwändig
  - unflexibel
- Zeiger im Vorspann
  Katalog: f5 | v | v | f6 | ... |



  - unflexibel

Speicherungsstrukturen für Sätze

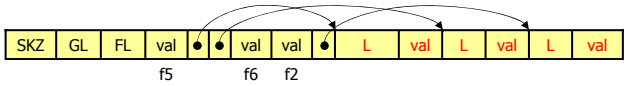
- Eingebettete Längenfelder
  Katalog: f5 | v | v | f6 | f2 | v |



  - GL = Gesamtlänge des Satzes
  - stärkere Nutzung des Katalogs
  - dynamische Erweiterung möglich

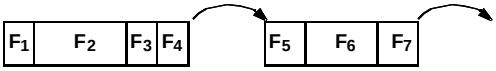
- Optimierung:
  eingebettete Längenfelder mit Zeigern
  Katalog: f5 | v | v | f6 | f2 | v |

  - FL = Längenangabe für den festen Strukturteil
  - Adresse des n-ten Attributs kann berechnet werden
  - dynamische Erweiterbarkeit



Speicherungsstrukturen für Sätze

- Problem: dynamisches Wachstum/variable Länge
  - Ausdehnung und Schrumpfung in einer Seite
  - Überlaufschemata, Garbage Collection
    - Eingeführte Möglichkeiten der Speicherung von Sätzen sind mit weiteren Optionen zu kombinieren
- Strikt zusammenhängende Speicherung von Sätzen
  - evtl. häufige Umlagerung bei hoher Änderungsfrequenz
  - Vorteile für indirekte Adressierungsschemata
- Aufspaltung des Satzes:



  - Ordnung nach Referenzhäufigkeiten
  - Verbesserung der Clusterbildung
  - Wiederholter Überlauf möglich
  - wird unvermeidlich bei der Einbeziehung von Attributen vom Typ TEXT oder BILD

Kapitel 4: Speicherungsstrukturen

- Ziel: Entwurf von
  - Speicherungsstrukturen für Sätze und komplexe Objekte
  - Hilfsstrukturen wie Freispeicherverwaltung, Adressierung usw.
- Freispeicherverwaltung
  - im Segment
  - in der Seite
- Externspeicherbasierte Satzadressierung
  - TID
  - Zuordnungstabelle
  - Indexierung von Tabellen (Satzmengen)
- Hauptspeicherbasierte Satzadressierung
  - Klassifikation der Lösungskonzepte
  - Pointer-Swizzling-Verfahren
- Abbildung von Sätzen
  - feste/variable Felder
  - Partitionierung
- Speicherungsstrukturen für komplexe Objekte
  - Listen- und Mengenkonstruktoren
  - Tupelkonstruktoren
- Darstellung langer Felder
  - Abbildung auf Segmente
  - Zugriffsstruktur des Objekts

Speicherungsstrukturen für komplexe Objekte

- Komplexe Objekte werden gebildet aus
  - atomaren Werten und darauf rekursiv angewandten Mengen-, Listen- und Tupelkonstruktoren
- Einfaches Beispiel:

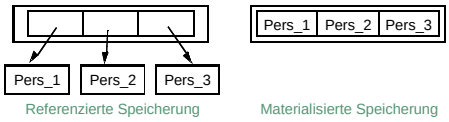
```
complex_object Mitarbeiter [. . .]
        set [. . .] of tuple (
                Pers_Nr      [. . .] : integer,
                Name         [. . .] : string (30),
                Gehalt       [. . .] : real,
                Lebenslauf   [. . .] : var_string)
```

  - [. . .] kennzeichnet Stelle für Speicherungsstrukturbeschreibung

Quelle: [KD93]

Speicherungsstrukturen für komplexe Objekte

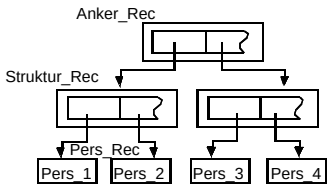
- Freiheitsgrade für physische Speicherungsstrukturen
  - Wahl der internen Speicherungsstrukturen zur Implementierung von Mengen, Listen und Tupeln (Konstruktordatenstruktur)
  - Direkte Speicherung oder Referenzierung der Elemente einer Menge oder Liste bzw. der Attribute eines Tupels in der Konstruktordatenstruktur
- Jeder Konstruktor hat eine Konstruktordatenstruktur
- Beispiel:
  - einfache Menge {Pers_1, Pers_2, Pers_3}
  - variabel langer Array als Konstruktordatenstruktur



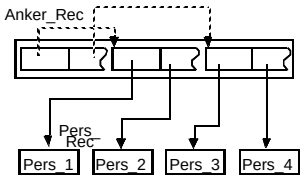
Referenzierte Speicherung Materialisierte Speicherung

Speicherungsstrukturen für komplexe Objekte

- Zweimalige Anwendung des Mengenkonstruktors
  - { {Pers_1 , Pers_2} , {Pers_3 , Pers_4} }
  - Vorgabe variabel langer Arrays als Konstruktordatenstrukturen
- Vier Implementierungen:

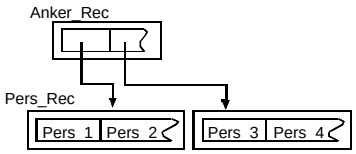


1. Elemente äußere Menge : referenziert
   Elemente innere Menge : referenziert

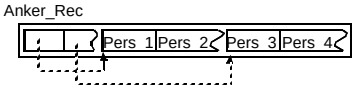


2. Elemente äußere Menge : materialisiert
   Elemente innere Menge : referenziert

Speicherungsstrukturen für komplexe Objekte



3. Elemente äußere Menge : referenziert
   Elemente innere Menge : materialisiert



4. Elemente äußere Menge : materialisiert
   Elemente innere Menge : materialisiert

- Sind zusätzlich verkettete Listen als Konstruktordatenstrukturen zulässig, so erhält man insgesamt 16 Varianten

Speicherungsstrukturen für Mengen und Listenkonstruktoren

- Unabhängige Freiheitsgrade
  - Konstruktordatenstruktur
    - variabel langes Array, verkettete Liste, …
  - Art der Speicherung der Elemente
    - direkt in Konstruktordatenstruktur
    - Referenzierung der Elemente über Zeiger
- Zur unabhängigen Spezifikation dieser Freiheitsgrade sind zwei Parameter (in einer Datendefinitionssprache) erforderlich:

```
object_type = . . .
/* Definition einer Menge. */
set       [implementation     =     implementation_type,
           element_placement  =     placement_type] of object_type |

/* Definition einer Liste. */
list      [implementation     =     implementation_type,
           element_placement  =     placement_type] of object_type | ...
```

Speicherungsstrukturen für Mengen und Listenkonstruktoren

- Parameterwerte:

```
implementation_type    = array | linked_list
placement_type         = inplace | referenced (record_type_name)
```

- Vollständige Definition der Speicherungsstruktur (Fall 1)

```
complex_object Menge_von_Mengen_von_Pers [anchor_record_type=Anker_Rec]

    set [implementation=array, element_placement=referenced (Struktur_Rec)] of

        set [implementation=array, element_placement=referenced (Pers_Rec)] of

            Pers.
```


Speicherungsstrukturen für Tupelkonstruktoren

- Prinzipiell existieren die gleichen Freiheitsgrade
  - Wahl einer Konstruktordatenstruktur: Zuordnung des Tupels zu einem Satz (Record) oder zu mehreren Sätzen
  - materialisierte oder referenzierte Speicherung der Attributwerte
- Neuer Parameter: „location"

```
attribute_description = attribute_name  [location = location_type,
                                         element_placement = placement_type]
```

  - Für jedes Attribut kann „location" und „element_placement" separat spezifiziert werden
- „location" erlaubt die Optimierung des Satzzugriffs nach den Zugriffshäufigkeiten der einzelnen Attribute

```
location_type = primary |secondary (record_type_name)
```

  - Die Konstruktordatenstruktur eines Tupels kann auf mehrere Sätze aufgeteilt werden. Mit „primary" wird das zugehörige Attribut im Primärblock angelegt.

Speicherungsstrukturen – Beispiel

- Ausprägung einer Mitarbeiterrelation

Mitarbeiter			
Pers_Nr	Name	Gehalt	Lebenslauf
77234	Maier	4000	Frau Bettina Maier ist am ...
77235	Schmidt	5000	Herr Fritz Schmidt ist am ...

- Definition einer dazugehörigen Speicherungsstruktur
  1. referenzierte Prim_Rec

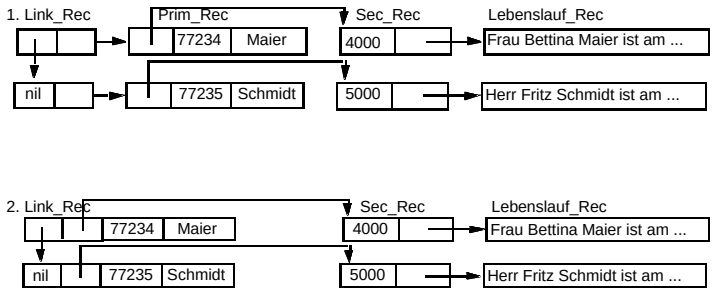
```
complex_object Mitarbeiter [anchor_record_type=Link_Rec]
      set [implementation=linked_list, element_placement=referenced (Prim_Rec)] of tuple (
            Pers_Nr      [location=primary, element_placement=inplace] : integer,
            Name         [location=primary, element_placement=inplace] : string (30),
            Gehalt       [location=secondary (Sec_Rec), element_placement=inplace] : real,
            Lebenslauf   [location=secondary (Sec_Rec),
                          element_placement=referenced (Lebenslauf_Rec)]] : var_string)
```

  2. materialisierte Prim_Rec

```
complex_object Mitarbeiter [anchor_record_type=Link_Rec]
            set [implementation=linked_list, element_placement=inplace] of . . .
```


Speicherungsstrukturen – Beispiel

- Dazugehörige Speicherungsstrukturen für die Mitarbeiterrelation



Kapitel 4: Speicherungsstrukturen

- Ziel: Entwurf von
  - Speicherungsstrukturen für Sätze und komplexe Objekte
  - Hilfsstrukturen wie Freispeicherverwaltung, Adressierung usw.
- Freispeicherverwaltung
  - im Segment
  - in der Seite
- Externspeicherbasierte Satzadressierung
  - TID
  - Zuordnungstabelle
  - Indexierung von Tabellen (Satzmengen)
- Hauptspeicherbasierte Satzadressierung
  - Klassifikation der Lösungskonzepte
  - Pointer-Swizzling-Verfahren
- Abbildung von Sätzen
  - feste/variable Felder
  - Partitionierung
- Speicherungsstrukturen für komplexe Objekte
  - Listen- und Mengenkonstruktoren
  - Tupelkonstruktoren
- Darstellung langer Felder
  - Abbildung auf Segmente
  - Zugriffsstruktur des Objekts

Darstellung und Handhabung langer Felder

- Anwendungsbereiche: CAD, CASE, GIS, Multimedia, …
- Anforderungen:
  - idealerweise keine Größenbeschränkung
  - allgemeine Verwaltungsfunktionen
  - cursorgesteuertes Lesen und Schreiben (stückweise Handhabung)
  - Verkürzen, Verlängern und Kopieren
  - Suche nach vorgegebenem Muster, Längenbestimmung
- Darstellung großer Speicherobjekte
  - besteht potentiell aus vielen Seiten und Segmenten
  - ist eine uninterpretierte Bytefolge
  - Adresse (OID, object identifier) zeigt auf Objektkopf (header)
  - OID ist Stellvertreter im Satz, zu dem das lange Feld gehört
  - geforderte Verarbeitungsflexibilität bestimmt Zugriffs- und Speicherungsstruktur

Quelle: [Bil92]

Darstellung und Handhabung langer Felder

- Verarbeitungsprobleme:
  - Ist die Objektgröße vorab bekannt?
  - Gibt es während der Lebenszeit des Objekts viele Änderungen?
  - Ist schneller sequentieller Zugriff erforderlich?
- Abbildung auf Externspeicher:
  - seitenbasiert
    - Einheit der Speicherzuordnung: eine Seite
    - "verstreute" Sammlung von Seiten
  - segmentbasiert (mehrere Seiten)
    - Segmente fester Größe (EXODUS)
    - Segmente mit einem festen Wachstumsmuster (STARBURST)
    - Segmente Variabler Größe (EOS)
  - Zugriffsstruktur zum Objekt
    - Kettung der Segmente/Seiten
    - Liste von Einträgen (Deskriptoren)
    - B*-Baum

Quelle: [Bil92]

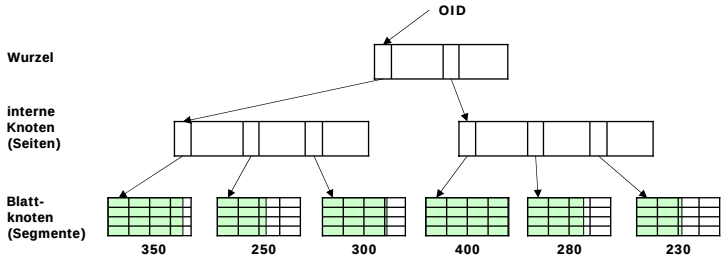
Lange Felder in EXODUS

- Speicherung langer Felder:
  - Daten werden in (kleinen) Segmenten fester Größe abgelegt
  - bei bekannter Verarbeitungscharakteristik Wahl geeigneter Segmentgrößen möglich
  - Einfügen von Bytefolgen einfach und überall möglich
  - schlechteres Verhalten bei sequentiellem Zugriff
- B*-Baum als Zugriffsstruktur:
  - Blätter sind Segmente fester Größe (hier 4 Seiten à 100 Bytes)
  - interne Knoten und Wurzel sind Index für Bytepositionen
  - interne Knoten und Wurzel speichern für jeden Kind-Knoten Einträge der Form (Zähler, Seiten-#)
  - Zähler enthält die maximale Bytenummer des jeweiligen Teilbaums (links stehende Seiteneinträge zählen zum Teilbaum)
  - Zähler im weitesten rechts stehenden Eintrag der Wurzel enthält Länge des Objekts

Quelle: [CD+86]

Lange Felder in EXODUS

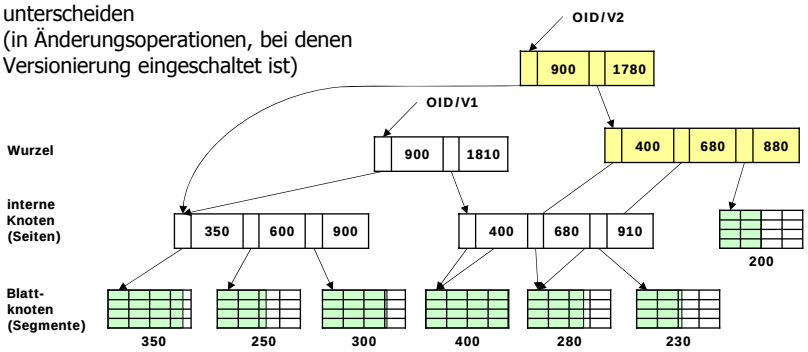
- Repräsentation sehr langer dynamischer Objekte:
  - bis zu 1GB mit drei Baumebenen (selbst bei kleinen Segmenten)
  - Speicherplatznutzung typischerweise ~ 80 %
- Einfache Operationen:
  - Suche nach einem Byteintervall
  - Einfügen/Löschen einer Bytefolge an/von einer vorgegebenen Position
  - Anhängen einer Bytefolge ans Ende des langen Feldes



Lange Felder in EXODUS

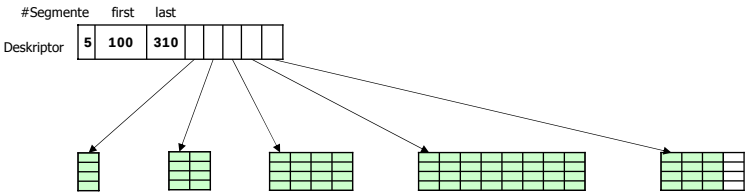
- Unterstützung versionierter Speicherobjekte:
  - Markierung der Objekt-Header mit Versionsnummer
  - Kopieren und Ändern nur der Seiten, die sich in der neuen Version unterscheiden (in Änderungsoperationen, bei denen Versionierung eingeschaltet ist)

Versionsbestimmende Operation:
Löschen von 30 Bytes
am Ende der Version V1



Lange Felder in Starburst

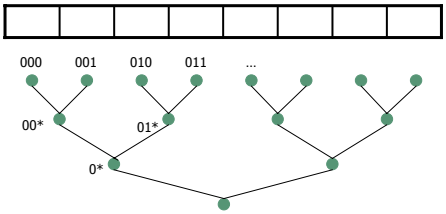
- Erweiterte Anforderungen:
  - Effiziente Speicherallokation und -freigabe für Feldgrößen von bis zu 100 MB - 2 GB (Sprache, Bild, Musik oder Video)
  - Hohe E/A-Leistung: Schreib- und Lese-Operationen sollen E/A-Raten nahe der Übertragungsgeschwindigkeit der Magnetplatte erreichen
- Prinzipielle Repräsentation:
  - Deskriptor mit Liste der Segmentbeschreibungen
  - Langes Feld besteht aus einem oder mehreren Segmenten
  - Segmente, auch als Buddy-Segmente bezeichnet, werden nach dem Buddy-Verfahren in großen vordefinierten Bereichen fester Länge auf Externspeicher angelegt



Quelle: [LL89]

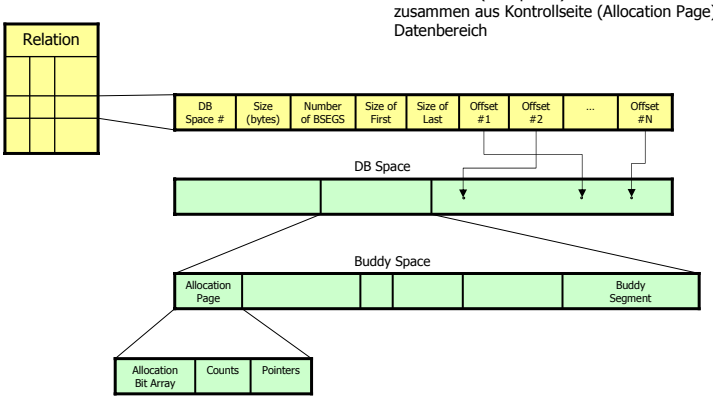
Segmentallokation in Starburst

- Segmentallokation bei vorab bekannter Objektgröße:
  - Objektgröße G (in Seiten)
  - G ≤ MaxSeg: es wird ein Segment angelegt
  - G > MaxSeg: es wird eine Folge maximaler Segmente angelegt
  - letztes Segment wird auf verbleibende Objektgröße gekürzt
- Segmentallokation bei unbekannter Objektgröße:
  - Wachstumsmuster der Segmentgrößen wie im Beispiel: 1, 2, 4, ..., 2^n Seiten werden jeweils zu einem Buddy-Segment zusammengefasst
  - MaxSeg = 2048 für n = 11
  - Falls MaxSeg erreicht wird, werden weitere Segmente der Größe MaxSeg angelegt
  - Letztes Segment wird auf die verbleibende Objektgröße gekürzt
- Allokation von Buddy-Segmenten in sequentiellem Buddy-Bereich gemäß binärem Buddy-Verfahren
  - Zusammenfassung zweier Buddies der Größe $2^n \Rightarrow 2^{n+1}$ ($n \geq 0$)



Speicherorganisation in Starburst

- Aufbau eines langen Feldes:
  - Deskriptor des langen Feldes (< 255 Bytes) ist in Relation gespeichert
  - Aufbau aus einem oder mehreren Buddy-Segmenten, die in großen vordefinierten Buddy-Bereichen fester Länge auf Platte angelegt werden
  - Buddy-Segmente enthalten nur Daten und keine Kontrollinformation
  - Segment besteht aus 1, 2, 4, 8, ... oder 2048 Seiten
    (=> max. Segmentgröße 2 MB bei 1 KB-Seiten)
  - Buddy-Bereiche sind allokiert in (noch größeren) DB-Dateien (DB Spaces). Sie setzen sich zusammen aus Kontrollseite (Allocation Page) und Datenbereich

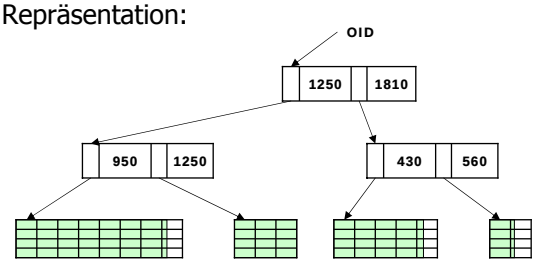


Lange Felder in Starburst

- Verarbeitungseigenschaften:
  - effiziente Unterstützung von sequentiellen und wahlfreien Lesevorgängen
  - einfaches Anhängen und Entfernen von Bytefolgen am Objektende
  - schwieriges Einfügen und Löschen von Bytefolgen im Objektinnern

Lange Felder in EOS

- Speicherallokation mit variablen Segmenten als Verallgemeinerung des EXODUS- und STARBURST-Ansatzes:
  - Objekt ist gespeichert in einer Folge von Segmenten variabler Größe
  - Segment besteht aus Seiten, die physisch zusammenhängend auf Externspeichern angeordnet sind
  - nur die letzte Seite eines Segmentes kann freien Platz aufweisen
  - Überbau von EXODUS-Ansatz übernommen
- Prinzipielle Repräsentation:



Quelle: [Bil92]

Lange Felder in EOS

- Verarbeitungseigenschaften:
  - die guten operationalen Eigenschaften der beiden zugrunde liegenden Ansätze können erzielt werden
  - Reorganisation möglich, falls benachbarte Segmente sehr klein (Seite) werden

Zusammenfassung

- Freispeicherinformation auf verschiedenen Ebenen erforderlich: Gerät, Segment (Datei), Seite
- Ziele bei der externspeicherbasierten Adressierung
  - Kombination der Geschwindigkeit des direkten Zugriffs mit der Flexibilität einer Indirektion
  - Satzverschiebungen in einer Seite ohne Auswirkungen
  - TID, DBK (Zuordnungstabelle) oder Primärschlüssel
- Indexierung von Tabellen
  - physische oder hybride Verfahren bei ungeordneten Tabellen
  - hybride Verfahren kombiniert mit Primärschlüssel bei geordneten Tabellen (Index-organisierte Tabellen)
- Hauptspeicherbasierte Adressierung (Pointer Swizzling)
  - transparenter Programmzugriff auf persistente und transiente Objekte
  - Abbildung von langen ES-Adressen auf Virtuelle Adressen
  - orthogonale Klassifikationskriterien: Ort, Zeitpunkt, Art
- Abbildung von Sätzen
  - Speicherung variabel langer Felder
  - dynamische Erweiterungsmöglichkeiten
  - Berechnung von Feldadressen
- Speicherung komplexer Objekte
  - Listen-, Mengen- und Tupelkonstruktoren
  - Konstruktor-Anwendung ist orthogonal und rekursiv
- Speicherung großer Objekte

Literatur zu diesem Kapitel

[Bil92] Biliris, A.: The Performance of Three Database Storage Structures for Managing Large Objects. In: Proc. ACM SIGMOD, San Diego, California, 1992.

[CD+86] M.J. Carey, D.J. DeWitt, J.E. Richardson, E.J. Shekita: Object and File Management in the EXODUS Extensible Database System. In: Proc. 12th VLDB Conference, 1986.

[KD93] Keßler, U., Dadam, P.: Benutzergesteuerte, flexible Speicherungsstrukturen für komplexe Objekte. In: Tagungsband BTW, Braunschweig, 1993.

[LL89] T.J. Lehman, B.G. Lindsay: The Starburst Long Field Manager. In: Proc. 15th VLDB Conference, 1989.

[WW95] White, S.J., DeWitt, D.J.: Quickstore: A High Performance Mapped Object Store. In: The VLDB Journal 4:4, Oct. 1995, pp. 629-674.