5 Row-based Record Management (Klassische Satzverwaltung)

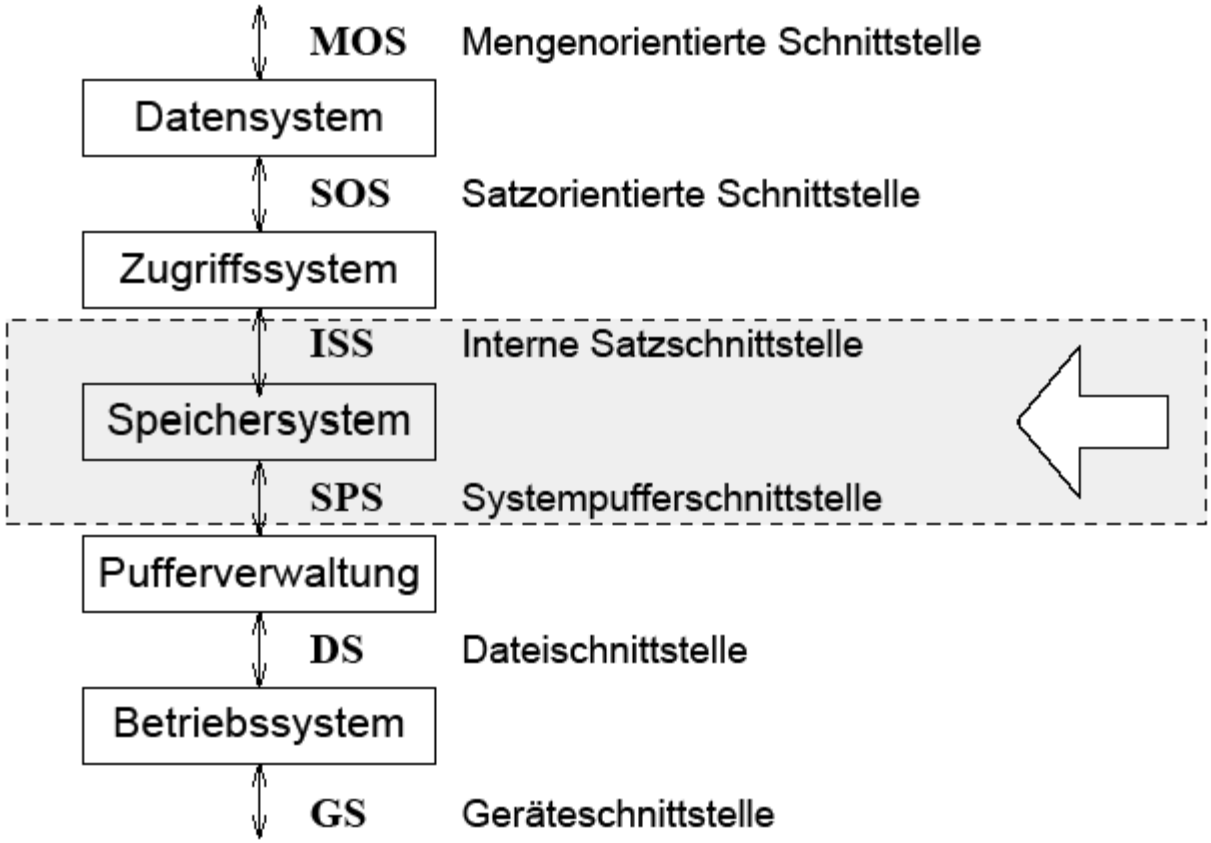
Abbildung ganzer Sätze auf Seiten

Aufbau und Speicherungsstrukturen für Sätze

Satzadressierung

- Zuordnungstabelle
- TID-Konzept

Freispeicherverwaltung


MOS
Mengenorientierte Schnittstelle
Datensystem
SOS
Satzorientierte Schnittstelle
Zugriffssystem
ISS
Interne Satzschnittstelle
Speichersystem
SPS
Systempufferschnittstelle
Pufferverwaltung
DS
Dateischnittstelle
Betriebssystem
GS
Geräteschnittstelle

Key/ Value Approach

- **Get(Key) →(object, context)**
  - Returns a list of data objects associated with key
  - More than one object only if there was a conflict
  - Returns a context
- **Put(key, context, object)**
  - Determine where replicas should be placed for associated key
  - Write the replicas to the disk
- Context
  - Encodes system metadata that the caller is not aware of
  - Includes versioning information

Differentation

- Key – either an internal key (TID/RID) or an application key (in the worst case: composite key)
- Values – either known to the system or only known to the application

Assumption for now

- key is “internal” / value structure is known to the system

wichtig

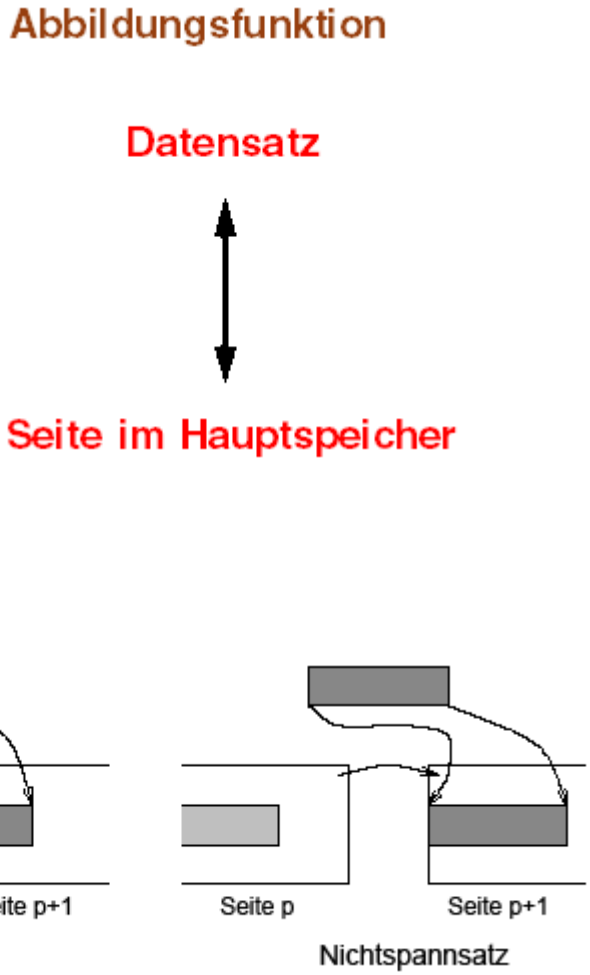
- bisher: Seite fester Länge als ‘Verarbeitungseinheit’
- jetzt: Datensatz beliebiger Länge als ‘Verarbeitungseinheit‘

Entkopplung von

- systemvorgegebenen Verarbeitungseinheiten (Seiten) (physischer Satz)
- Datenstrukturen einer Anwendung (Sätze) (logischer Satz)

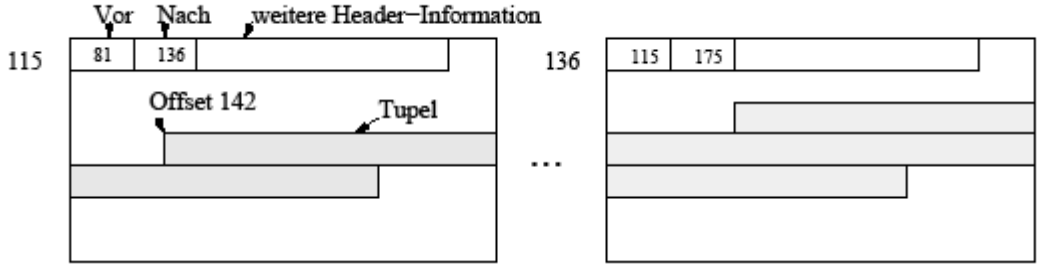
Erinnerung

- Spannsatz
- Nicht-Spannsatz



Verkettung

- Seiten sind untereinander durch doppelt verkettete Listen verbunden
- Aufzeichnung freier Seiten: Freispeicherverwaltung

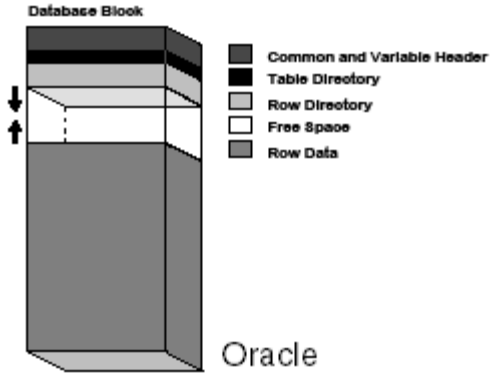


Seiten-Header

- Informationen über Vorgänger- und Nachfolger-Seite
- eventuell auch Nummer der Seite selbst
- Informationen über Typ der Sätze (Table Directory)
- freier Platz

Row Directory

- TID-Verweise



Definition “Satz”

- Zusammenfassung von Daten, die zu einem Gegenstand, einer Person, einem Sachverhalt usw. einer Anwendung gehören und Eigenschaften des Gegenstands widergeben.
- Sätze sind aus Feldern zusammengesetzt (Komponenten der Struktur in C).

Merke

- Die Strukturierung eines Satzes ist für die Speicherverwaltung auf dieser Ebene irrelevant.
- In dieser Schicht ist ein Satz nur eine Bytefolge, deren Länge aber nicht mehr vom System, sondern von der Anwendung bestimmt wird!
- Eine Datei ist auf dieser Abstraktionsebene eine (lineare) Folge von Sätzen fester oder variabler Länge.

Aufgabe des Record-Managers

- physische Abspeicherung / Organisation von Sätzen in Seiten
- Operationen: Lesen, Einfügen, Modifizieren, Löschen

Anforderungen für Verarbeitungseffizienz und -flexibilität

- Jeder Satz wird durch ein Satzkennzeichen (SKZ oder OID) identifiziert
- möglichst platzsparend Speicherung (Speicherökonomie)
- Erweiterbarkeit des Satztyps muss im laufenden Betrieb möglich sein
- einfache Berechnung der satzinternen Adresse des n-ten Feldes (bei Zugriff nur auf einen Teilaspekt der Sätze)

Satzbeschreibung

- Satz- und Zugriffspfadbeschreibung im Katalog
- besondere Methoden der Speicherung
  - Blank-/Nullunterdrückung
  - Zeichenverdichtung
  - kryptographische Verschlüsselung
  - Symbol für undefinierte Werte
- Organisation
  - n Satztypen pro Segment
  - m Sätze verschiedenen Typs pro Seite
  - Satzlänge < Seitenlänge

Attribut- /Feldbeschreibung

- Name (meist Unterschied zwischen internem Feldnamen und externem Attributnamen
- Charakteristik (fest, variabel, multipel)
- Länge
- Typ (alpha-numerisch, numerisch, gepackt, ...)
- besondere Methoden bei der Speicherung (z.B. Nullenunterdrückung, Zeichenverdichtung, Kryptographie etc.)
- ggf. Symbol für den undefinierten Wert (falls nicht als Segment- oder Systemkonstante global definiert).

Die Formatbeschreibung steuert alle Operationen auf Sätzen

Satztypen

- Typischerweise viele Sätze mit gleichem Aufbau, d.h. gleichen Felder einmalige Beschreibung im Datenwörterbuch für alle
- **Satztyp:** Menge von Sätzen mit gleicher Struktur bekommt einen Namen
- Jeder Satz muss beim Abspeichern einem Satztyp zugeordnet werden (Sätze ohne Typ sind nicht erlaubt).

Länge der Sätze eines Satztyps

- fest, wenn alle Felder feste Länge haben oder bei Feldern variabler Länge immer die Maximallänge reserviert wird.
- variabel sonst

Problem

- In welcher Seite wird ein **Satz abgelegt**, und wie kann anschließend dieser **Satz wieder gefunden** werden, auch wenn zwischenzeitlich etliche andere Sätze gelöscht und eingefügt wurden?
- siehe Satzadressierung !!!

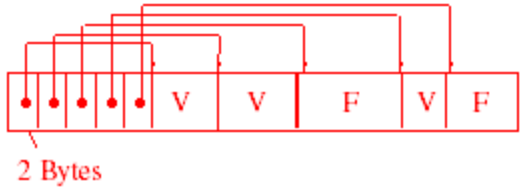
Annahmen

- variable Satzlänge (allgemeinerer Fall)
- Reihenfolge der Abspeicherung muss nicht Reihenfolge des Einfügens sein
- direkter Zugriff auf einzelne Sätze über ihre Satzadresse
- Ein Satz sollte in einer Seite ablegbar sein: L_R £ L_S - L_{SK} (Standard)
- Mehrere Satztypen pro Seite sollen möglich sein.

Konkatenation von Feldern fester Länge

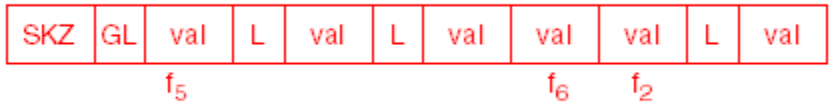
- speicheraufwendig
- unflexibel

Zeiger im Vorspann



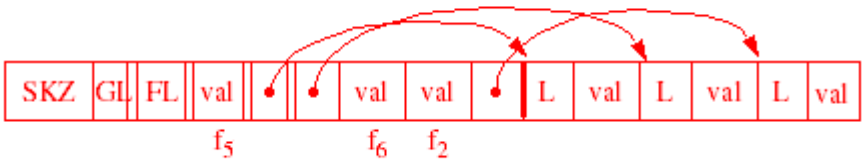
- unflexibel

eingebettete Längenfelder



- dynamische Erweiterung möglich
- aber: zusätzliches Wissen notwendig: f5 | v | v | f6 | f2 | v |

eingebettete Längenfelder mit Zeigern

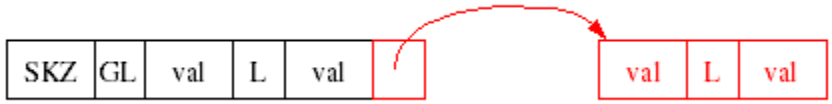


- Adresse des n-ten Attributes wird berechnet

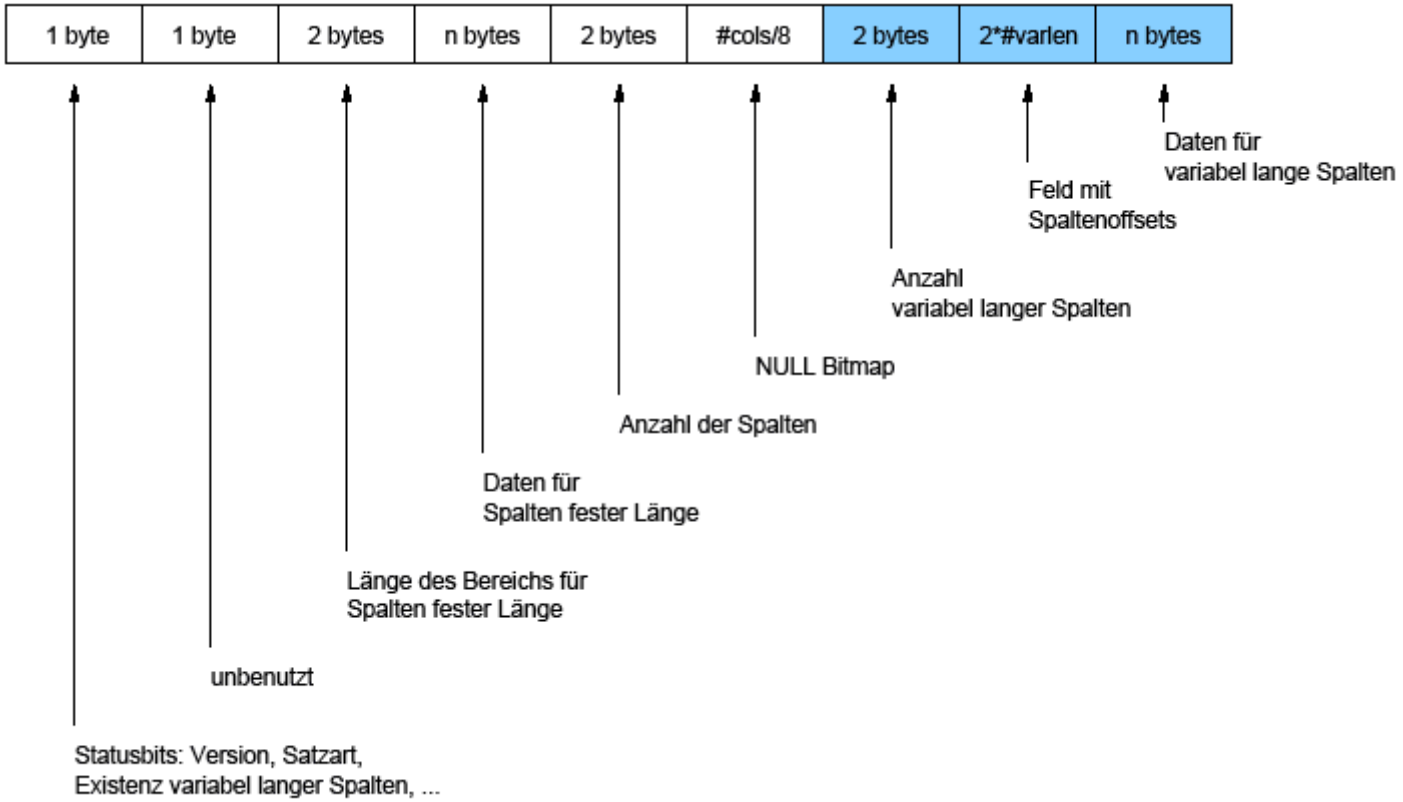
Problem

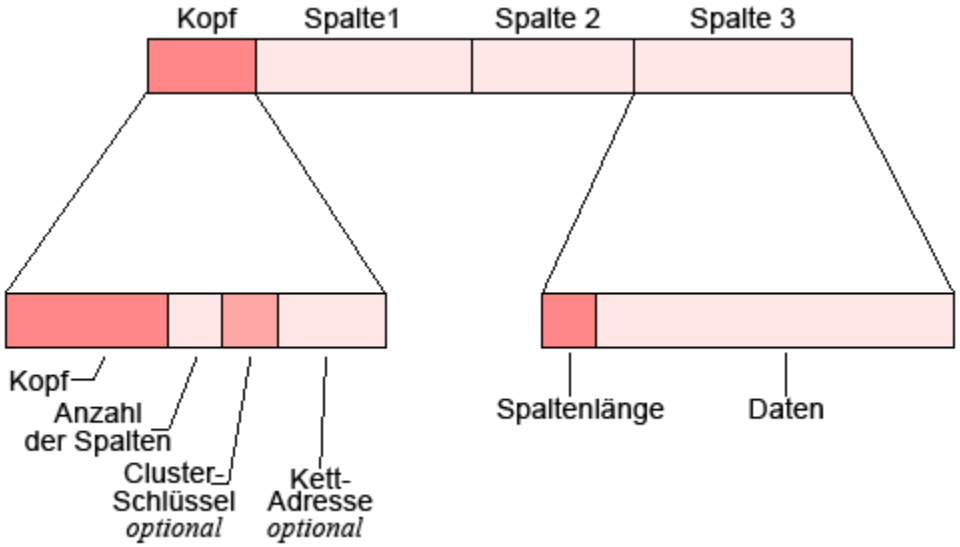
- dynamisches Wachstum/variable Länge
  - Ausdehnung und Schrumpfung in einer Seite
  - Überlaufschemata
  - Garbage Collection
- strikt zusammenhängende Speicherung von Sätzen
  - evtl. häufige Umlagerung bei hoher Änderungsfrequenz
  - Vorteile für indirekte Adressierungsschemata

Aufspaltung des Satzes



- Ordnung nach Referenzhäufigkeiten
- Verbesserung der Clusterbildung
- Wiederholter Überlauf möglich
- wird unvermeidlich bei der Einbeziehung von Attributen vom Typ TEXT oder BILD (sofern nicht als BLOB/CLOB abgespeichert)


1 byte
1 byte
2 bytes
n bytes
2 bytes
#cols/8
2 bytes
2*#varlen
n bytes
Statusbits: Version, Satzart, Existenz variabel langer Spalten, ...
unbenutzt
Länge des Bereichs für Spalten fester Länge
Daten für Spalten fester Länge
Anzahl der Spalten
NULL Bitmap
Anzahl variabel langer Spalten
Feld mit Spaltenoffsets
Daten für variabel lange Spalten



Kettadresse für Row-Chaining

- Verteilung von Verkettung zu großer Datensätze (>255 Spalten) über mehrere Blöcke
- row id = (data object identifier, data file identifier, block identifier, row identifier)

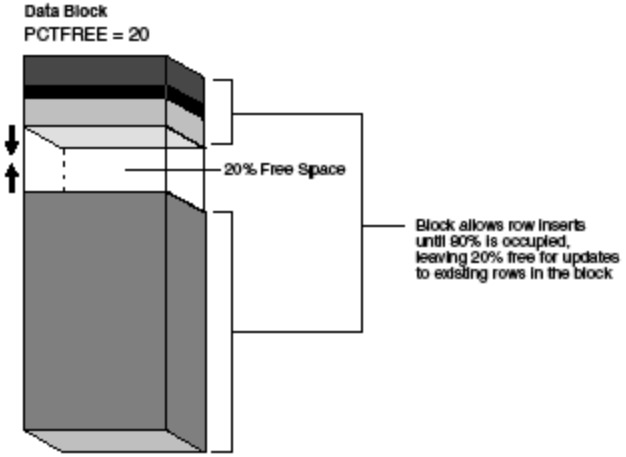
Syntax für Tabellendefinition

```
CREATE TABLE TABELLE ( ...)
PCTFREE 20 PCTUSED 40
STORAGE (
   INITIAL 10MB, NEXT 2MB,
   MINEXTENTS 1, MAXEXTENTS 20,
   PCTINCREASE 0, FREELISTS 3 )
TABLESPACE USER_TBLSPACE;
```

- initial, next: Größe des ersten bzw. der weiteren Extents (Default: 5 Blöcke)
- minextents, maxextents: Anzahl der mind. bzw. max. zu allokierenden Extents
- pctincrease: prozentuale Vergrößerung der nachfolgenden Extents (0: gleich große Extents)
- freelists: Anzahl der Freispeicherlisten (insb. für paralleles Einfügen)
- tablespace: Zuordnung zum Tablespace
- pctfree: Datenblockanteil, der nicht für insert-Operationen genutzt werden soll (Reservebereich für update); Default 10
- pctused: Grenze, bei der ein zuvor bis zu pctfree gefüllter Block wieder für insert genutzt werden darf; Default 40

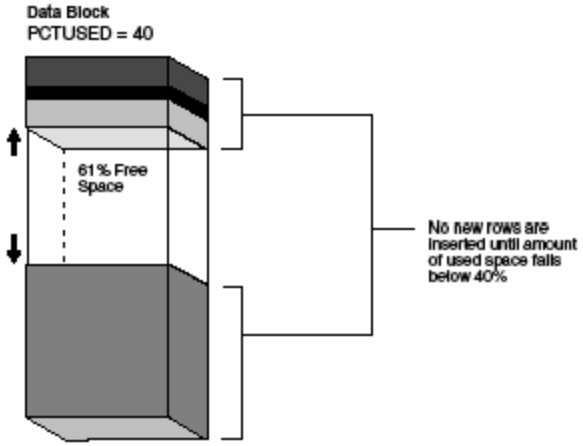
❑ PCTFREE

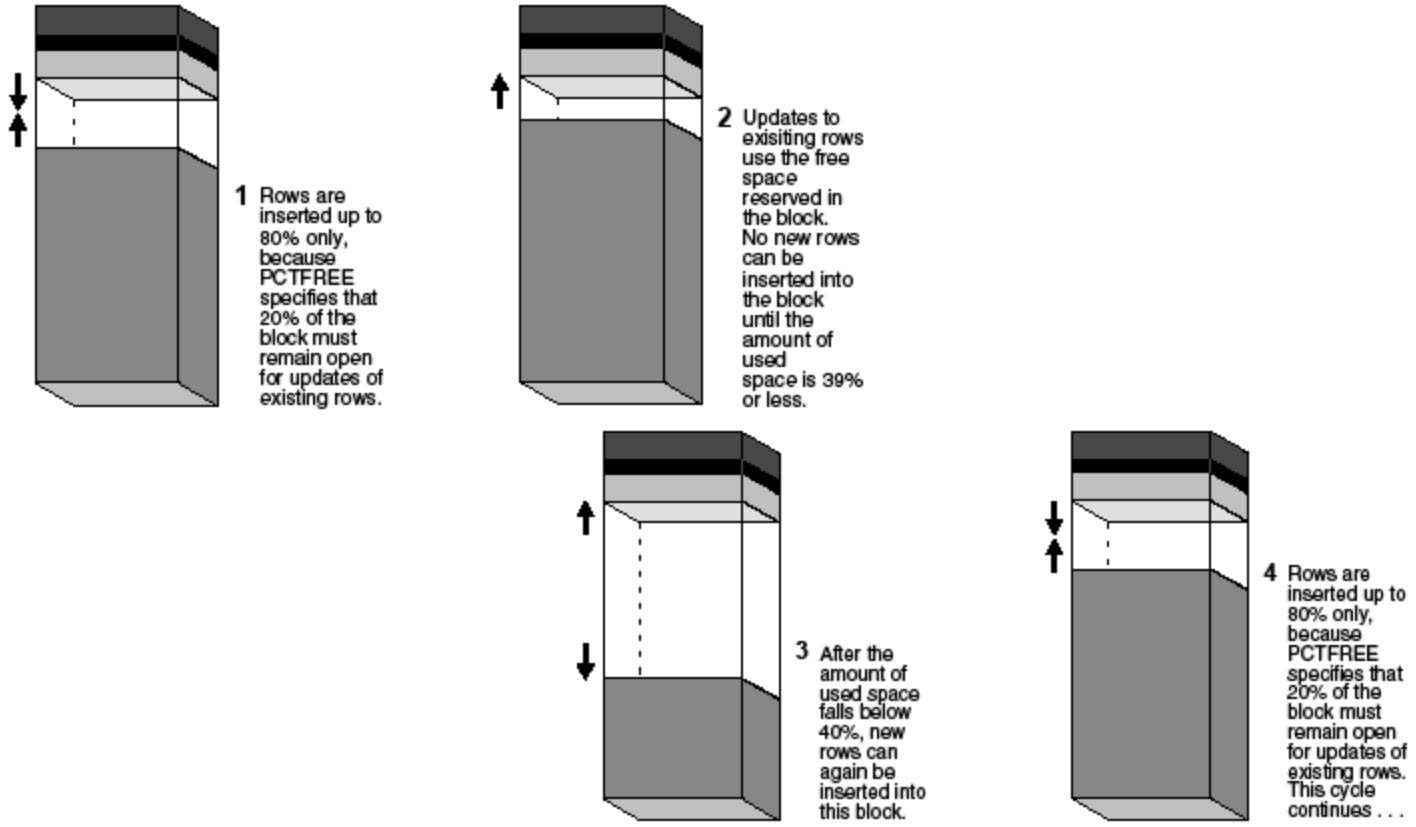
- Anteil am Block, der für Updates an existierenden Datensätzen freigehalten wird
- erlaubt neue Datensätze bis Füllgrad > 1-DPCTFREE



❑ PCTUSED

- Füllgrad eines Blocks, ab dem neue Sätze in den block wieder eingefügt werden dürfen
- keine neuen Datensätze, falls Füllgrad > PCTUSED




1 Rows are inserted up to 80% only, because PCTFREE specifies that 20% of the block must remain open for updates of existing rows.
2 Updates to exisiting rows use the free space reserved in the block. No new rows can be inserted into the block until the amount of used space is 39% or less.
3 After the amount of used space falls below 40%, new rows can again be inserted into this block.
4 Rows are inserted up to 80% only, because PCTFREE specifies that 20% of the block must remain open for updates of existing rows. This cycle continues . . .

... Individuelle Speicherrepräsentation

...(**PARTITION** LINEITEM2000 **VALUES LESS THAN** '2001-01-01' **)**,
PCTFREE 0 **TABLESPACE** DATA2000
STORAGE (**INITIAL** 2M **NEXT** 4M **PCTINCREASE** 0),
(**PARTITION** LINEITEM2001 **VALUES LESS THAN** '2002-01-01'),
PCTFREE 30 **PCTUSED** 60 **TABLESPACE** DATA2001
STORAGE (**INITIAL** 16K **NEXT** 16K **PCTINCREASE** 0.1), ...

- ... pro Partition

Typischerweise passen mehrere Sätze in einen Seite (Satzlängen 100 - 1000 Bytes)

Blockungsfaktor: Anzahl der Sätze pro Seite

Annahme: keine blockübergreifenden Sätze ("spanned records"), d.h. jeder Satz wird vollständig in einer Seite abgelegt

- feste Satzlänge
  - Blockungsfaktor aus Seitengröße und Satzlänge berechenbar
  - meist ungenutzter Speicherplatz am Ende einer Seite
- variable Satzlänge
  - Blockungsfaktor ändert sich von Seite zu Seite

Problem

- langfristige Speicherung der Datensätze
- Vermeiden von Technologieabhängigkeiten
- Unterstützung von Migration u. a.

Satzadressen

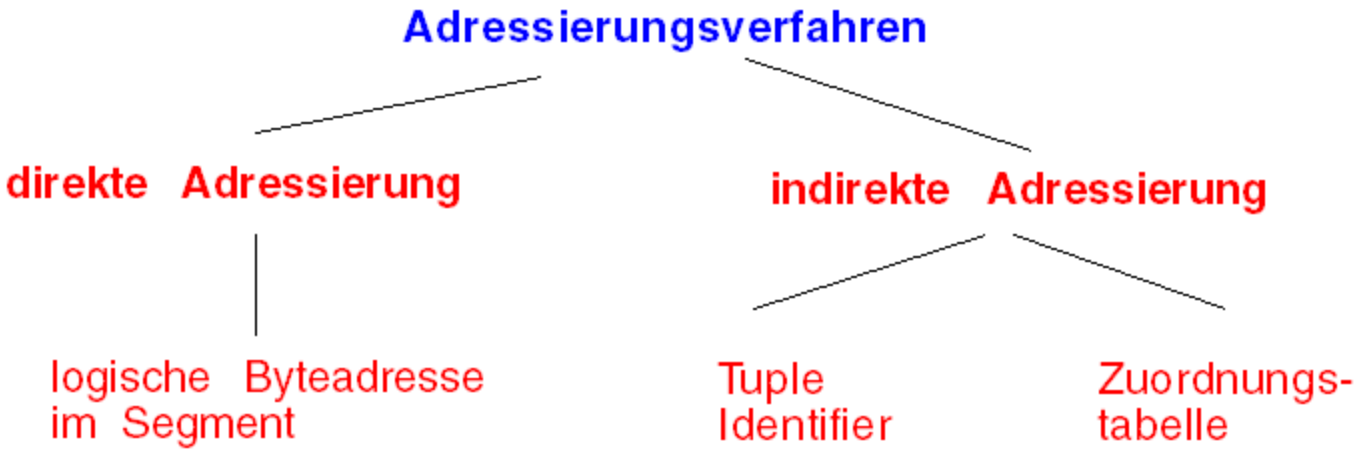
- Satzadressen werden beim Einfügen von Sätzen vergeben und können später zum Zugriff auf die Sätze verwendet werden.

Ziele der Adressierungstechnik

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

Allgemeine Form einer Satzadresse

- DBID, SID, TID und ggf. Relationenkennzeichnung (RID)
- Relation vollständig in einem Segment gespeichert: TID DBID, SID im DB-Katalog
- Relation in mehreren Segmenten: SID, TID


Adressierungsverfahren
direkte Adressierung
indirekte Adressierung
logische Byteadresse im Segment
Tuple Identifier
Zuordnungs-tabelle

Laufende Nummer des Satzes

- Instabil!
  Die laufende Nummer, und somit die Satzadresse ändert sich bei Einfügungen und Löschvorgängen, sowie bei Änderungen in der Abspeicherungsreihenfolge.

Blocknummer und Byte-Position innerhalb des Blocks

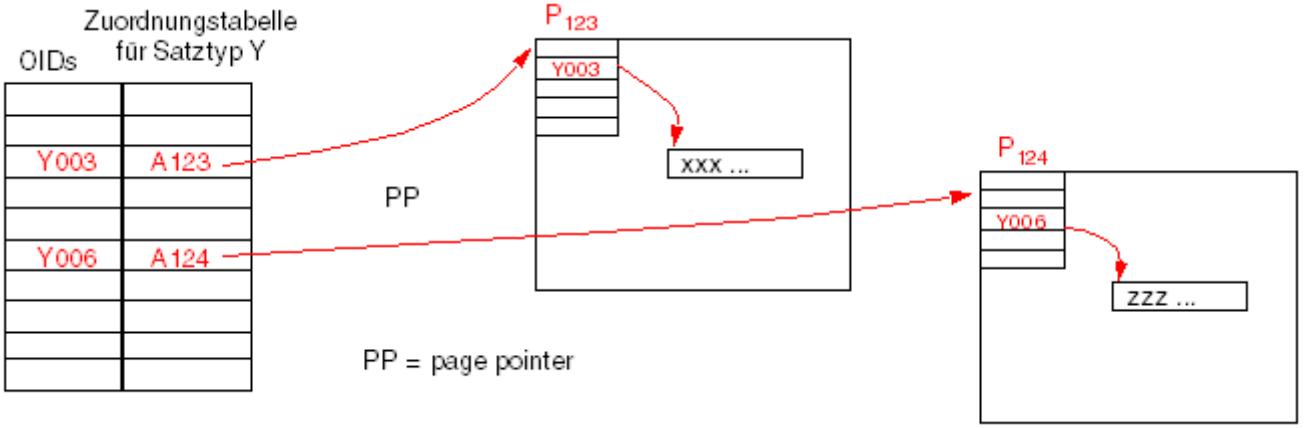
- Instabil!
  Ändert ein Satz innerhalb des Blocks seine Länge, müssen i.allg. die anderen Sätze verschoben werden.
- Wird der Satz selbst zu lang für den Block, so muss er in einen anderen Block verlegt werden; dann ändert sich auch die Blocknummer.

Adressierung in Segmenten

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

Satzadressierung über Zuordnungstabelle (vollständige Indirektion)

- Idee:
  Verwaltung eines Felds (Array) in aufeinanderfolgenden Seiten des Segments, das zu jeder Satznummer (Index) die Seitennummer angibt.
  - Einfügen eines Satzes
    es wird grundsätzlich eine neue Satznummer (Datenbankschlüssel DBK) durch das Datenbanksystem vergeben
  - Löschen eines Satzes
    der Eintrag der entsprechenden Satznummer wird als ungültig gekennzeichnet
  - Zugriff auf einen Satz
    erfordert zwei Seitenzugriffe: einen für das Feld, einen für die Seite mit dem Satz selbst
  - Verlagerung eines Satzes
  - in eine andere Seite
    nur der Eintrag des wird Satzes geändert; die Satznummer bleibt unverändert - der Satz ist also über die Satznummer weiterhin auffindbar
  - innerhalb einer Seite
    der Eintrag im Feld muss auch geändert und wieder auf Platte geschrieben werden (zusätzliche E/A-Operation)
  - die Zuordnungstabelle kostet selbst einigen Speicherplatz



Merke: der DBK ist eine "nicht sprechende" Adresse (a la Telefonnummer)

- Der Database-Key wird gebildet aus einer Satztypbezeichnung r und einer Folgenummer f. r und f identifizieren den Satz während seiner Lebenszeit in der DB.
- Es wird auf die Seite P_k im Segment S_i verwiesen.

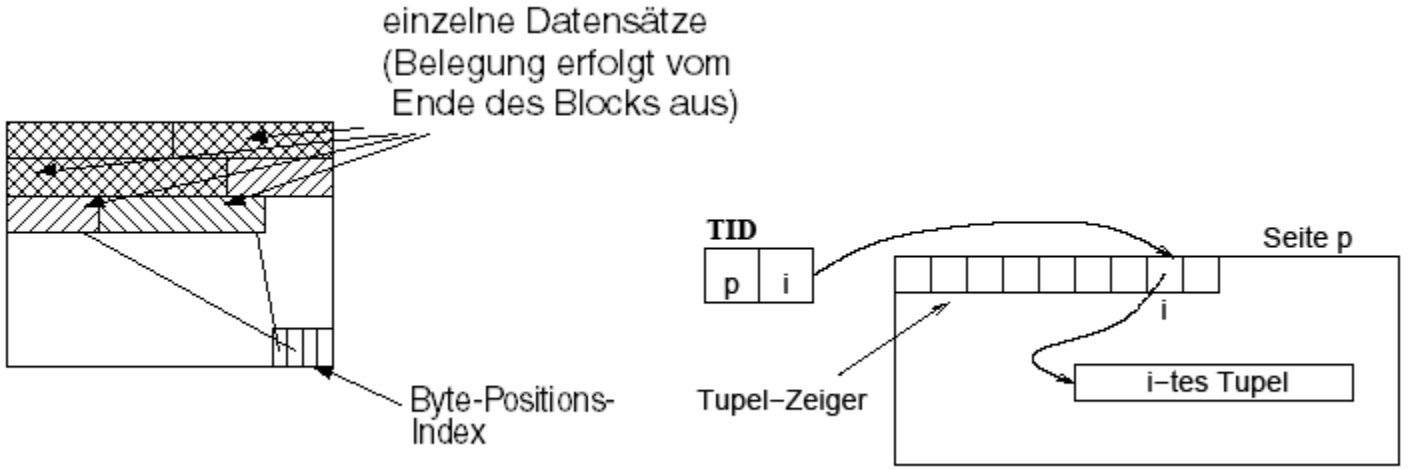
Problem: Wo wird die Zuordnungstabelle abgespeichert?

- am Anfang - wie erweitern?
- am Ende? wie den Datenbereich erweitern?
- in einem eigenen Segment?

Satzadressierung über Indirektion innerhalb eines Blocks

- Array mit Byte-Positionen der Sätze in diesem Block
- Adresse ist das Paar bestehend aus Blocknummer und Index in diesem Array ("TID = Tuple IDentifier")

- Für den Zugriff auf einen Satz wird nur ein Blockzugriff benötigt.
- Struktur eines Blocks:



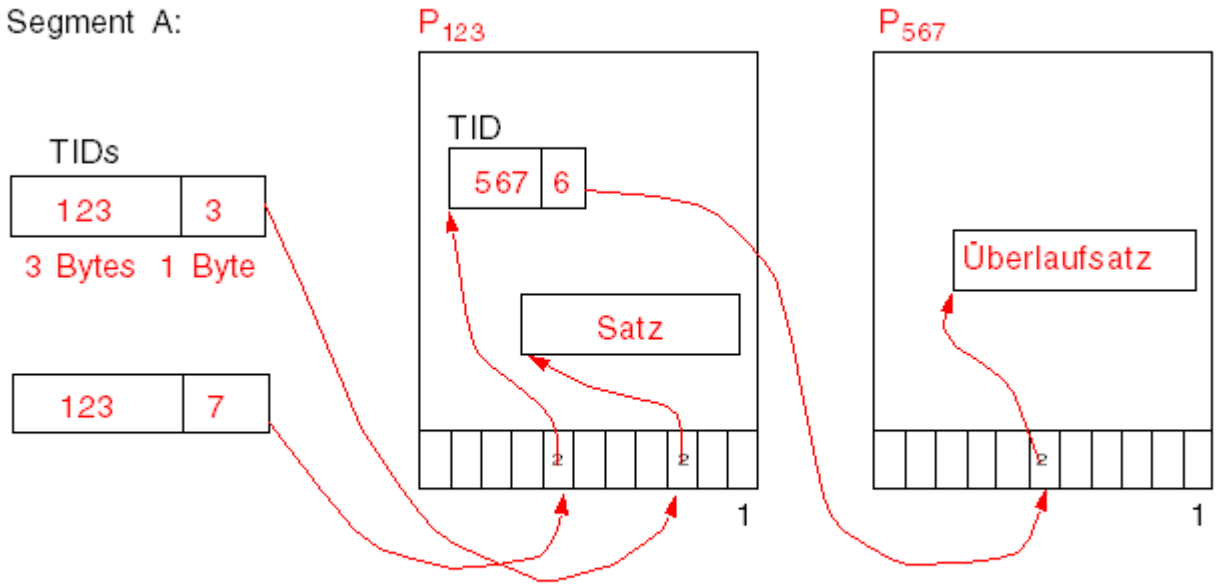
Löschen eines Satzes

- der entsprechende Eintrag des Block-Arrays wird als ungültig gekennzeichnet
- Alle anderen Sätze im selben Block können verschoben werden, um den freien Platz zu maximieren
  es ändern sich nur ihre Anfangsadressen im Block-Array.
- Alle Satzadressen bleiben stabil.

Update-Operation auf einen Datensatz

- es kann sich die Länge eines Datensatzes verändern !!!
- Datensatz schrumpft oder wird größer (ohne Überlauf):
- Alle Sätze werden innerhalb des Blocks verschoben und der Byte-Positionsindex wird angepasst.
- Datensatz wird größer und der freie Platz im Block reicht nicht mehr für die Speicherung des jetzt größeren Datensatzes (Überlauf):
- Verschiebung des Datensatzes in einen anderen Block!

Verlagerung eines Satzes



Vorgehen

- Im alten Block verbleibt an der Stelle des Originalsatzes eine neue Satzadresse, die auf den neuen Block verweist.
  - In diesem (seltenen) Fall müssen also zwei Blöcke gelesen werden.
  - Wird der Satz ein weiteres Mal verlagert, so wird die Satzadresse im ersten Block verändert. Dadurch bleibt es bei maximal einer Indirektion.

Trick

- Die Länge der Überlaufkette ist immer kleiner oder gleich 1, d.h. ein Überlaufsatz darf nicht weiter “überlaufen”, sondern muss von seiner Hausadresse neu plaziert werden.

Vorteile

- Keine Zuordnungstabelle (Umsetztabelle)
- Ein Satz kann innerhalb einer Seite und über Seitengrenzen hinweg verschoben werden, ohne dass der TID sich ändert.

Situation

- Beispiel: Wo findet sich ausreichend Platz, um einen neuen Datensatz aufzunehmen?

Freispeicherverwaltung (FPA, Free Place Administration):

- In einer Tabelle F_i zum Segment S_i wird für jede Seite s_k angegeben, wieviele Bytes in ihr noch frei sind.
- $F_i(k) = n$ <-> In Seite s_k des Segmentes S_i sind n Bytes frei.

Problem

- Wie groß wird die FPA-Tabelle?
- Wo (in welchen Seiten eines Segmentes) wird die FPA-Tabelle abgespeichert?

Speicheraufwand für Freispeicherverwaltung

- mit
  - L_S = Seitenlänge
  - L_{SK} = Länge Seitenkopf (page header) für die beschreibenden Informationen einer Seite
  - L_F = Länge eines Eintrags (im allgemeinen 2 Byte)
- ergibt sich
  - $k = (L_S - L_{SK}) / L_F$ = Anzahl der Einträge pro Seite
  - s = Anzahl der Seiten im Segment
  - $n = s / k$ = belegte Seiten im Segment

Lokation für Freispeicherverwaltung

- Äquidistante Verteilung der Tabellenseiten gemäß (i*k +1) mit i = 0, 1, 2, ..., n - 1
  d.h. eine Tabellenseite steht vor den k Seiten, für die sie die Freispeicherinformation enthält.
  - Vorteil: Segment kann problemlos erweitert werden.
  - Nachteil: Suche nach freiem Speicher “hüpft” durch das Segment.
- Bei direkter Seitenadressierung werden deshalb für FPA-Tabelle üblicherweise die ersten n Seiten eines Segments belegt.
  - Nachteil: Erweiterung des Segments
- Bei indirekter Seitenadressierung befindet sich die Freispeicherinformation mit in der Seitentabelle

Seite	1	2	3	. . .	s
Block	i	j	k	. . .	r
Freier Platz	F(1)	F(2)	F(3)	. . .	F(s)

Anforderungen

- idealerweise keine Größenbeschränkung
- Verkürzen, Verlängern und Kopieren
- Suche nach vorgegebenem Muster, Längenbestimmung, ...

Erweiterte Anforderungen

- Effiziente Speicherallokation und -freigabe für Feldgrößen von bis zu 100MB - 2GB (Sprache, Bild, Musik oder Video)
- hohe E/A-Leistung:
  Schreib- und Lese-Operationen sollen E/A-Raten nahe der Übertragungsgeschwindigkeit der Magnetplatte erreichen

Verarbeitungsprobleme

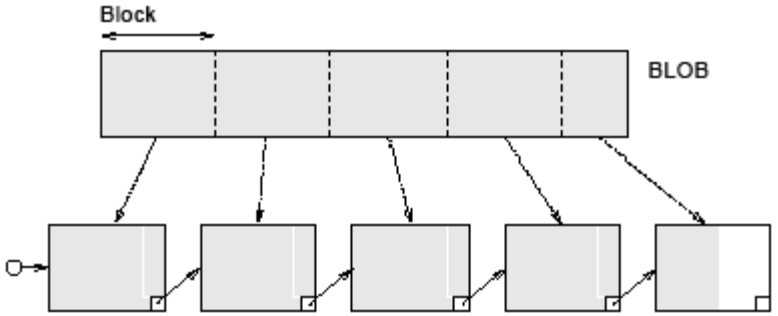
- Ist Objektgröße vorab bekannt?
- Gibt es während der Lebenszeit des Objektes viele Änderungen?
- Ist schneller sequentieller Zugriff erforderlich? - ...

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

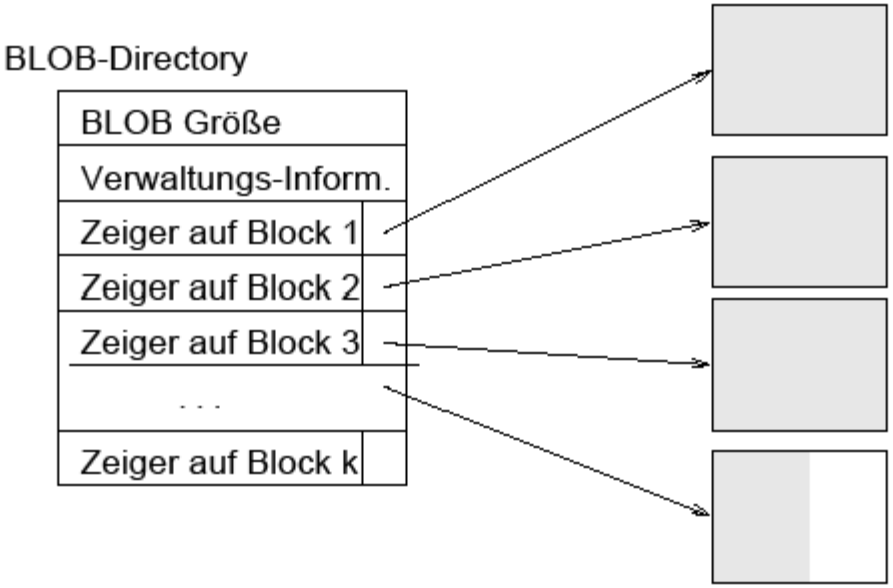
Abbildung auf Externspeicher

- seitenbasiert (Seite als Einheit)
- verstreute Sammlung von Seiten
- segmentbasiert (mehrere Seiten)
- Segmente fester Größe (EXODUS)
- Segmente mit einem festen Wachstumsmuster (STARBURST)
- Segmente variabler Größe (EOS)



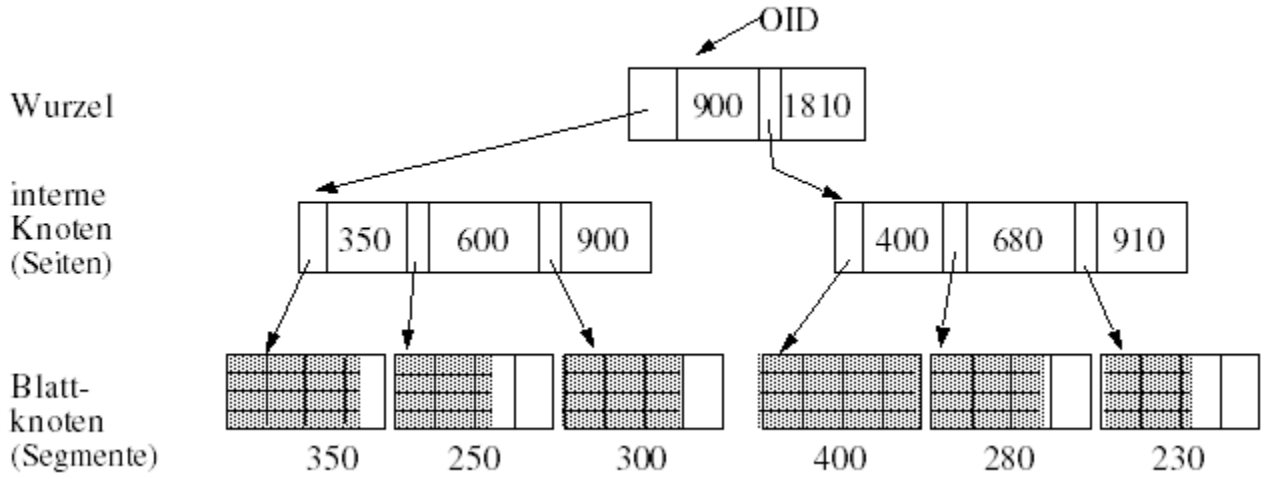
Speicherverfahren

- zentrales BLOB-Verzeichnis mit Zeiger auf die einzelnen Blöcke
  - kann in ursprünglichen Satz eingebettet werden



Speicherverfahren

- Daten werden in Seiten / (kleinen) Segmenten fester Größe abgelegt



Nutzung

- Baumorganisierte Zugriffsstruktur (B*-Baum o.ä)

Baumstruktur

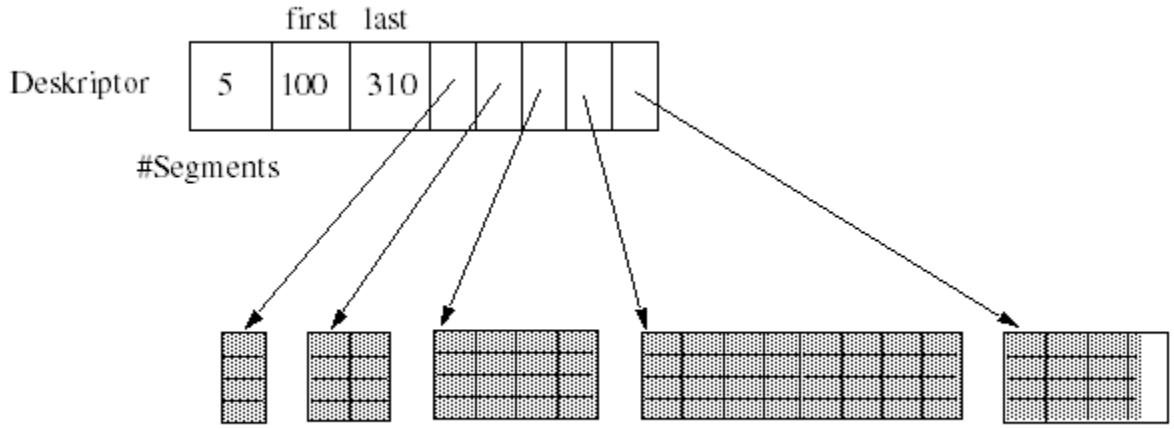
- Blätter sind Segmente fester Größe (hier 4 Seiten a 100 Bytes)
- interne Knoten und Wurzel sind Index für Bytepositionen
- interne Knoten und Wurzel speichern für jeden Kind-Knoten Einträge der Form (Zähler, Seitennummer)
  - Zähler enthält die maximale Bytenummer des jeweiligen Teilbaums (links stehende Seiteneinträge zählen zum Teilbaum)
  - Zähler im weitesten rechts stehenden Eintrag der Wurzel enthält Länge des Objektes
  - Repräsentation sehr langer dynamischer Objekte
  - bis zu 1GB mit drei Baumebenen (selbst bei kleinen Segmenten)
  - Speicherplatznutzung typischerweise ~ 80 %

Bewertung

- bei bekannter Verarbeitungscharakteristik Wahl geeigneter Segmentgrößen möglich
- Einfügen von Bytefolgen einfach und überall möglich
- schlechteres Verhalten bei sequentiellem Zugriff

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

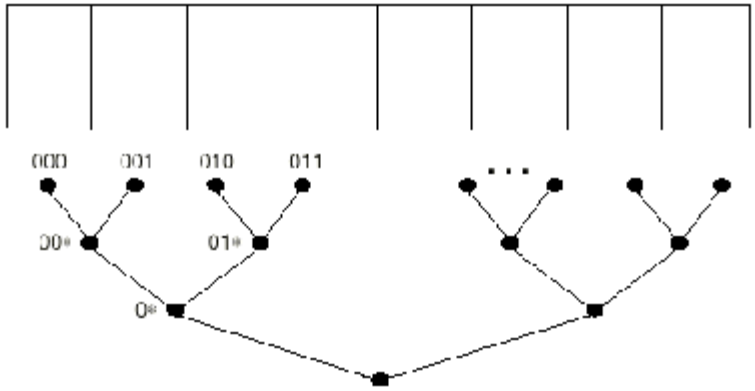


...bei vorab bekannter Objektgröße G

- G ≤ MaxSeg: es wird ein einzelnes Segment angelegt
- G > MaxSeg: es wird eine Folge maximaler Segmente angelegt; das letzte Segment wird auf verbleibende Objektgröße gekürzt

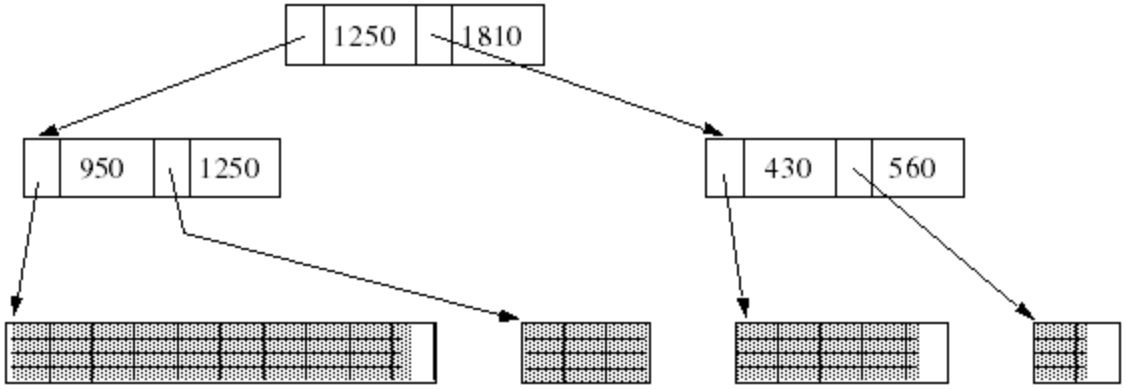
bei unbekannter Objektgröße: Allokation von Buddy-Segmenten

- Wachstumsmuster der Segmentgrößen gemäß: 1, 2, 4, ..., 2^n
- Seiten werden jeweils zu einem Buddy-Segment zusammengefasst



Repräsentation

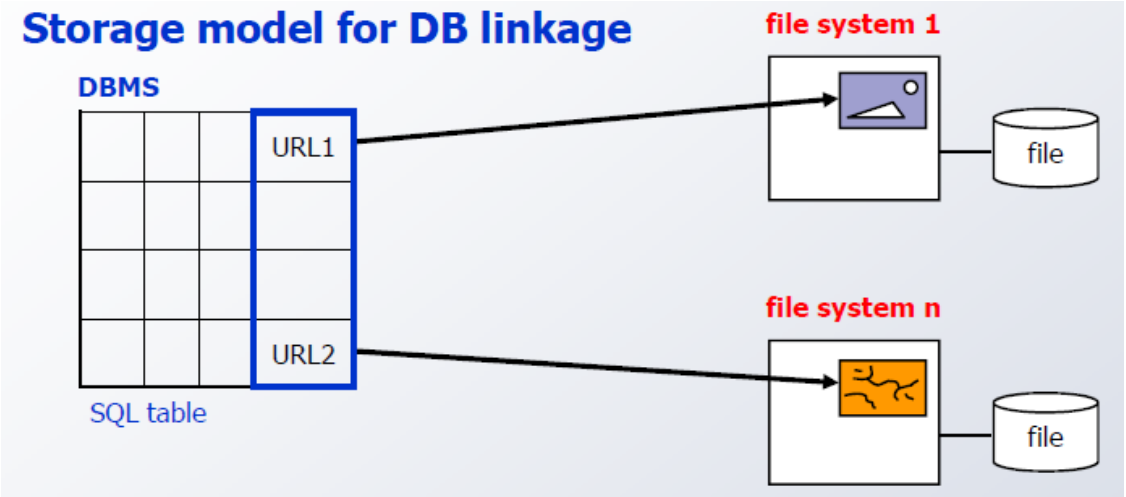
- Objekt ist gespeichert in einer Folge von Segmenten variabler Größe
- Segment besteht aus Seiten, die physisch zusammenhängend auf Externspeicher angeordnet sind
- nur die letzte Seite eines Segmentes kann freien Platz aufweisen



- ererbt die guten operationalen Eigenschaften der beiden Vorgängeransätze

Motivation

- More and more data is (still) stored in files
- Many applications are working in a file-based fashion, native file access has to be supported
  - CAD solutions,
  - Multimedia objects (movies)
  - HTML and XML files



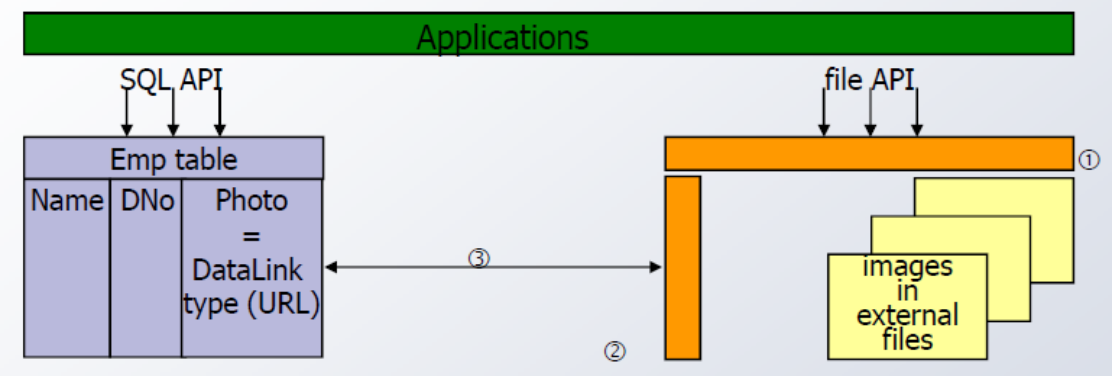
Drawbacks

- File systems do not support classical DB features
  - Referential integrity
  - Fine-grained access control
  - Consistent backup and recovery
  - Transactional consistency/isolation/…
  - Sophisticated support for an efficient search
- DBMS are tuned to work on well-structured (and potentially) large datasets

Goal

- Combination of file systems and DBMSs as best-of-breed approach

Application Support



DataLinks File System Filter (DLFF)

- Enforces referential integrity when files are renamed or deleted
- Enforced db-centric access control when a file is opened
- File API remains unchanged in the read/write parth for external files
- DLFF does not reside in the read/write parth for external files

DataLinks FileManager (DLFM)

- Executes Link/Unlink operations under transaction protection
- Guarantees referential integrity
- Supports coordinated backup/recovery

DBMS manages/coordinates operations on external files

- Viel referenced URLSs or via DLFM API

LINK CONTROL

- NO LINK CONTROL: URL-Format des Datalinks; keine weitere Kontrolle
- FILE LINK CONTROL: existierende Datei muss referenziert werden; Art der Kontrolle durch die weiteren Optionen bestimmt

Integrität (INTEGRITY CONTROL OPTION)

- INTEGRITY ALL: referenzierte Dateien können nur über SQL gelöscht oder umbenannt werden
- INTEGRITY SELECTIVE: referenzierte Dateien können mittels File-Manager-Operationen gelöscht oder umbenannt werden, solange kein Datalinker vorhanden ist
- INTEGRITY NONE: referenzierte Dateien können ausschließlich mittels File-Manager-Operationen gelöscht oder umbenannt werden -> nicht verträglich mit FILE LINK CONTROL

Lese-Zugriff (READ PERMISSION OPTION)

- READ PERMISSION FS: Leserecht für referenzierte Dateien wird durch den File-Manager bestimmt
- READ PERMISSION DB: Leserecht für referenzierte Dateien wird über SQL bestimmt

Schreibzugriff (WRITE PERMISSION OPTION)

- WRITE PERMISSION FS: Schreibrecht für referenzierte Dateien wird durch den File-Manager bestimmt
- WRITE PERMISSION BLOCKED: kein Schreibzugriff auf referenzierte Dateien, es sei denn, es existiert implementierungsabhängiger Mechanismus
- WRITE PERMISSION ADMIN [NOT] REQUIRING TOKEN FOR UPDATE: Schreibrecht für referenzierte Dateien durch SQL bestimmt

Wiederherstellung (RECOVERY OPTION)

- RECOVERY YES: mit Datenbankserver koordinierte Recovery (Datalinker-Mechanismus)
- RECOVERY NO: keine Recovery auf referenzierten Dateien

Auflösen der Link-Kontrolle (UNLINK OPTION)

- ON UNLINK RESTORE: vor der Herstellung des Links bestehende Rechte (Ownership, Permissions) werden durch den File-Manager bei Auflösung des Links (Unlink) wiederhergestellt
- ON UNLINK DELETE: Löschung bei Unlink
- ON UNLINK NONE: keine Auswirkungen auf die Rechte bei Unlink

Neue SQL-Funktionen

- Konstruktor: DLVALUE, ...
- (Komponenten von) URLs: DLURLCOMPLETE,

Beispiele

```
INSERT INTO Movies (Title, Minutes, Movie)
       VALUES('My Life', 126,
              DLVALUE('http://my.server.de/movies/mylife.avi'))

SELECT Title, DLURLCOMPLETE(Movie)
  FROM Movies
 WHERE Title LIKE '%Life%'

UPDATE Movies
  SET Movie =DLVALUE('http://my.newserver.de/mylife.avi')
  WHERE Title = 'My Life'

 SELECT Title, DLURLCOMPLETEWRITE(Movie) INTO :t, :url ...

 UPDATE Movies
  SET Movie = DLNEWCOPY(:url, 1)
  WHERE Title = :t
```

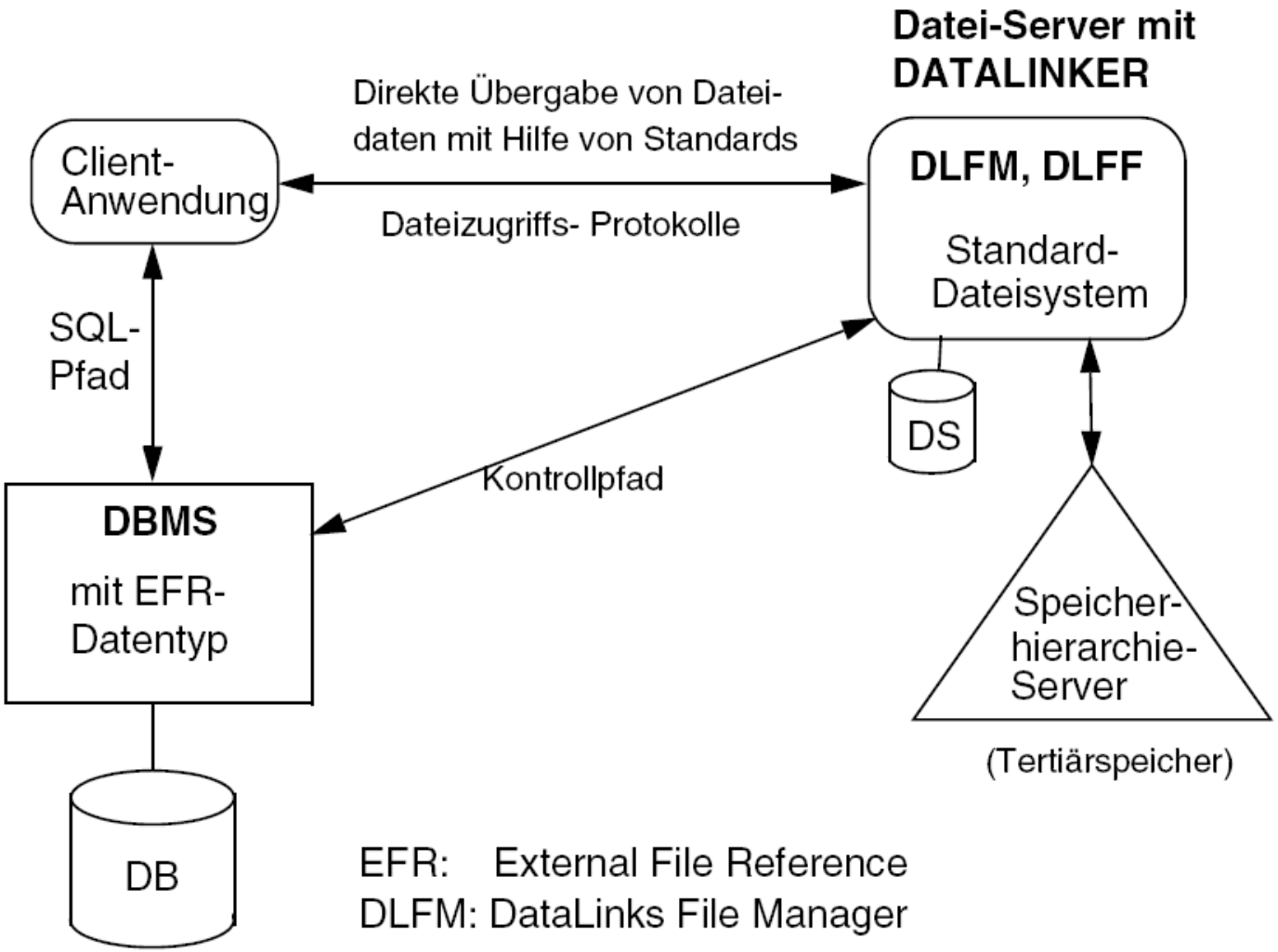

Datei-Server mit DATALINKER
Direkte Übergabe von Datei-daten mit Hilfe von Standards
Client-Anwendung
Dateizugriffs- Protokolle
DLFM, DLFF
Standard-Dateisystem
SQL-Pfad
DS
Kontrollpfad
DBMS
mit EFR-Datentyp
Speicher-hierarchie-Server
(Tertiärspeicher)
DB
EFR: External File Reference
DLFM: DataLinks File Manager
DLFF: DataLinks Filesystem Filter

Abbildung von Sätzen

- Speicherung variabel langer Felder
- dynamische Erweiterungsmöglichkeiten
- Berechnung von Feldadressen

Ziele bei der externspeicherbasierten Adressierung

- Kombination der Geschwindigkeit des direkten Zugriffs mit der Flexibilität einer Indirektion
- Satzverschiebungen in einer Seite ohne Auswirkungen

Alternativen

- TID-Konzept
- Zuordnungstabelle

Speicherung großer Datensätze (BLOBS)

Verwaltung externer Datenbestände