

## 7. Speicherungsstrukturen

- ☐ Freispeicherverwaltung
  - in einer Datei und in einer Seite
- ☐ Externspeicherbasierte Satz- und Objektadressierung
  - TID und Zuordnungstabelle
  - Physische und logische OID
- ☐ Hauptspeicherbasierte Satzadressierung
- ☐ Abbildung von Sätzen
  - feste/variable Felder
  - Partitionierung
- ☐ Speicherungsstrukturen für komplexe Objekte
- ☐ Darstellung langer Felder
  - Abbildung auf Segmente
  - Zugriffsstruktur zum Objekt

282.

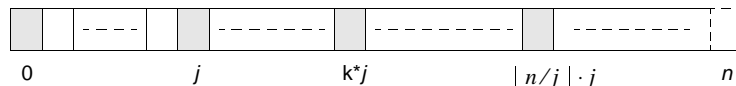
## Datenstrukturen zur Freispeicherverwaltung

### Freispeicherlisten

- ☐ Alle Seiten, die mit hoher Wahrscheinlichkeit einen Datensatz aufnehmen können, werden miteinander verkettet.
- ☐ Solche Listen werden aber sehr selten (z. B. in Oracle) verwendet.

### Freispeicherseiten (FSS)

- ☐ Solche Seiten werden equidistant in der Datei gespeichert.



- ☐ Seiten beinhalten statistische Daten über den Füllgrad von den nachfolgenden Seiten (bis zur nächsten FSS):
  - Anzahl der freien Bytes
  - oder einfach eine Klassifizierung des Füllgrads (z. B. in 16 Klassen)

284.

## 7.1 Freispeicherverwaltung

- ☐ Freispeicherverwaltung (FPA) für
  - Externspeicher (Allokation von Dateien)
  - Dateien (Allokation von internen Sätzen)
  - Seiten (Verwaltung von belegten/freien Einträgen)

### Freispeicherverwaltung in Dateien

- ☐ Hauptaufgabe:  
Gegeben ein Datensatz mit  $b$  Bytes. Suche eine Seite, in der der Datensatz abgespeichert werden kann.

### (widersprechende) Ziele

- ☐ schnelles Auffinden einer freien Seite
- ☐ gute Speicherplatzausnutzung
  - optimale Platzierung von Datensätzen  $\Rightarrow$  Bin-Packing Problem.
  - heuristische Lösungen

283.

## Algorithmen

### Append Only

- ☐ Naiver Ansatz (AO):
  - Speichere die Datensätze in einer ausgewählten Seite bis diese voll ist. Bestimme danach eine neue (leere) Seite
  - Problem:  
Einfügen langer Datensätze kann Seiten erzeugen, die einem kleineren Datensatz noch genügend Platz bieten würde.
- ☐ Verallgemeinerung AO( $n$ ):
  - Speichere die Datensätze in  $n$  ausgewählten Seiten.
- ☐ Hauptproblem beider Ansätze:  
Löschen von Datensätzen erzeugt Löcher in der Datei, die aber nicht im laufenden Betrieb gestopft werden.

285.

## FirstFit

- ❑ Durchlaufe beginnend vom Anfang der Datei die Seiten bis eine gefunden wurde, in die der Datensatz eingefügt werden kann (ggf. Allokation einer neuen Seite).
- ❑ Eigenschaften
  - Speicherplatzausnutzung ist nur um den Faktor 1.7 schlechter (im worst case) als beim optimalen Algorithmus.
  - teure Berechnung einer freien Seite (insbesondere wenn Datensätze gleicher Größe eingefügt werden).

## Varianten

- ❑ NextFit (NF)
    - Zeiger verweist auf die Seite, wo der letzte Datensatz eingefügt wurde.
    - Suche startet immer von dieser Seite (statt von der ersten Seite in der Datei)
    - ⇒ Reduzierung der Kosten bei einer **erfolgreichen** Suche
  - ❑ NextFit mit Histogrammen NF(H)
    - Zur Vermeidung von einer teuren **erfolglosen** Suche kann in einem Histogramm die Anzahl der Seiten in einer Füllgradklasse abgespeichert werden.
- Erfolgreiche Suchen werden durch Histogramme aber nicht unterstützt.

286.

## 7.2 Externspeicherbasierte Satzadressierung

### Anforderungen

- schneller, möglichst direkter Satzzugriff
- hinreichend stabil gegen geringfügige Verschiebungen (Verschiebungen innerhalb einer Seite ohne Auswirkungen)
- seltene oder keine Reorganisationen

### Adressierung in Dateien

- ❑ logisch zusammenhängender Adressraum
- ❑ direkte Adressierung (logische Byte-Adresse, RBA)
  - instabil bei Verschiebungen
- ❑ deshalb indirekte Adressierung

288.

## Hybrides Verfahren (McAuliffe, Carey, Solomon 1996):

- ❑ Einführung eines Kontrollparameters **u** für die Speicherplatzausnutzung (SPAN) in einer Datei.
 
$$\text{SPAN} = \frac{\text{Anzahl der durch Datensätze belegten Bytes}}{\text{Anzahl der durch Seiten belegten Bytes}}$$
- ❑ Verwendung eines Caches, der den Füllgrad von bis zu n Seiten protokolliert.

## Algorithmus

Falls ( $\text{SPAN} > u$ )

Verwende die Strategie AO(n) unter Ausnutzung der Information im Cache;

Sonst

Falls (eine der Seiten im Cache frei ist)

Speichere den Datensatz in dieser Seite;

Ändere die entsprechende Information im Cache;

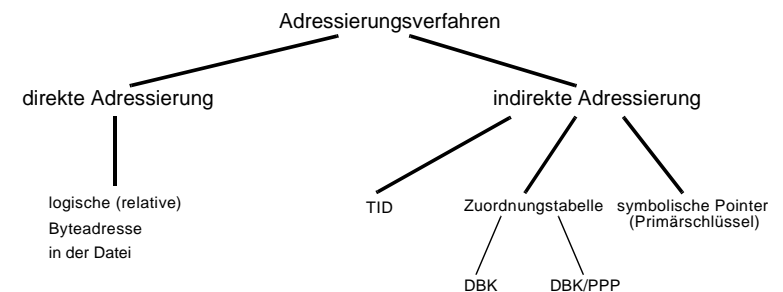
Sonst

Verwende die Strategie NF(H);

Füge die entsprechende Seiteninformation in den Cache ein;

287.

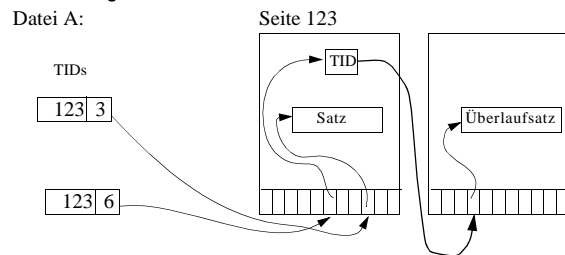
## Techniken zur externspeicherbasierten Satzadressierung



289.

## 7.2.1 Satzadressierung: TID-Konzept

- ❑ TID (tuple identifier) besteht aus zwei Komponenten:
  - Seitennummer
  - relative Indexposition innerhalb der Seite
  - dient zur Adressierung in einer Datei (z.B. Datei A)
- ❑ Migration eines Satzes in andere Seite ohne TID-Änderung möglich
  - Einrichten eines Stellvertreter-TID in Primärseite
- ❑ Überlaufkette: Länge  $\leq 1$



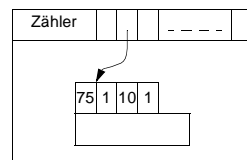
290.

## 7.2.2 Objektidentifikation

### Physische Identifikation

- ❑ Ein OID entspricht der Adresse auf dem Hintergrundspeicher, an der das Objekt abgespeichert ist.
- ❑ Falls das Objekt verschoben wurde, muß ein Verweis auf die neue Position abgespeichert werden.
- ❑ Beispiel einer Seitenkennung im System EXODUS

Seite (4 Byte)	Slot (2 Byte)	Segment (2 Byte)	Zähler (4 Byte)
75	1	10	1

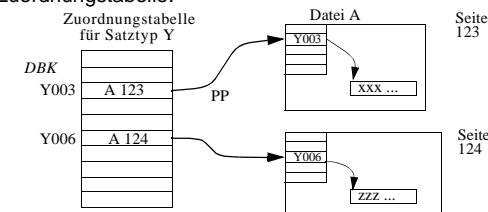


- Im **Unterschied zum TID** wird noch ein Zähler mitgeführt. Dadurch soll sichergestellt werden, daß eine OID nur einmalig an ein Objekt vergeben wird.

292.

## Satzadressierung über Zuordnungstabellen

- ❑ jeder Satz erhält eindeutigen logischen Identifikator
  - Datenbankschlüssel (**DBK**)
  - Vergabe des DBK erfolgt i.a. durch DBS
  - systeminterne Verweise auf Sätze erfolgen ausschließlich über den DBK
- ❑ Zuordnungstabelle enthält pro DBK zugehörige Seitenadresse (**PP**)
  - Identifikationsnummer des Segments (**SID**)
  - Seitennummer
- ❑ Verwendung von 'probable page pointers' (PPP) in Zugriffspfaden erspart u.U. Zugriff auf Zuordnungstabelle.



291.

### Operationen

- ❑ Erzeugen eines neuen Identifikator
  - Es wird zunächst eine Seite gesucht, in der das Objekt abgespeichert werden kann. Der Zähler des OID bekommt den Zähler der Seite zugewiesen.
- ❑ Bestimmung des Aufenthaltsorts eines Objekts
  - Es wird über den Slot auf das Objekt zugegriffen. Stimmen die OID des dort abgespeicherten Objekts nicht mit dem vorgegebenen OID überein, liegt ein Fehler vor.
- ❑ Verschieben eines Objekts
  - Wie bei TID wird auch beim OID verfahren. In dem Slot der Seite wird eine Vorwärtsreferenz auf eine neue Seite abgespeichert.
- ❑ Löschen eines Objekts
  - Die Seite wird besucht und eine ggf. vorhandene Vorwärtsreferenz gelöscht. Danach wird das eigentliche Objekt gelöscht.

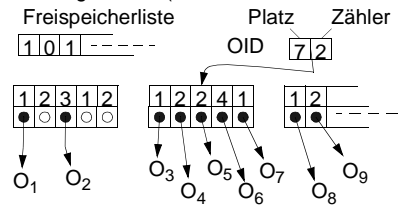
### Beurteilung

- ❑ Solange nicht viele Objekte verschoben werden, ist eine Realisierung mit physischen Identifikatoren interessant.

293.

## Logische Identifikatoren

- ❑ OID entspricht einer Seriennummer, die mittels einer Übersetzungstabelle in die physische Adresse umgewandelt wird.
- ❑ Zentraler Punkt ist hierbei die Implementierung der Übersetzungstabelle
  - Hashtabelle
  - B+-Baum
  - direkte Abbildung
- ❑ direkte Übersetzungstabelle (wird auch bei Netzwerk-Datenbanken benutzt)



- Platz eines OID entspricht der Position in der Übersetzungstabelle
- Zähler dient der Wiederverwendung eines Platzes in der Übersetzungstabelle

294.

## Operationen

- ❑ Erzeugen eines neuen Identifikator
  - In der Freispeicherliste wird für jede Seite genau ein Bit reserviert. Falls das Bit gesetzt ist, gibt es noch freie Einträge in der entsprechenden Seite der Übersetzungstabelle.
  - Zuerst: Suche nach einem freien Platz (Freispeicherliste). Bei erfolgloser Suche: Anhängen einer neuen Seite an die Übersetzungstabelle.
- ❑ Bestimmung des Aufenthaltsorts eines Objekts
  - mit Hilfe der OID direkter Zugriff auf Übersetzungstabelle
  - In der Übersetzungstabelle wird zusätzlich zur Adresse des Objekts noch ein Zähler abgespeichert. Falls dieser Zähler ungleich dem Zähler im OID ist, existiert das gesuchte Objekt nicht. Bei Übereinstimmung kann mittels der Adresse auf das Objekt zugegriffen werden.
- ❑ Löschen eines Objekts
  - physische Adresse wird auf einen ungültigen Wert (**null**) gesetzt
  - ggf. Abänderung der Freispeicherliste

295.

## Vergleich physischer und logischer Identifikationen

- ❑ Suche nach einem Objekt
  - Umsetzen einer logischen Referenz kostet ein Zugriff auf die Übersetzungstabelle
  - physische Referenz kann direkt umgesetzt werden (aber ggf. noch einen weiteren Zugriff)
- ❑ Länge der Identifikatoren
  - logische Referenz ist i. a. kürzer als eine physische Referenz
- ❑ Umspeicherung der Objekte
  - bei logischen Referenzen: schnelle Reorganisation des Datenbestands möglich
  - Bei physischen Referenzen ist dies nur dadurch möglich, daß man an der ursprünglichen Position einen Verweis auf die neue Position des Objekts abspeichert.

## Praxis

- ❑ In den früheren kommerziellen OODBMS findet man i. a. nur physische Identifikatoren, während in neueren Systemen auch logische Identifikatoren verwendet werden.

296.

## 7.3 Abbildung von Sätzen

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

Attributname	Typ	...		Länge	Attributwert
Vorname	Char	var	...	5	Xaver

Metadaten im Katalog

Ausprägung im Satz

- Satz- und Zugriffspfadbeschreibung im Katalog (Data Dictionary)
- ❑ besondere Methoden der Speicherung
  - Blank-/Nullunterdrückung und Zeichenverdichtung
  - kryptographische Verschlüsselung
  - Symbol für undefinierte Werte

297.

## Abspeicherungsformen

- ❑ Seite besteht aus
  - Seitenkopf (beinhaltet z. B. Zeiger auf nächste Seite der Relation)
  - Folge von Datensätzen
- ❑ mehrere Satztypen pro Datei, mehrere Satztypen pro Seite
- ❑ Annahme: Satzlänge < Seitenlänge (d.h.  $S_L < L_S - L_{SK}$ )
  1. Satz mit Attributen fester Länge:
 

z.B. TID

SKZ				...
-----	--	--	--	-----

f      f      f

    - ❑ für Attribute variabler Länge
      - speicheraufwendig
      - unflexibel
  2. Satz mit Attributen variabler Länge (mit Zeiger im Vorspann):
 

SKZ										...
-----	--	--	--	--	--	--	--	--	--	-----

    - unflexibel

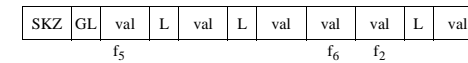
298.

## 11.1 Verwaltung langer Felder (BLOB)

- ❑ BLOB = **B**inary **L**arge **O**bject
- ❑ Anforderungen
  - idealerweise keine Größenbeschränkung an Objekte
  - 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 oder 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

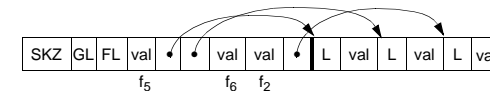
300.

- ❑ Flexiblere Abspeicherungsformen mit Zugriffsmöglichkeit auf einzelne Attribute
- ❑ Betrachte einen Datensatz der Form (f5, v, v, f6, f2, v) (f = fest, v = variabel)
  - eingebettete Längfelder



- dynamische Erweiterung möglich

- Optimierung: eingebettete Längfelder mit Zeigern



Adresse des n-ten Attributs kann berechnet werden

299.

- ❑ Verarbeitungsprobleme
  - Ist Objektgröße vorab bekannt?
  - Gibt es während der Lebenszeit des Objektes 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<sup>+</sup>-Baum wird als Zugriffsstruktur verwendet

301.

## 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 sequentiellm Zugriff

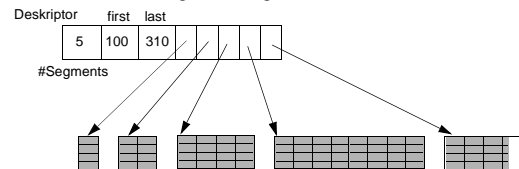
### B\*-Baum als Zugriffsstruktur

- ☐ Blätter sind Segmente fester Größe (hier 4 Seiten à 400 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 Objektes

302.

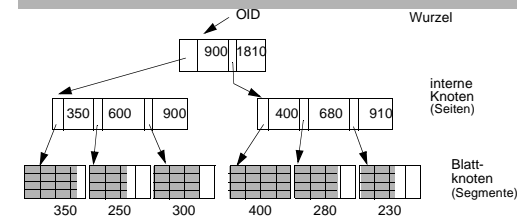
## Lange Felder in STARBURST

- ☐ 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
  - Segmente zur Darstellung des langen Feldes

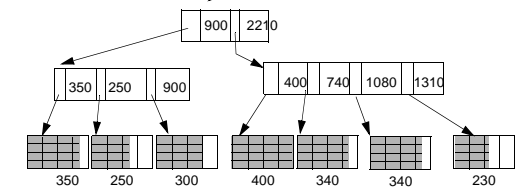


304.

## Beispiel



### Einfügen von 400 Bytes an der Position 1400:



### Eigenschaften

- ☐ Objekte bis zu 1 GB können mit drei Baumebenen (selbst bei kleinen Segmenten) verwaltet werden.
- ☐ Speicherplatznutzung typischerweise ~ 80 %
- ☐ Einfügen und Löschen wirkt sich i.a. nur auf einen Pfad des Baums aus.

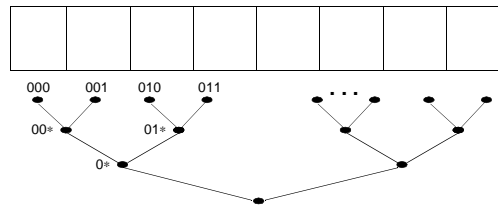
303.

## Segmentallokation

- ☐ bei vorab bekannter Objektgröße
  - Objektgröße G (in Seiten)
  - $G \leq \text{MaxSeg}$ : es wird ein Segment angelegt
  - $G > \text{MaxSeg}$ : 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
  - $\text{MaxSeg} = 2048$  für  $n = 11$
  - Falls  $\text{MaxSeg}$  erreicht wird, werden weitere Segmente der Größe  $\text{MaxSeg}$  angelegt
  - Letztes Segment wird auf die verbleibende Objektgröße gekürzt

305.

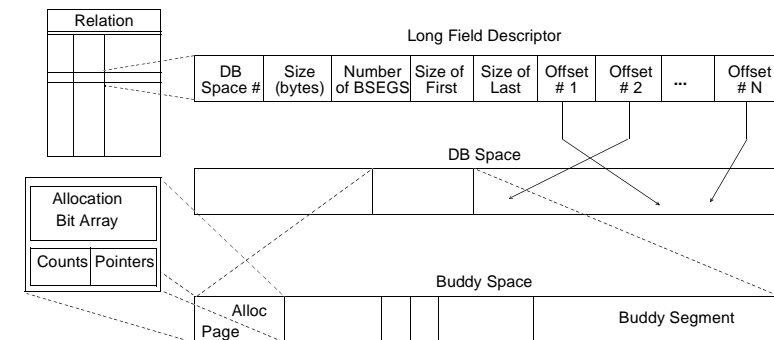
- ❑ Allokation von Buddy-Segmenten in sequentiellm Buddy-Bereich gemäß binärem Buddy-System
  - Zusammenfassung zweier Buddies der Größe  $2^n \Rightarrow 2^{n+1}$  ( $n \geq 0$ )



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

306.

## Starburst: Speicherorganisation zur Realisierung Langer Felder

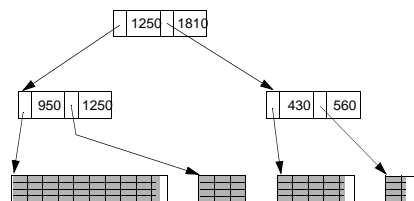


- **Aufbau eines Langen Feldes**
  - Deskriptor des Langen Feldes (< 255 Bytes) ist in Relation gespeichert
  - Long Field ist aufgebaut aus einem oder mehreren **Buddy-Segmenten**, die in großen vordefinierten **Buddy-Bereichen** fester Länge auf Platte angelegt werden
  - Buddy-Segmente enthalten nur Daten und keine Kontrollinformation
  - Segment besteht aus 1, 2, 4, 8, ... oder 2048 Seiten ( $\Rightarrow$  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

307.

## Speicherallokation mit variablen Segmenten

- ❑ Verallgemeinerung des EXODUS- und STARBURST-Ansatzes in EOS
  - 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
- ❑ Prinzipielle Repräsentation



- ❑ Die Größen der einzelnen Segmente können sehr stark variieren
- ❑ Verarbeitungseigenschaften
  - die guten operationalen Eigenschaften der beiden zugrundeliegenden Ansätze können erzielt werden
  - Reorganisation möglich, falls benachbarte Segmente sehr klein (Seite) werden

308.

## Zusammenfassung

- ❑ Freispeicherinformation auf verschiedenen Ebenen erforderlich: Gerät, Datei, Seite
- ❑ Ziele bei der externspeicherbasierten Adressierung
  - Kombination der Geschwindigkeit des direkten Zugriffs mit der Flexibilität einer Indirektion
  - Satzverschiebungen in einer Seite ohne Auswirkungen
  - $\Rightarrow$  TID-Konzept oder Zuordnungstabelle
  - Einmalige Benutzung einer OID
  - $\Rightarrow$  Einführung eines Zählers
- ❑ Abbildung von Sätzen
  - Speicherung variabel langer Felder
  - dynamische Erweiterungsmöglichkeiten
  - Berechnung von Feldadressen
  - Speicherung von sehr langen Feldern

309.