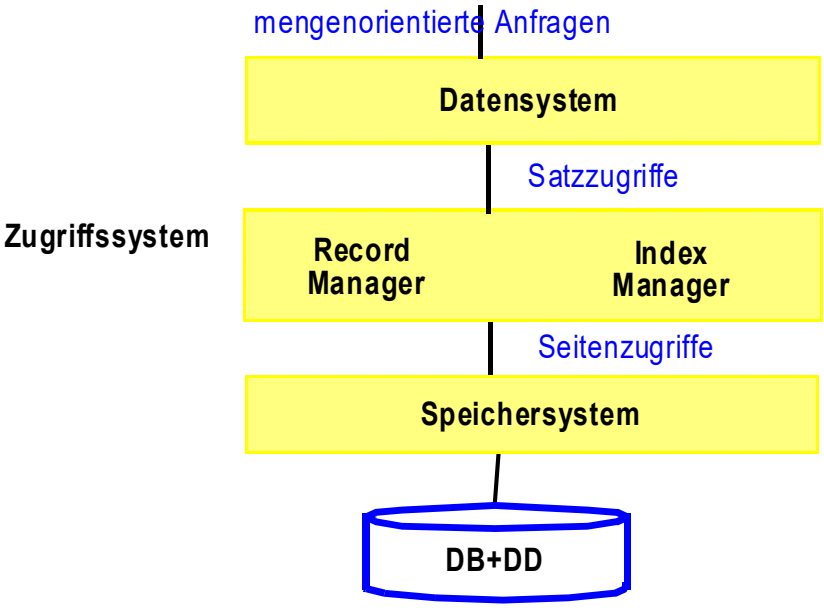
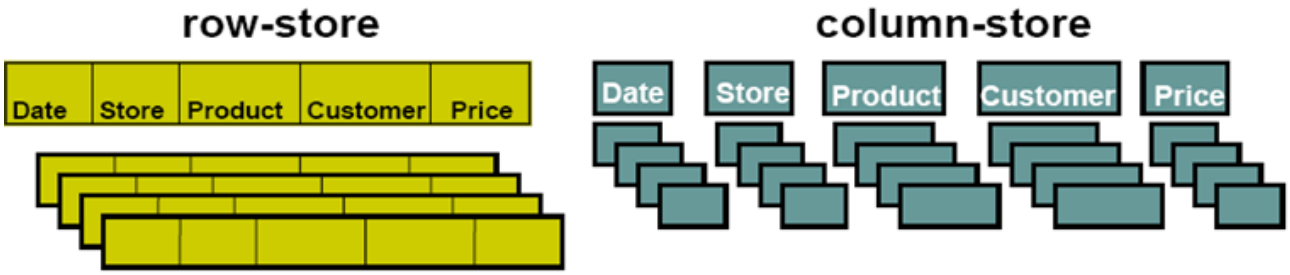
4. Satzverwaltung

- Einführung
- Zuordnung Sätze - Seite
  - Freispeicherverwaltung (im Segment, in der Seite)
  - Abbildung von Sätzen in Seiten
- Clusterung / Speicherung komplexer Objekte
- Satzadressierung
  - TID
  - Zuordnungstabelle / PPP
- Zuordnung Attribute - Sätze
  - Repräsentation von Attributwerten
  - Abbildung von Attributwerten
  - Speicherung von BLOBs/CLOBs
- Column Stores
  - Datenkompression



Row Store vs. Column Store



Standard-Modell in DBS: Sätze von Tabellen werden vollständig innerhalb je einer Seite gespeichert

\+ einfaches Hinzufügen neuer Sätze

\- Lesen nicht benötigter Attribute

Spaltenweise Zerlegung und Speicherung von Tabellen

\+ nur relevante Daten werden gelesen

\- mehrere Zugriffe zum Einfügen neuer Sätze

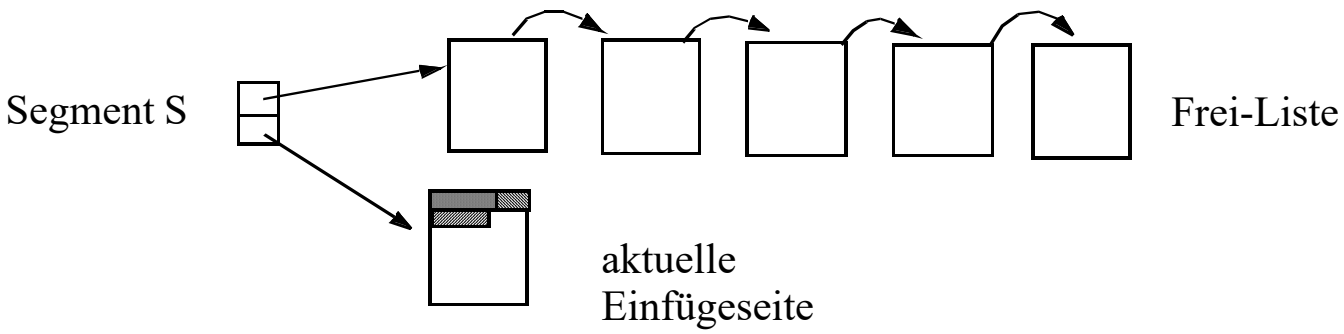
besonders geeignet zur Analyse-Unterstützung, z.B. für Data Warehouses

Satzoperationen

- Einfügen eines Satzes (INSERT)
  - Seite mit ausreichend freiem Platz bestimmen
  - einfach falls beliebige Seite möglich
  - ansonsten (z.B. bei festgelegter Sortierreihenfolge) ggf. Platz zu schaffen
- Bulk load (Laden zahlreicher Sätze)
  - initialer Füllgrad der Seite beachten (Parameter PCTFREE o.ä.)
- Satzänderung (Update) mit Änderung der Satzlänge
  - bei Wachstum sollte Satz möglichst in Ursprungsseite bleiben
- Löschen eines Satzes (Delete)
  - zunächst wird Speicherplatz nur als wiederverwendbar gekennzeichnet (free vs. reusable)
  - periodisches Kompaktieren (reusable -> free)
- Reorganisation
  - Zusammenlegen freier und wiederverwendbarer Bereiche

Freispeicherverwaltung

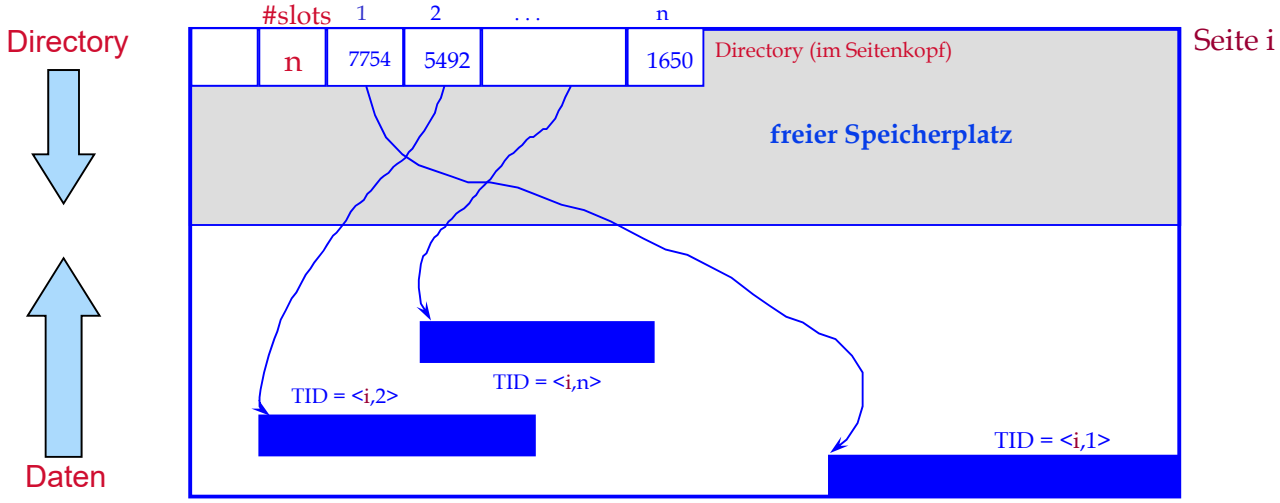
- pro Segment
  - Verweis auf aktuelle Seite für Einfügungen sowie
  - verkettete Liste leerer Seiten (Verweis pro leerer Seite erforderlich)



- falls aktuelle Seite voll ist, wird erste Seite der Frei-Liste die aktuelle Einfügeseite
- falls Seite durch Löschvorgänge leer wird, kommt sie an das Ende der Frei-Liste

Abbildung von Sätzen in Seiten

- Organisation
  - n Satztypen pro Segment
  - m Sätze verschiedenen Typs pro Seite
- oft vollständige Speicherung von Sätzen pro Seite
  - Voraussetzung: Satzlänge < Seitenlänge
  - variable Satzlänge: Verweise auf Satzbeginn in Directory am Anfang der Seite



Satz-ID (Tuple ID, TID) = <Seiten-ID, slot #>, zB <i,2>

Sortierte Speicherung von Sätzen

- Ziel: schneller Zugriff auf Sätze eines Satztyps in Sortierreihenfolge eines Attributes (z.B. Primärschlüssel)
- physisch benachbarte Speicherung in Sortierordnung: Clusterung
  - optimaler sortiert sequenzieller Zugriff: bei N Sätzen und mittlerem Blockungsfaktor B lediglich N/B physische Seitenzugriffe
  - pro Satztyp kann Clusterung nur bezüglich eines (Sortier-)Kriteriums erfolgen, falls keine Redundanz eingeführt werden soll
  - Änderungen können sehr teuer werden (Domino-Effekt)
    Verschiebekosten: N/(2*B) Seiten => Splitting-Technik

K1	K8	K17
K3	K9	K18
K4	K11	• • •
K6	K12	

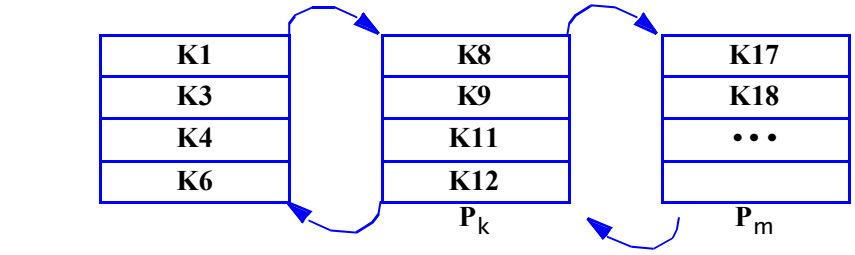
Änderung bei sortiert-sequenzieller Speicherung

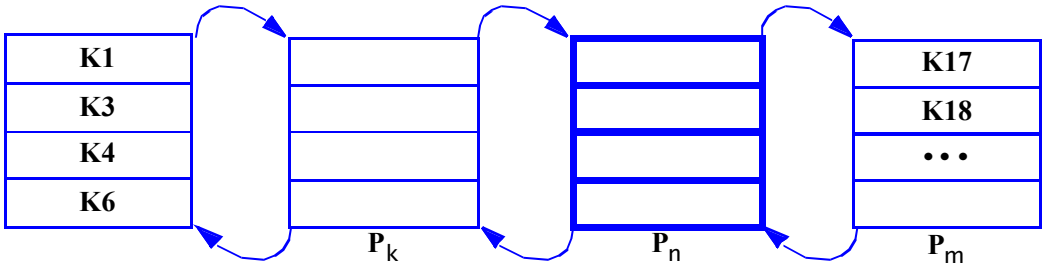
- Einfügen von K7?

B=4

K1	K8	K17
K3	K9	K18
K4	K11	• • •
K6	K12	

- Splitting-Technik:
  - Änderungen auf max. 3 Seiten beschränkt





Mehrere Satztypen pro Seite

- satztyp-übergreifende Clusterung von häufig zusammen benötigten Sätzen
- kann v.a. für schnelle Join-Bearbeitung vorteilhaft sein (hierarchische Clusterung entlang von 1:n-Beziehungen)

Kunde

KU-NR	KNAME	...

Konto

KTO-NR	KU-NR	KTOSTAND

```
Select  KNAME, KTO-NR, KTOSTAND
From    KUNDE JOIN  KONTO
```

gemeinsame Speicherung in Seite:

Kunde1
Konto11, Konto 12
Kunde 2
Konto21
Kunde 3 …

- nachteilig jedoch, wenn Anfragen auf 1 Satztyp dominieren

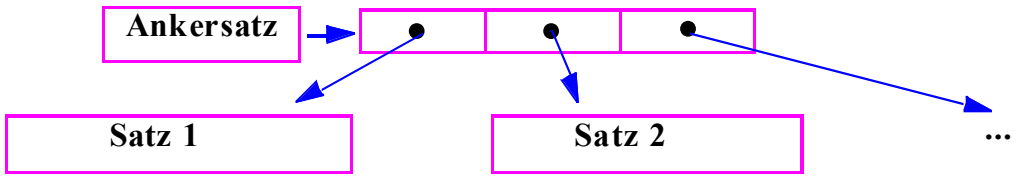
```
Select  *
From KUNDE
```

Speicherung komplexer Objekte

- Attribut eines (Anker-) Satzes können Kollektionen (Menge, Liste) von Sätzen enthalten
  - Beispiel: Abteilung - Mitarbeiter, Kunde - Konten, etc.
- generelle Speicheranordnung zwischen Ankersatz und zugehörigen Sätzen

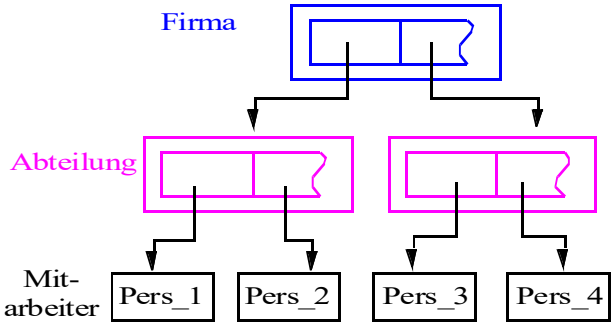
1. physische Nachbarschaft der Sätze: **Clusterung** (Listen, materialisierte Speicherung)

2. referenzierte Speicherung / **Verzeigerung** (Mini-Directory, Pointer-Array)

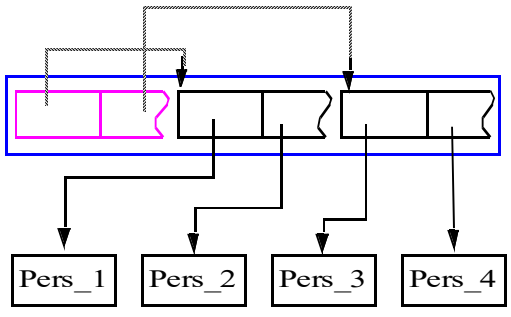


Speicherung komplexer Objekte (2)

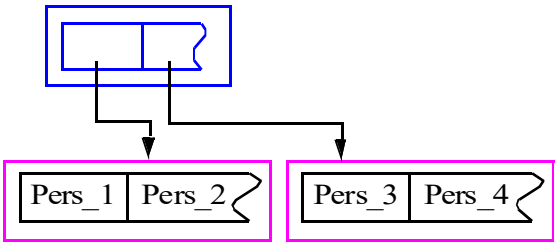
- beliebig tiefe Schachtelung komplexer Objekte: auf jeder Stufe kann zwischen den Speichermöglichkeiten gewählt werden
- 2-stufiges Beispiel: komplexes Objekt Firma mit Abteilungen und Mitarbeitern



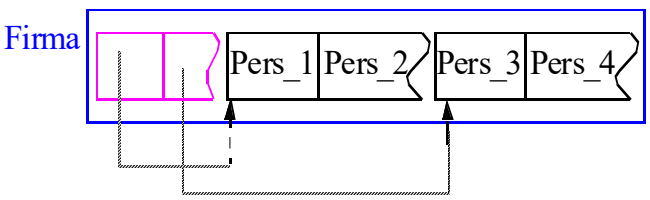
1) Verzeigerung / Verzeigerung



2) Clusterung / Verzeigerung



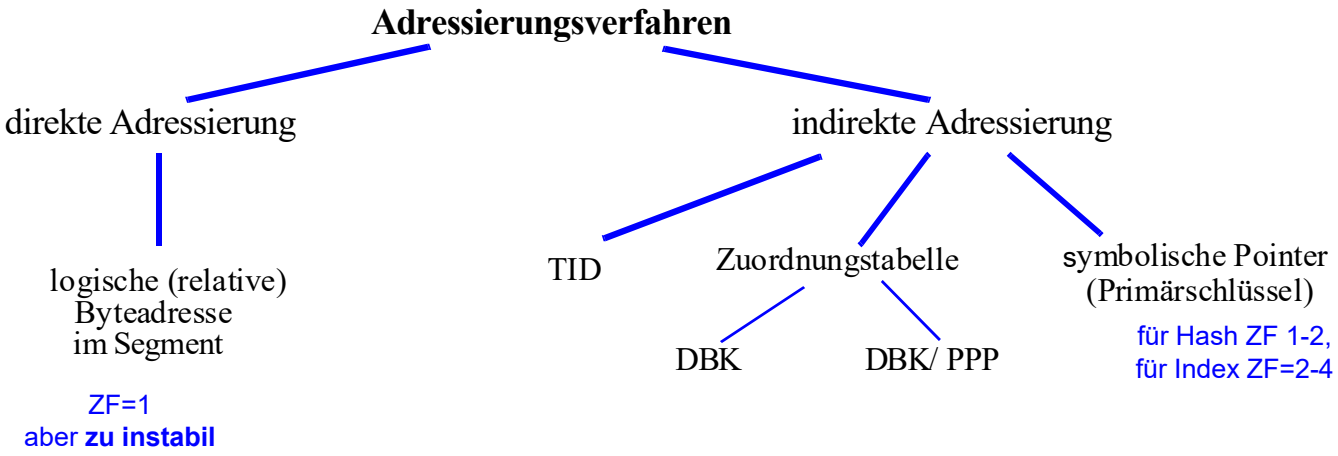
3) Verzeigerung / Clusterung



4) Clusterung / Clusterung

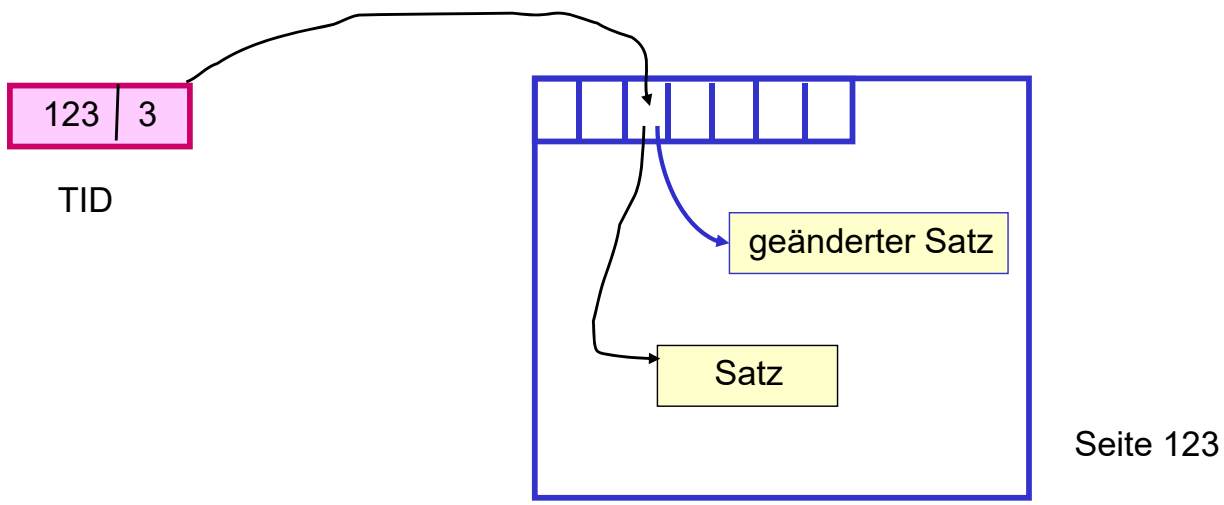
Externspeicherbasierte Satzadressierung

- DB-Adresse eines Satzes: Segment-ID + Adresse im Segment
- Ziele:
  - schneller, Satzzugriff, d.h. wenig Seitenzugriffe / geringer *Zugriffsfaktor (ZF)*
  - hinreichend stabil gegen geringfügige Verschiebungen (Verschiebungen innerhalb einer Seite ohne Auswirkungen)
  - seltene oder keine Reorganisationen (z.B. Umspeicherung, Adressänderung für Sätze)
- Adressierung in Segmenten: logisch zusammenhängender Adressraum



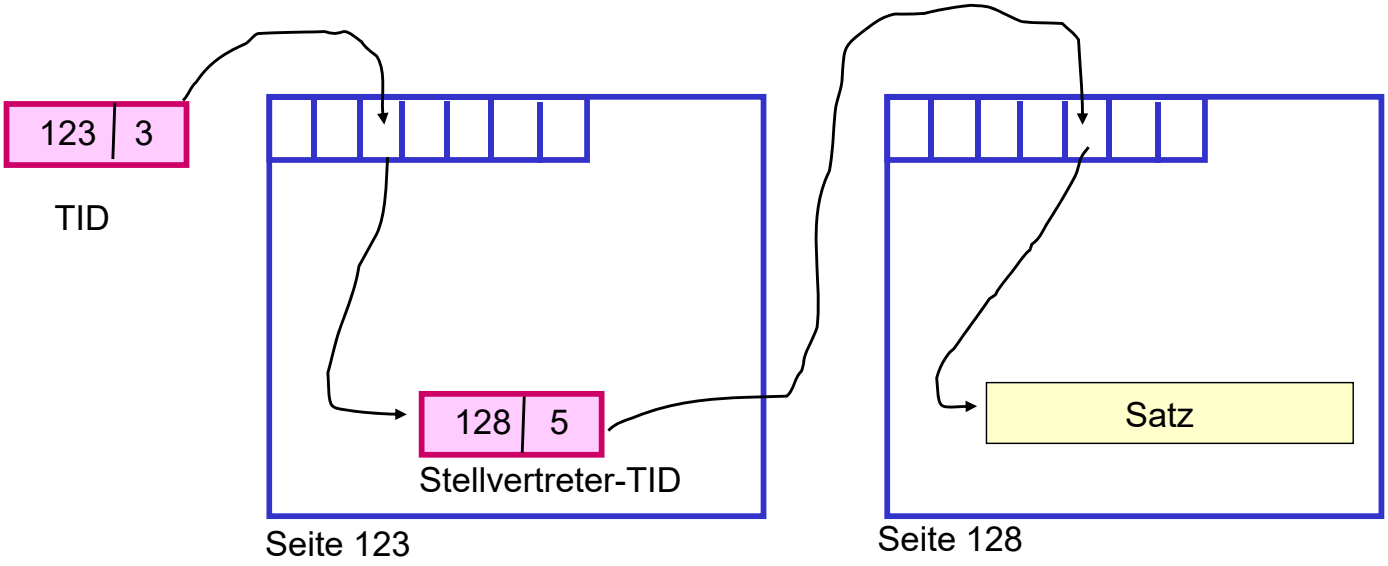
Satzadressierung: TID-Konzept

- TID (Tuple Identifier) dient zur Adressierung in einem Segment und besteht aus zwei Komponenten:
  - Seitennummer (3-6 B) + relative Indexposition innerhalb Seite (1-2 B)
    -> **TID-Gesamtlänge 4-8 B**
- Satzverschiebungen innerhalb einer Seite (z.B. wegen Verlängerung)
  - keine Auswirkungen auf TID und Zugriffskosten



TID-Adressierung (2)

- Migration eines Satzes in andere Seite
  - Vorwärtsverweis in Primärseite (Stellvertreter-TID)
  - eigentliche TID-Adresse bleibt stabil
- Überlaufkette: Länge <= 1 -> max. Zugriffskosten: 2 Seitenzugriffe
- 10% Überläufer-Anteil: mittlerer Zugriffsfaktor ca 1,1

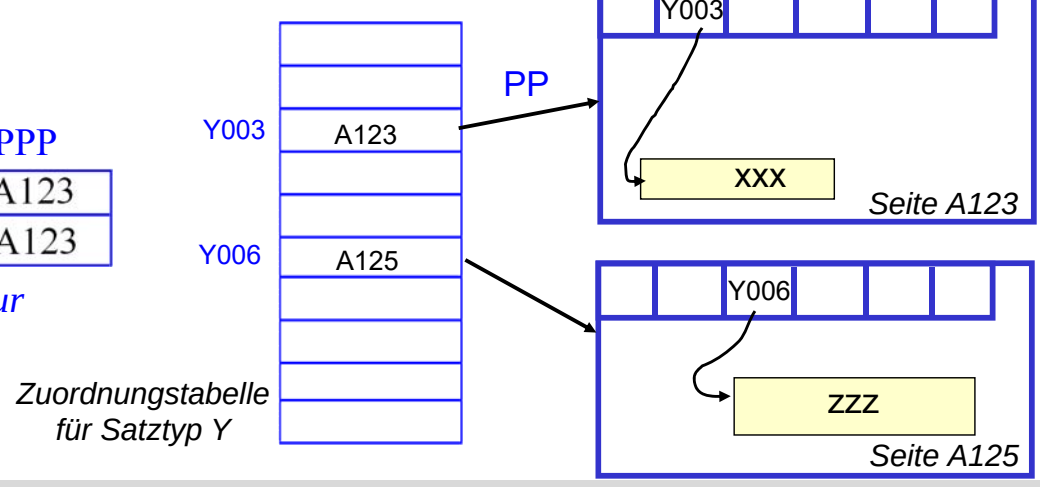


Satzadressierung über Zuordnungstabellen

- jeder Satz erhält eindeutigen Identifikator: OID bzw. DBK (database key)
  - Vergabe erfolgt i.a. durch DBMS
  - DBS-interne Verweise auf Sätze, z.B. in Indexen, erfolgen über OID / DBK
- Zuordnungstabelle enthält pro OID/DBK zugehörigen Page Pointer (PP)
  - Segment-ID (1-2 B) + Seitennummer (3-6 B)
  - Zugriffsfaktor 2 (Zugriffe auf Tabelle und Seite)
- Probable Page Pointers (PPP) in Indexen ersparen u.U. Zugriff auf Zuordnungstabelle
  - Zugriffsfaktor 1 wenn PPP aktuell, sonst 3

Such-kriterium	DBK	PPP
xxx	Y003	A123
zzz	Y006	A123

Indexstruktur



Repräsentation von Attributwerten

- Repräsentation von DBS-Datentypen
  - Int (short): 2 Bytes, z.B. 35 ist 0000 0000 0010 0011
  - Real, Floating Point: n Bits für Mantisse, m für Exponent
  - Character: 1 Byte pro Zeichen, z.B. ASCII-Codierung
  - Boolean: 1 Byte pro Wert (z.B. TRUE: 1111 1111, FALSE: 0000 0000);
    - weniger als 1 Byte pro Wert i.a. zu aufwendig
  - DATE: INTEGER (#Tage seit 1. Jan. 1900) bzw.
    YYYYMMDD (8 Zeichen) oder YYYYDDD (7 Zeichen)
  - TIME: INTEGER (Sekunden seit Mitternacht), Zeichen: HHMMSS

- Strings: feste vs. variable Länge
  - feste (maximale) Länge: CHAR (15), VARCHAR (255)
  - variable Länge: vorgestellte Längenangabe bzw. spezielles Endezeichen
  - ggf. Tabellenersetzung für Werte (L = Leipzig), Verschlüsselung ...

Abbildung von Attributwerten in Sätzen

- Satz: Aggregation zusammengehöriger Attributwerte (Felder)

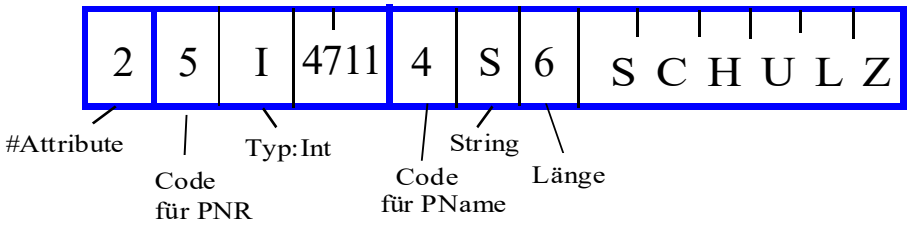
- Forderungen
  - günstiger Platzbedarf
  - Unterstützung dynamischer Attributlängen
  - effiziente Speicherung von Nullwerten
  - einfaches Hinzufügen neuer Attributdefinitionen
  - direkter Zugriff auf i-tes Attribut

- feste vs. variable Satzlänge

- festes vs. variables Satzformat
  - DBS meist festes Satzformat; Metadaten weitgehend im Katalog
  - variables Format z.B. für semistrukturierte/selbstbeschreibende Daten; eingebettete Metadaten

Variables Satzformat

- "selbstbeschreibende" Sätze: Mitführen von Attributnamen und Attributtypen
- Beispiel



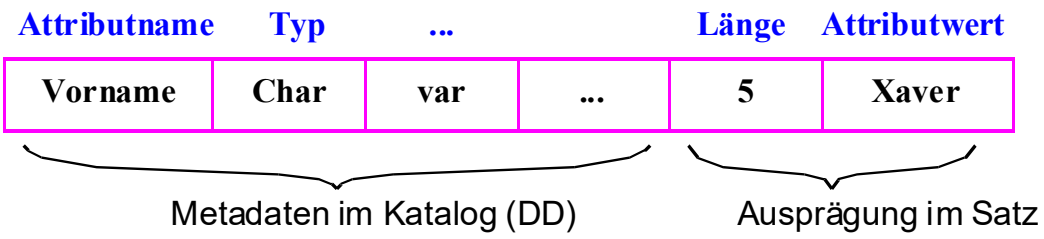
Code-Tabelle

Code	Wert
4	PNAME
5	PNR
…	…
I	INT
S	STRING
…	…

- Attributnamen / Tags können auch als Strings gespeichert werden
  - Nutzung von Kurznamen / Kodierungen
- keine Speicherung von Nullwerten
- i.a. hoher Platzbedarf / aufwändiger Zugriff
- große Flexibilität

Festes Satzformat

- Trennung von Metadaten (im Katalog) und gespeicherten Sätzen
  - pro Attribut:



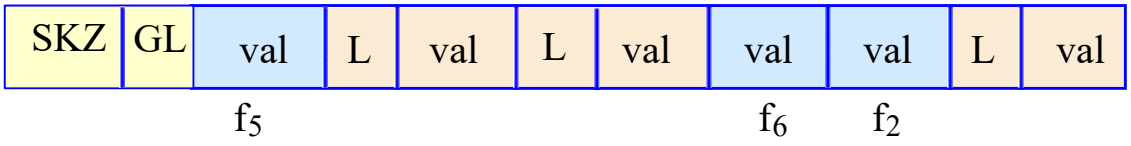
  - Satz- und Zugriffspfadbeschreibung im Katalog
  - Anzahl von Attributen, Reihenfolge, Datentypen, Bedeutung

- unterschiedliche Realisierungsmöglichkeiten u.a. für
  - Verwaltung variabel langer Attributwerte
  - Adressierung des i-ten Attributes
  - Verwaltung von Nullwerten

Abspeicherungsformen

- eingebettete Längenfelder

  Beispiel: PERS (PNR, Name, Beruf, Gehalt, ANR, Ort)

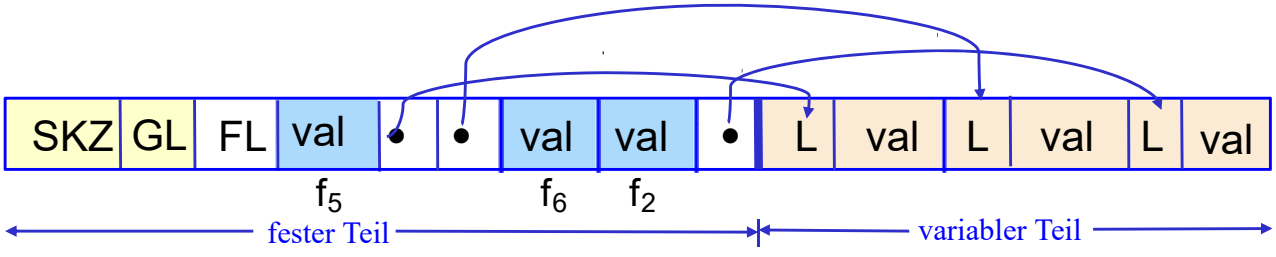


Katalogeintrag: f5 | v | v | f6 | f2 | v |
SKZ = Satzkennzeichen / OID
GL = Gesamtlänge (bzw. #Felder)

  - speicherökonomisch: viele Sätze pro Seite möglich
  - Nullwert: Länge 0
  - nicht repräsentierte Attribute (am Ende) haben per Definition Nullwert
  - einfaches Hinzufügen neuer Attribute
  - Bestimmung der Attributwertadresse erst zur Laufzeit

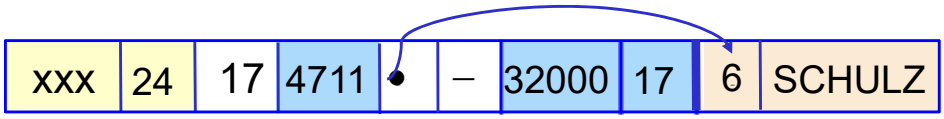
Abspeicherungsformen (2)

- Zerlegung von Sätzen in Teile mit fester und variabler Länge
  - fester Teil: Attributwerte fester Länge + Zeiger auf variabel lange Attributwerte im 2. Teil
  - variabler Teil: variabel lange Attributwerte mit eingebetteten Längenfeldern



Katalogeintrag: f5 | v | v | f6 | f2 | v |
SKZ = Satzkennzeichen / OID
GL = Gesamtlänge
FL = Länge des festen Teils

  - Adresse für jedes Attribut an fester Position
  - effiziente Speicherung von Nullwerten
  - einfaches Hinzufügen neuer Attributdefinitionen



Darstellung und Handhabung langer Felder

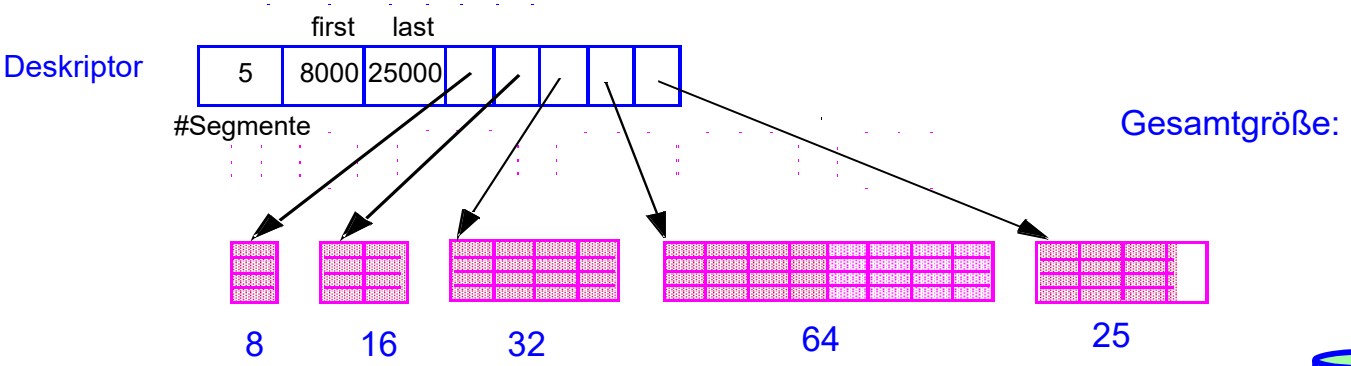
- lange Attribute, z.B. für Typen TEXT, IMAGE, VIDEO erfordern Sonderbehandlung
- Speicherung als BLOBs oder CLOBs unter Kontrolle des DBS
- Anforderungen
  - idealerweise keine Größenbeschränkungen
  - gezieltes Lesen und Schreiben von Teilbereichen
  - Verkürzen, Verlängern und Kopieren
  - Suche nach vorgegebenem Muster, Längenbestimmung. . .

```
CREATE TABLE  Pers
    (PNR   INTEGER,
     Name  VARCHAR (80),
     Lebenslauf  CLOB,
     Bild        BLOB (12M) …)
```

- Darstellung großer Speicherobjekte
  - besteht potenziell aus vielen Seiten
  - ist eine uninterpretierte Bytefolge
  - OID-Verweis (Adresse) im Satz zeigt auf Objektkopf (Header) des großen Objekts
  - unterschiedliche Speicherungsstrukturen möglich: Kette von Einträgen fester Länge, sequenzielle Liste (Datei), B*-Baum etc.

Clusterung für lange Felder

- Implementierung im Starburst-Prototyp (IBM Research)
  - Grundlage für DB2-Realisierung
  - effiziente Speicherallokation und -freigabe für Feldgrößen von bis zu 2 GB (Sprache, Bild, Musik oder Video)
- hohe E/A-Leistung durch Clusterung
  - Schreib- und Lese-Operationen sollen E/A-Raten nahe der Übertragungsgeschwindigkeit der Magnetplatte erreichen
- prinzipielle Repräsentation
  - 1 oder mehrere „Segmente“ (Cluster) zur Darstellung des langen Feldes
  - Deskriptor mit Liste der Segmentbeschreibungen

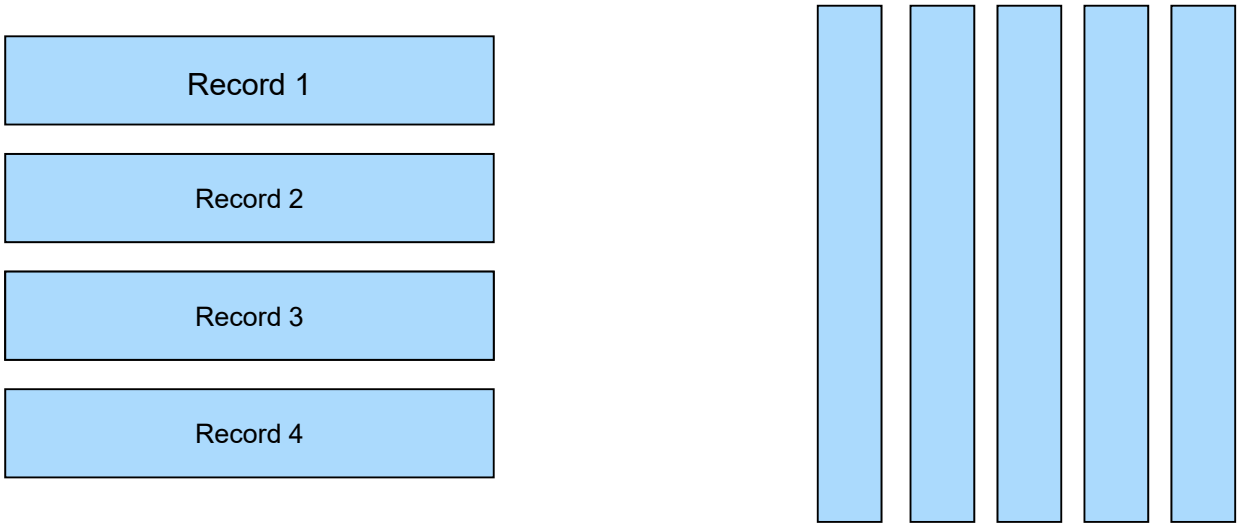


Clusterung für lange Felder (2)

- Datenallokation bei unbekannter Objektgröße
  - Wachstumsmuster der Segmentgrößen wie im Beispiel:
    1, 2, 4, ..., 2^n Seiten werden jeweils zu einem Segment zusammengefasst
  - MaxSeg = 2048 Seiten für n = 11
  - falls MaxSeg erreicht, werden weitere Segmente der Größe MaxSeg angelegt
  - das letzte Segment wird auf die verbleibende Objektgröße gekürzt
- Datenallokation bei vorab bekannter Objektgröße
  - Objektgröße G (in Seiten)
  - G ≤ MaxSeg: es wird 1 Segment angelegt
  - G > MaxSeg: es wird Folge maximaler Segmente angelegt; letztes Segment wird auf verbleibende Objektgröße gekürzt
- Verarbeitungseigenschaften
  - effiziente Unterstützung von sequenziellen und wahlfreien Lesevorgängen
  - einfaches Anhängen und Entfernen von Bytefolgen am Ende des Objektes
  - schwieriges Einfügen und Löschen von Bytefolgen in der Mitte des Objektes

Column Stores

- spaltenweise statt zeilenweise Speicherung von Tabellen
- frühe Realisierungen im Rahmen vertikaler Partitionierung (Sybase IQ)
- viele neue Realisierungen seit ca. 2005, v.a. zur Unterstützung von Analyse-Anfragen / OLAP
  - SAP Hana, Vertica …
- hybride Lösungen mit Row und Column Store (z.B. MS SQLServer, DB2)



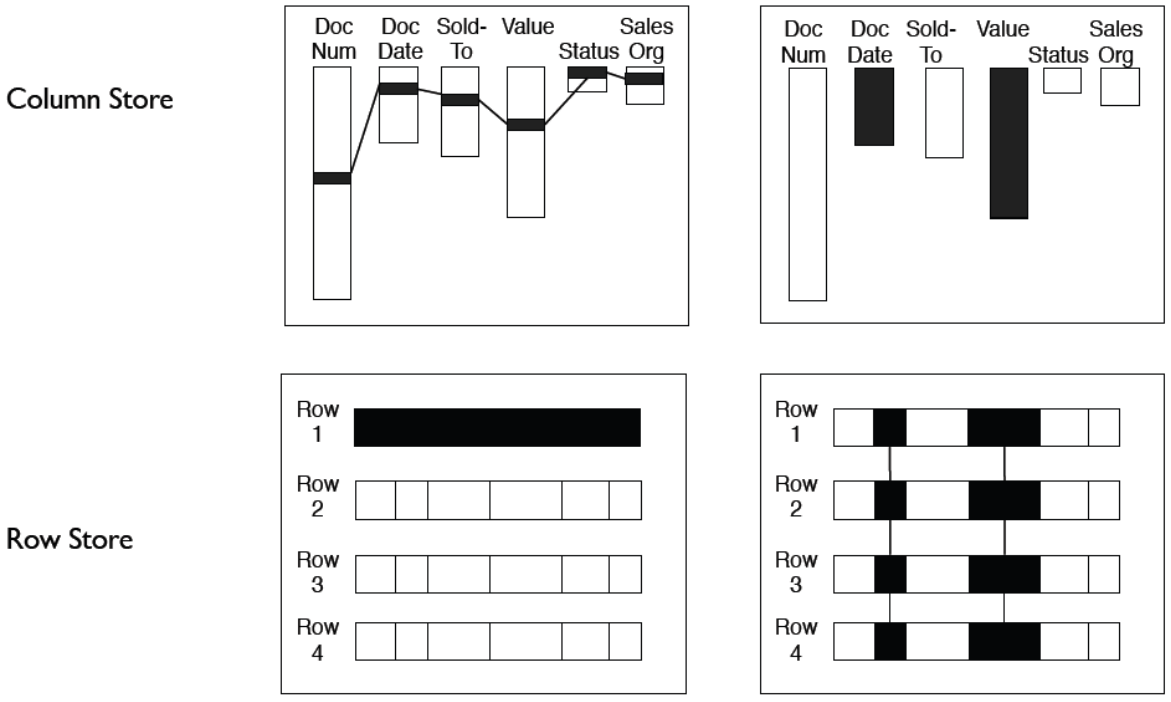
Vorteile / Nachteile

- Vorteile Column Store
  - I/O-Einsparungen, falls nur wenige Attribute auszuwerten sind
    (z.B. Berechnung von Durchschnittsgehalt, Umsatzsumme ...)
  - zusätzliche Einsparungen falls Attributwerte komprimiert werde
  - effiziente Aggregationsmöglichkeiten
  - OLAP-orientiert
  - oft deutlich bessere Trefferraten im CPU-Cache durch Fokussierung auf relevante Daten
- Nachteile Column Store (-> Vorteile Row Store)
  - ungünstig für Operationen, die (fast) alle Attribute von Tupeln betreffen, z.B. für Änderungen / effizientes Einfügen neuer Tupel
  - weniger günstig für OLTP

Anfragen – Column vs. Row Store *

SELECT *
FROM Sales Orders
WHERE Document Number = '95779216'

SELECT SUM(Order Value)
FROM Sales Orders
WHERE Document Date > 2009-01-20



* Quelle: Hasso Plattner: Enterprise Applications – OLTP and OLAP – Share One Database Architecture

Datenkompression

- Column Stores nutzen oft komprimierte Speicherung von Attributwerten
  - reduzierter Speicherbedarf
  - reduzierter I/O-Aufwand
- leichtgewichtige Kompression mit geringem Aufwand zur Dekomprimierung
  - wünschenswert: Auswertungen auf komprimierten Werten selbst
- zahlreiche Varianten
  - Run Length Encoding (RLE)
  - Bit Vector Encoding
  - Delta Coding
  - Wörterbuch-Kodierung
  - etc.
- pro Spalte kann das beste Kodierungsverfahren gewählt werden

Run Length Encoding (RLE) / Lauflängenkodierung

- Zusammenfassung von Nachbarn mit gleichem Wert
  (Wert, Startposition, #Vorkommen)
- effizient bei langen Folgen gleicher Werte
  - wird durch Sortierung der Attributwerte pro Spalte unterstützt
- einfache Dekodierung
- Auswertungen auch auf komprimierten Werten möglich

Ort
Berlin
Berlin
Berlin
Dortmund
Leipzig
Leipzig
Leipzig
München
München

Ort	ab	Anz
Berlin	1	3
Dortmund	4	1
Leipzig	5	3
München	8	2

Bitvektor-Kodierung

- Spalte wird für k mögliche Attributwerte durch k Bitvektoren repräsentiert
  - Bit 1 (0) für Bitliste j an Position i bedeutet, dass Satz i den Wert j (nicht) hat
- speichergünstig bei wenigen Attributwerten
  - kann mit RLE kombiniert werden (lange 1- bzw. 0-Folgen)
- effiziente Auswertungen von Selektionen

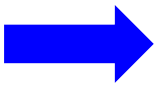
Matnr	…	Stud.gang
2345678		Bachelor Informatik
3456789		Master Informatik
1234567		Bachelor Informatik
4321876		Master Data Science
5678912		Master Informatik
6785432		Bachelor Informatik
2198765		Master Data Science
4321987		Bachelor Informatik

Bachelor Informatik	1010.0101
Master Informatik	0100.1010
Master Data Science	0001.0010

Delta-Kodierung

- Speicherung der Differenz zu Vorgängerwerten
- v.a. für numerische Werte
- Einsparungen v.a bei Sortierung

Matnr	…
2345678	
2345679	
2345683	
2345698	
2756789	
2756809	
2756814	
…	

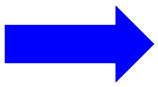


Matnr	…
2345678	
1	
4	
15	
411091	
10	
4	
…	

Wörterbuch-Kodierung (Dictionary Encoding)

- Wörterbuch mit Code für alle zu ersetzende (String-)Werte
  - günstig v.a. bei wenigen und langen Attributwerten
- erfordert keine Sortierung
- Auswertung auf komprimierten Daten möglich
- Lookup zur Dekomprimierung

Knr	Bundesland
…	Sachsen
…	Sachsen
…	Bayern
…	Mecklenburg-Vorpommern
…	Sachsen-Anhalt
…	Mecklenburg-Vorpommern
…	Nordrhein-Westfalen
…	Sachsen



Knr	Bundesland
…	4
…	4
…	1
…	2
…	5
…	2
…	3
…	4

Dictionary

Bayern	1
Mecklenburg-Vorpommern	2
Nordrhein-Westfalen	3
Sachsen	4
Sachsen-Anhalt	5

Zusammenfassung

- Freispeicherinformation auf verschiedenen Ebenen
  - Segment (Datei), Seite
- Abbildung von Sätzen:
  - meist festes Format, variable Länge
  - Clusterung, komplexe Objekte
- Ziele bei der Satzadressierung
  - Kombination der Geschwindigkeit des direkten Zugriffs mit der Flexibilität einer Indirektion
  - Satzverschiebungen in Seite ohne Auswirkungen ⇒ TID-Konzept oder Zuordnungstabelle
- Speicherung variabel langer Felder
  - dynamische Erweiterungsmöglichkeiten
  - Berechnung von Feldadressen
- Sonderbehandlung zur Speicherung großer Objekte (BLOBs)
  - große sequenzielle Listen (Clusterung): hohe E/A-Leistung
- Column Store-Techniken hilfreich für beschleunigte Anfragen
  - Kompressionsverfahren: RLE, Bit Vector- / Delta- / Wörterbuch-Kodierung