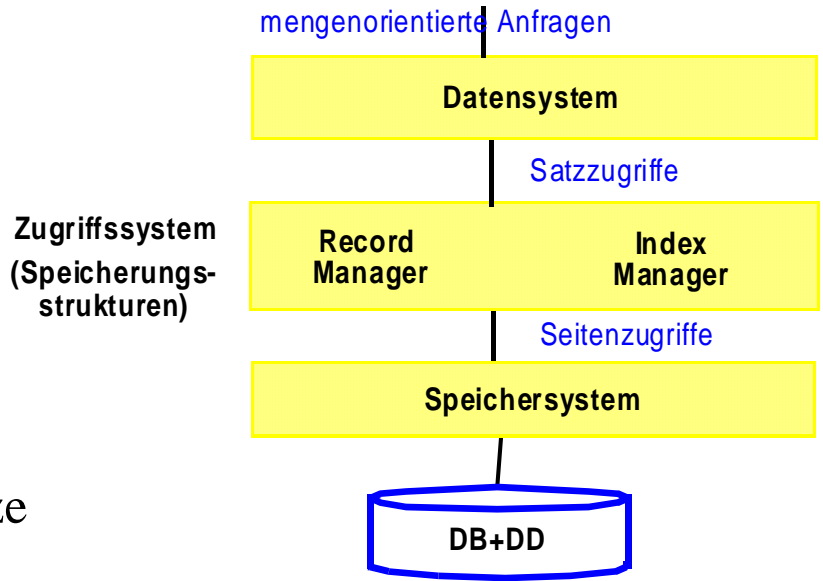
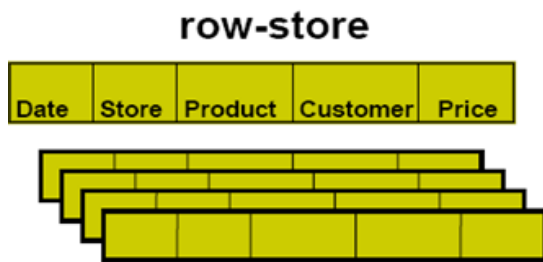


# 4. Satzverwaltung

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



## Row Store vs. Column Store



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

Spaltenweise Zerlegung und Speicherung von Tabellen

- + Einfaches Hinzufügen neuer Sätze
- Lesen nicht benötigter Attribute

- + 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

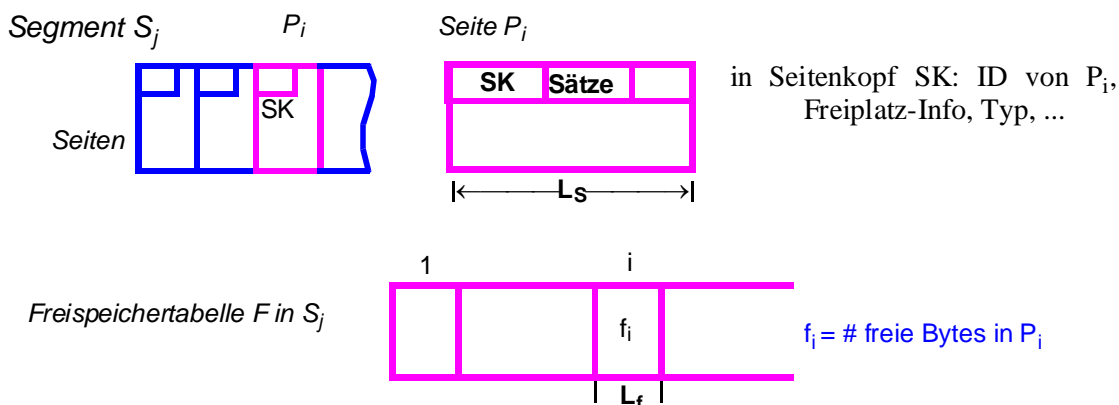
# Freispeicherverwaltung

## ■ Freispeicherverwaltung (FPA) für

- Segmente (Auswahl von Seiten zur Speicherung von 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 (2)

### ■ Größe der Freispeichertabelle F

Einträge pro Seite der Länge  $L_s$   $k = \left\lfloor \frac{L_s - L_{SK}}{L_f} \right\rfloor$

mit  $s = \# \text{Seiten im Segment}$   $\rightarrow n = \left\lceil \frac{s}{k} \right\rceil$  Seiten für F

### ■ Lage von F

- Segmentanfang bzw. -ende
- äquidistante Verteilung

### ■ Art der FPA

- **exakt**:  $L_f = 2$  Bytes
- **unscharf**:  $L_f = 1$  Byte (oder weniger)  
 $\rightarrow f_i$  in Vielfachen von  $\lceil L_s / 256 \rceil$

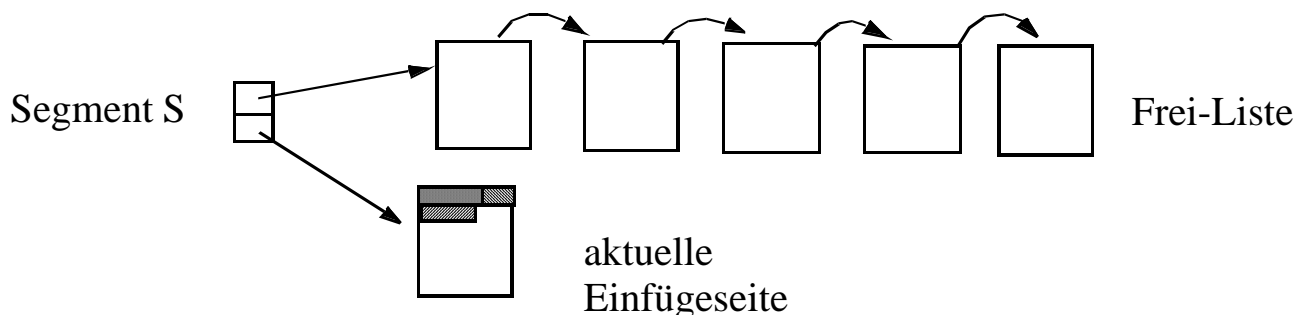
### ■ FPA innerhalb von $P_i$

- exaktes  $f_i$  in SK
- zusammenhängende Verwaltung (Verschiebungen!)  
oder Freispeicherkette (best-fit/ first-fit)

## Freispeicherverwaltung (3)

### ■ Alternative: 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

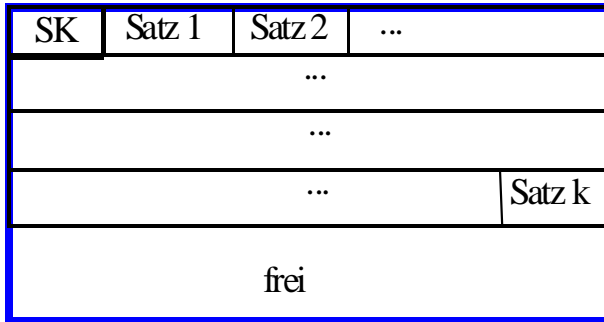
## ■ Organisation

- n Satztypen pro Segment
- m Sätze verschiedenen Typs pro Seite

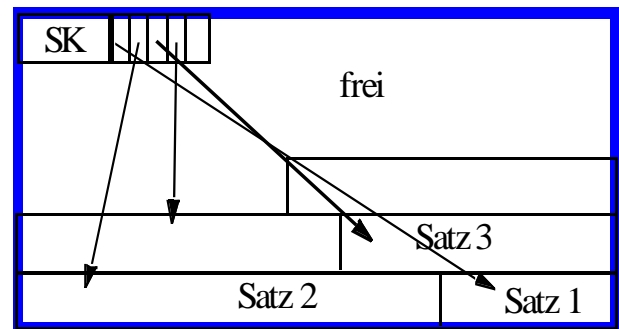
## ■ oft vollständige Speicherung von Sätzen pro Seite

- Voraussetzung: Satzlänge < Seitenlänge  $S_L \leq L_S - L_{SK}$
- Bei variabler Satzlänge: Verweise auf Satzbeginn (am Anfang oder Ende der Seite)

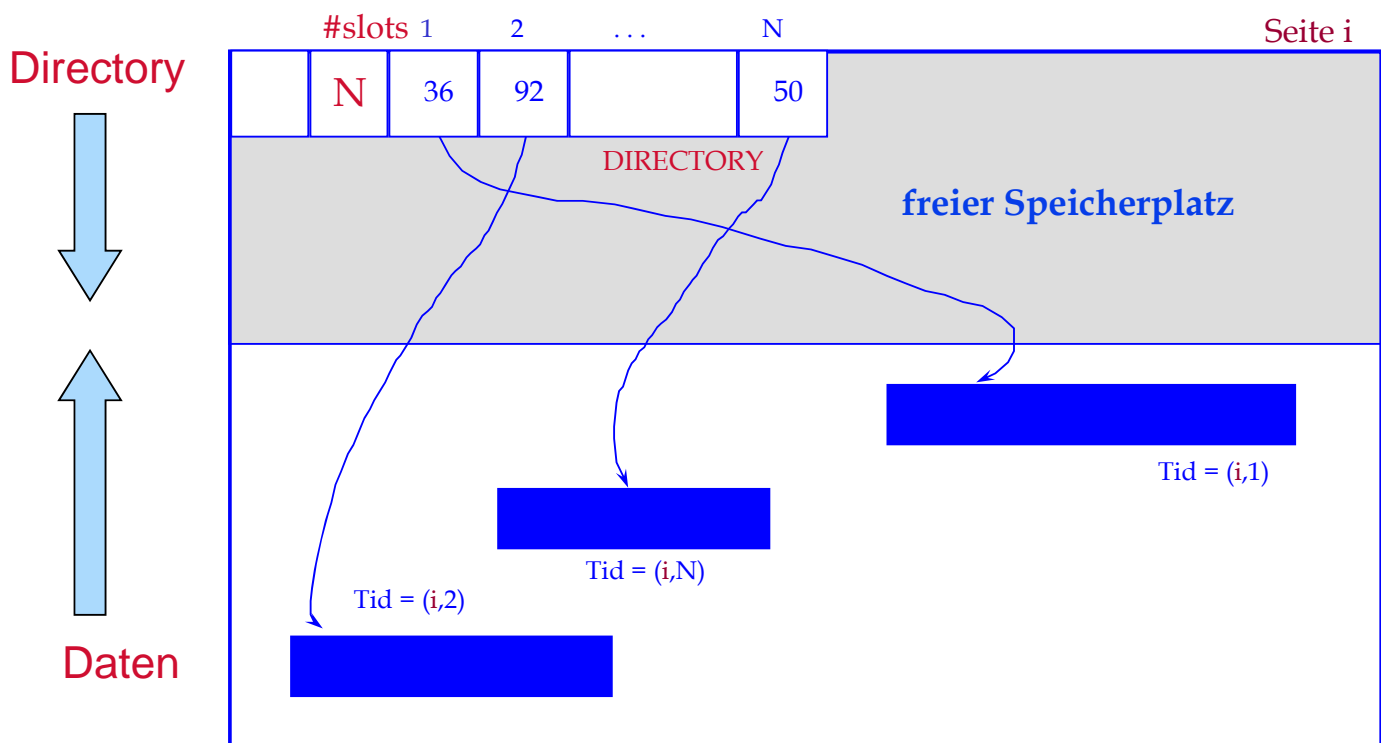
a) feste Satzlänge



b) variable Satzlänge



## Seitenaufbau (2)

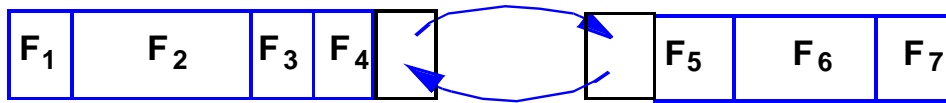


Satz-ID (Tuple ID) = <Seiten-ID, slot #>

## Abbildung von Sätzen in Seiten (3)

- Aufspalten von Sätzen auf mehrere Seiten ("spanned records", **Spannsatz**)

Bsp.: attributweises Aufspalten



- mögliche Gründe

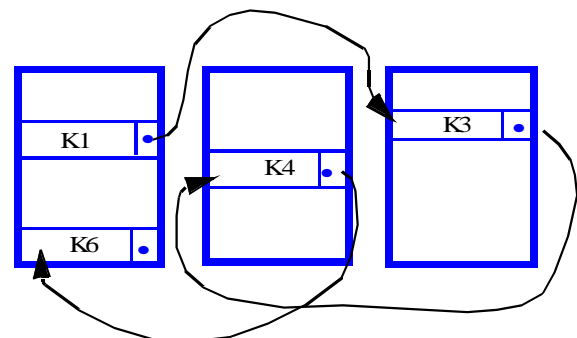
- Satzgröße  $S_L >$  Seitengröße  $L_S - L_{SK}$
- schlechte Platzausnutzung bei fester Satzlänge (Bsp.:  $L_S=4096$  B,  $S_L=2050$  B)
- Auslagern selten benötigter Attribute
- Auslagern variabler Satzanteile

- Spezialfall: separate Speicherung für große Attribute ("long filelds") wie BLOBs (z.B. für Video-Clips) oder Texte (CLOBs)

## 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 sortierter sequentieller 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
- Alternative: Verkettung von Sätzen (geringe Änderungskosten, langsamer Lesezugriff)

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



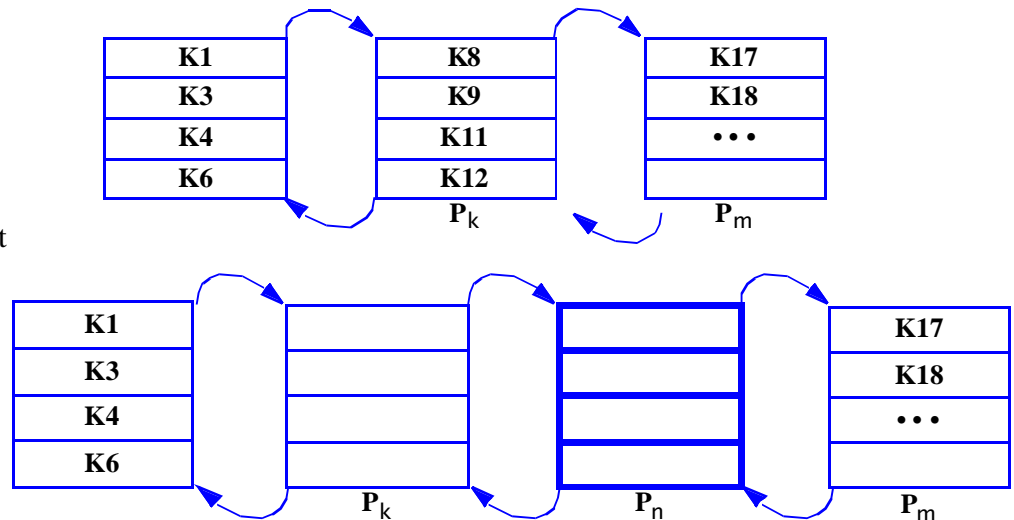
# Änderung bei sortiert-sequentieller Speicherung

## ■ Einfügen von K7?

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**

<u>KU-NR</u>	KNAME	...
--------------	-------	-----

**Konto**

<u>KTO-NR</u>	KU-NR	KTOSTAND
---------------	-------	----------

```
Select KNAME, KTO-NR, KTOSTAND
From KUNDE, KONTO
Where KONTO.KU-NR=KUNDE.KU-NR
```

- nachteilig jedoch, wenn Anfragen auf 1 Satztyp dominieren

```
Select *
From KUNDE
```

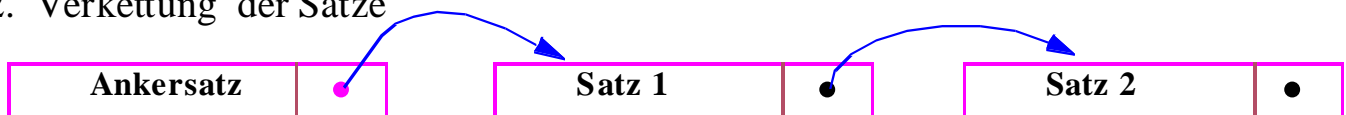
# Speicherung komplexer Objekte

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

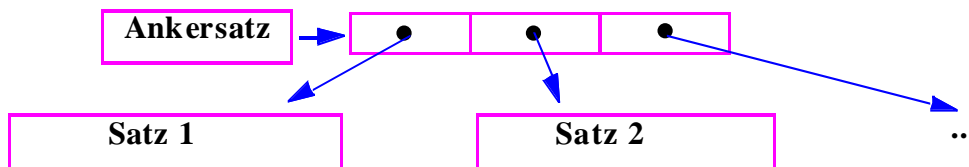
1. Physische Nachbarschaft der Sätze: Clusterung (Listen, materialisierte Speicherung)



2. Verkettung der Sätze

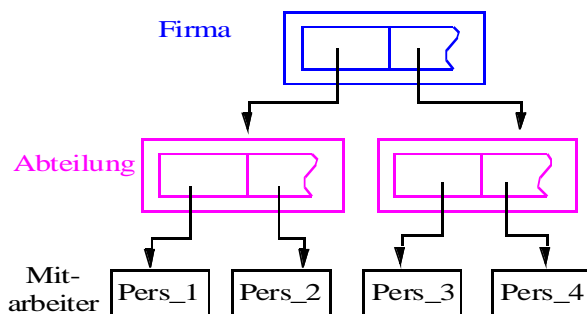


3. Referenzierte Speicherung / Verzeigerung (Mini-Directory, Pointer-Array)

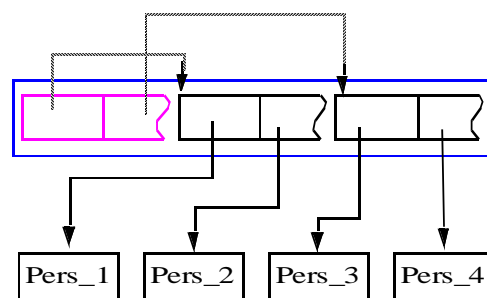


# Speicherung komplexer Objekte (2)

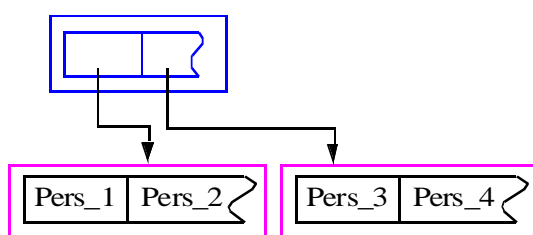
- beliebig tiefe Schachtelung komplexer Objekte: auf jeder Stufe kann zwischen den Speichermöglichkeiten gewählt werden
- 2-stufiges Beispiel: komplexes Objekt Firma mit Abteilungen und Mitarbeitern



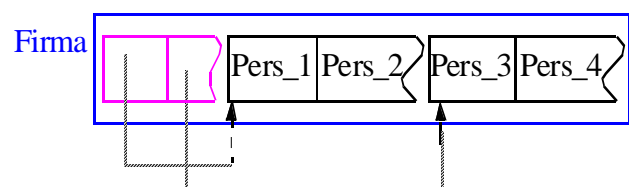
1) Verzeigerung / Verzeigerung



2) Clusterung / Verzeigerung



3) Verzeigerung / Clusterung

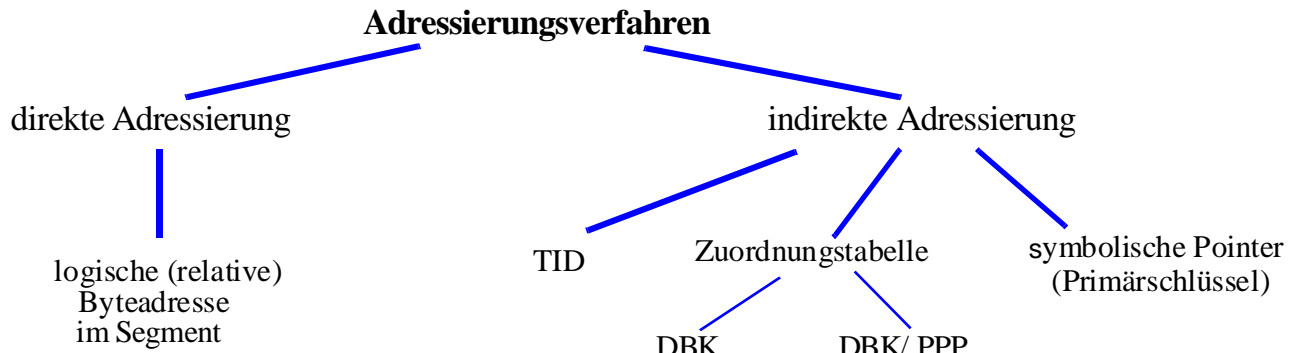


4) Clusterung / Clusterung



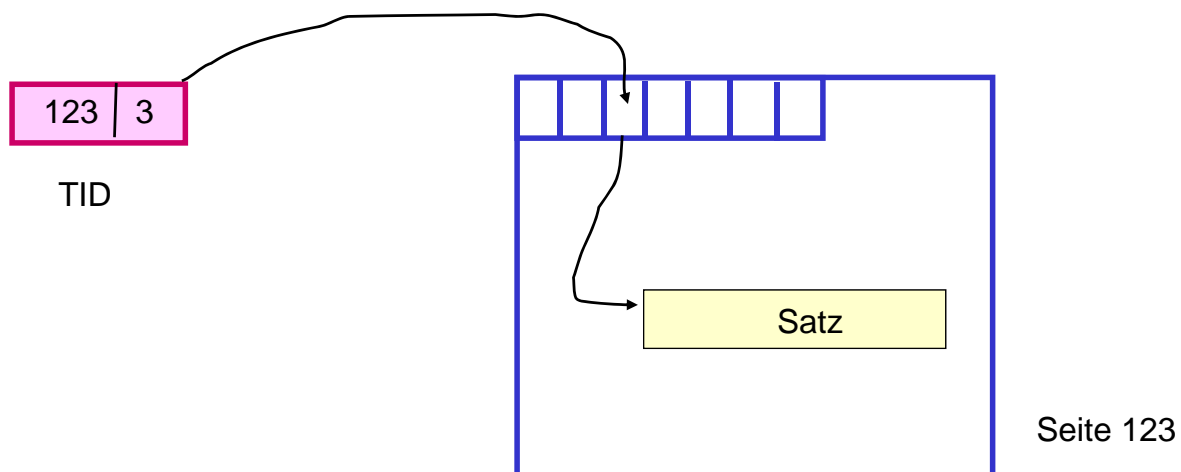
# Externspeicherbasierte Satzadressierung

- DB-Adresse eines Satzes (OID): Segment-ID (bzw. Satztyp-ID) + Adresse im Segment
- Ziele:
  - schneller, möglichst direkter Satzzugriff
  - hinreichend stabil gegen geringfügige Verschiebungen (Verschiebungen innerhalb einer Seite ohne Auswirkungen)
  - seltene oder keine Reorganisationen
- Adressierung in Segmenten: logisch zusammenhängender Adressraum



## Satzadressierung: TID-Konzept

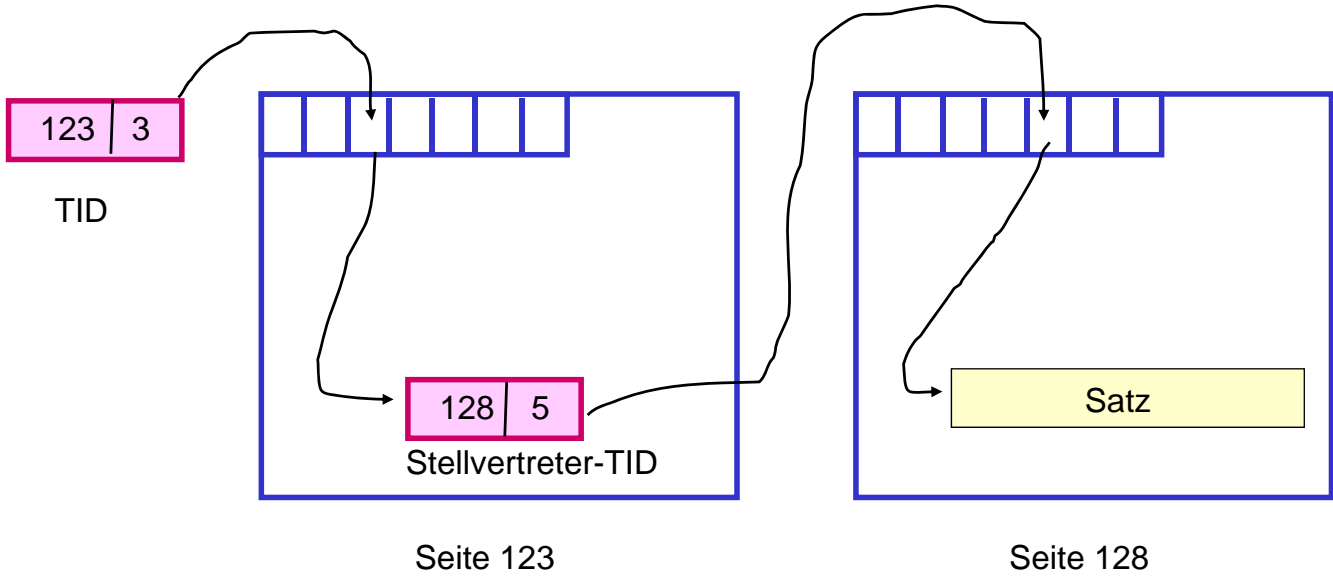
- TID (Tuple Identifier) dient zur Adressierung in einem Segment und besteht aus zwei Komponenten:
  - Seitennummer (3-6 B)
  - relative Indexposition innerhalb der Seite (1-2 B)
- Satzverschiebungen innerhalb einer Seite bleiben ohne Auswirkungen auf TID und Zugriffskosten





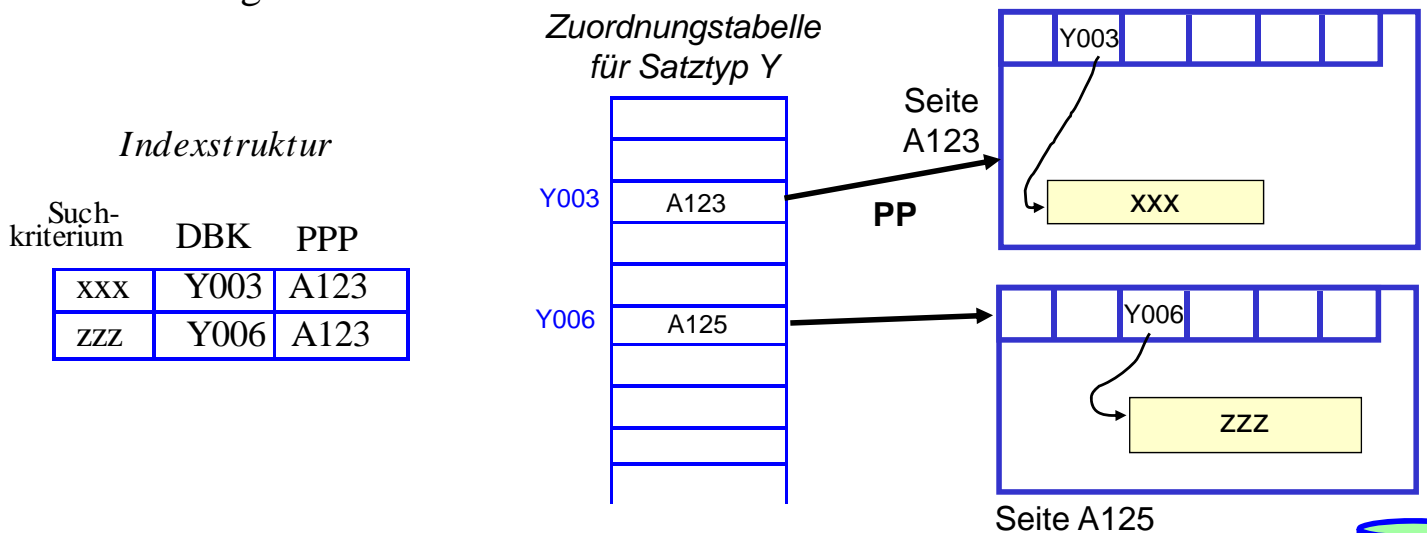
## TID-Adressierung (2)

- Migration eines Satzes in andere Seite
  - Vorwärtsverweis in Primärseite (Stellvertreter-TID)
  - eigentliche TID-Adresse bleibt stabil
- Überlaufkette: Länge  $\leq 1 \rightarrow$  max. Zugriffskosten: 2 Seitenzugriffe



## Satzadressierung über Zuordnungstabellen

- jeder Satz erhält eindeutigen Identifikator: OID bzw. Datenbankschlüssel (DBK)
  - Vergabe erfolgt i.a. durch DBVS
  - systeminterne Verweise auf Sätze erfolgen über DBK / OID
- Zuordnungstabelle enthält pro OID zugehörigen **Page Pointer (PP)**
  - Segment-ID (1-2 B) + Seitennummer (3-6 B)
- 'Probable Page Pointers' (PPP) in Zugriffspfaden ersparen u.U. Zugriff auf Zuordnungstabelle



# 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

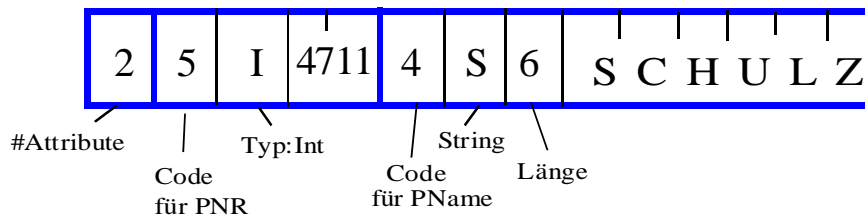
### ■ festes vs. variables Satzformat

- 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

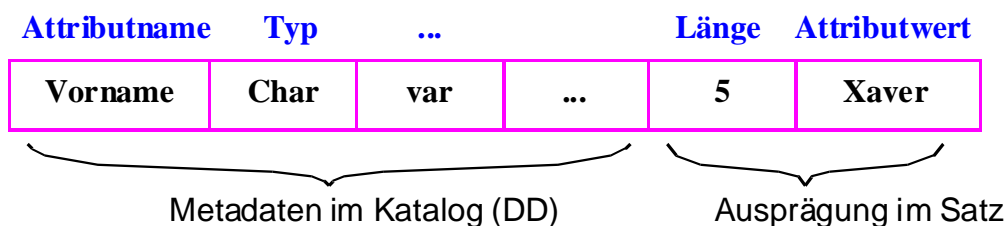


- Attributnamen / Tags können auch als Strings gespeichert werden
- keine Speicherung von Nullwerten
- i.a. hoher Platzbedarf / aufwändiger Zugriff
- große Flexibilität

# Festes Satzformat

- Trennung von Metadaten (im Katalog) und gespeicherten Sätzen

– pro Attribut:



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

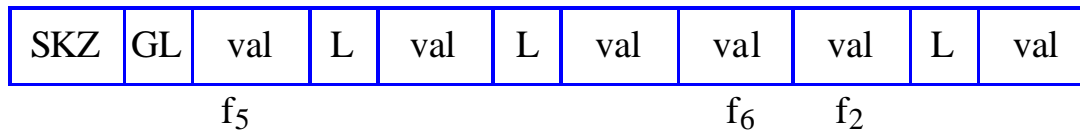
- Unterschiedliche Realisierungsmöglichkeiten u.a. für

- Verwaltung variabel langer Attributwerte
- Adressierung des i-ten Attributes
- Verwaltung von Nullwerten

# Abspeicherungsformen

## ■ Eingebettete Längfelder

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



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

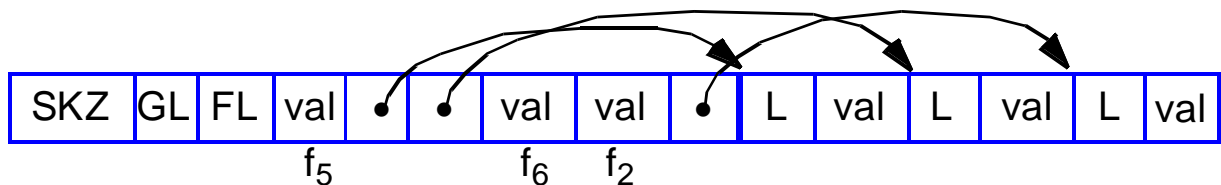
- speicherökonomisch: viele Sätze pro Seite möglich
- Nullwert: Länge 0
- nicht repräsentierte Attribute haben per Definition Nullwert
- einfaches Hinzufügen neuer Attribute
- Bestimmung der Attributwertadresse erst zur Laufzeit



# 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 Längfeldern



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

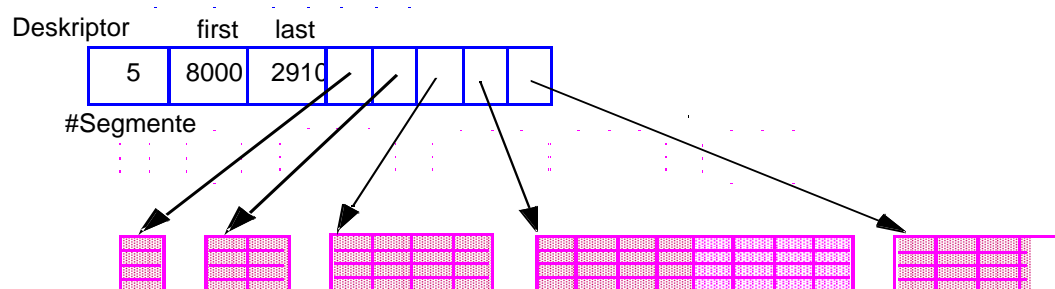


# Darstellung und Handhabung langer Felder

- lange Attribute, z.B. für Typen TEXT, IMAGE, VIDEO erfordern Sonderbehandlung
- Speicherung als BLOBs oder CLOBs unter Kontrolle des DBS
- Anforderungen
  - Idealerweise keine Größenbeschränkungen
  - allgemeine Verwaltungsfunktionen
  - gezieltes Lesen und Schreiben von Teilbereichen
  - Verkürzen, Verlängern und Kopieren
  - Suche nach vorgegebenem Muster, Längenbestimmung. . .
- Darstellung großer Speicherobjekte
  - besteht potentiell aus vielen Seiten
  - ist eine uninterpretierte Bytefolge
  - OID-Verweis (Adresse) im Satz zeigt auf Objektkopf (header) des großen Objekts
  - unterschiedliche Speicherstrukturen möglich: Kette von Einträgen fester Länge, sequentielle Liste (Datei), B\*-Baum etc.

## Clusterung für lange Felder

- Implementierung im Starburst-Prototyp
  - 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 wird, 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)
- $G \leq \text{MaxSeg}$ : es wird ein Segment angelegt
- $G > \text{MaxSeg}$ : es wird eine Folge maximaler Segmente angelegt; letztes Segment wird auf verbleibende Objektgröße gekürzt

### ■ Verarbeitungseigenschaften

- effiziente Unterstützung von sequentiellen und wahlfreien Lesevorgängen
- einfaches Anhängen und Entfernen von Bytefolgen am Ende des Objektes
- schwieriges Einfügen und Löschen von Bytefolgen in der Mitte des Objektes

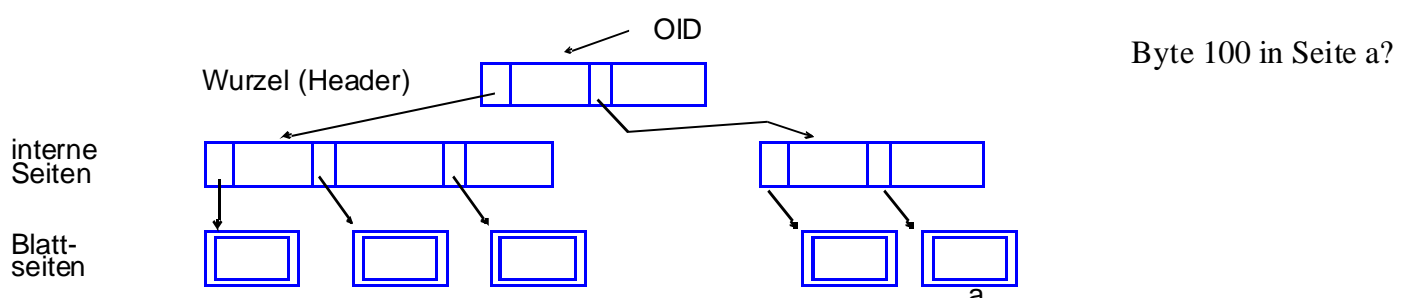
## Baum-artige Verwaltung von langen Feldern / BLOBs

### ■ Physische Darstellung als B\*-Baum

- Blattseiten enthalten die Daten
- interne Seiten (Tabellen) und Wurzel entsprechen einem Index für Bytepositionen
- interne Seiten und Wurzel speichern für jede Kind-Seite Einträge der Form (Zähler, Seiten-#)
- Zähler enthält die maximale Bytenummer, die zum jeweiligen Teilbaum gehört (links stehende Knoten (Einträge) in einer Seite zählen zum Teilbaum).
- Zähler im weitesten rechts stehenden Eintrag der Wurzel enthält Länge des Objektes

### ■ Repräsentation sehr langer dynamischer Objekte

- bis zu 8 GB mit drei Baumebenen
- Speicherplatznutzung typischerweise ~ 80 %



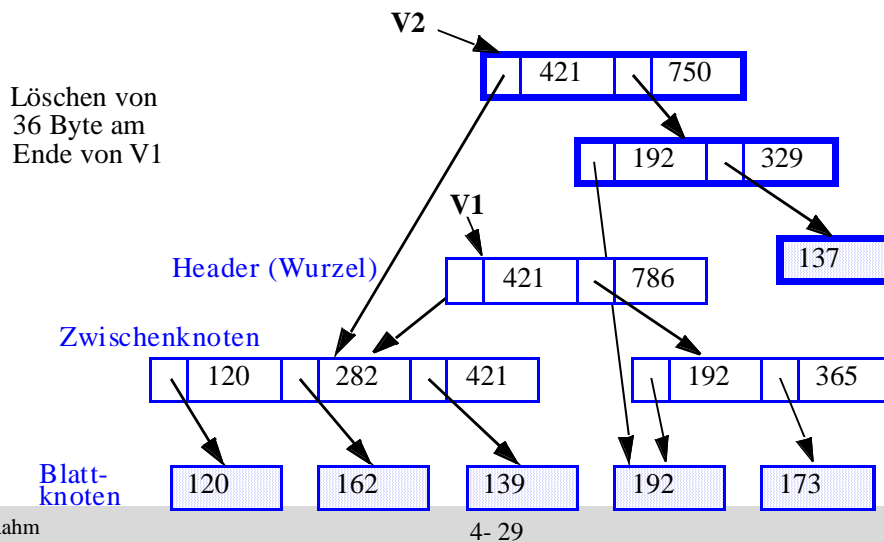
# Baum-artige Verwaltung von BLOBs (2)

## ■ Spezielle Operationen

- Suche nach einem Byteintervall
- Einfügen/Löschen einer Bytefolge an/von einer vorgegebenen Position
- Anhängen einer Bytefolge ans Ende des langen Feldes

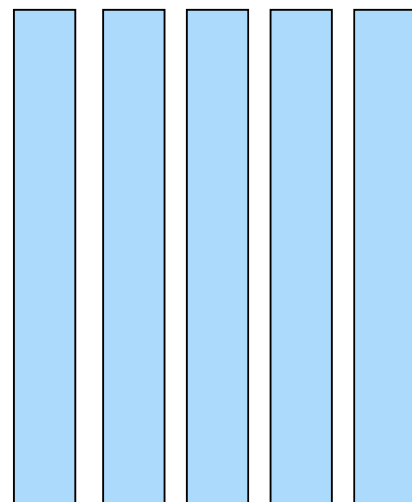
## ■ Unterstützung versionierter Speicherobjekte:

- 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)



# 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
  - Vertica (kommerzielle Version von C-Store), Infobright ICE
  - Monet DBX, SansoussiDB ...
- Hybride Lösungen (z.B. Column-Index in MS SQLServer 2012)



# Spaltenweise Speicherung

- Vorläufer: *DSM (Decomposition Storage Model)*
- vertikale Partitionierung von Tabellen
- Keine replizierte Speicherung des Primärschlüssels pro Partition, falls eindeutiger Zugriff über relative Satznummer
- starke E/A-Einsparungen verglichen mit Zugriff auf vollständige Sätze, wenn nur ein Attribut bzw. wenige Attribute auszuwerten
  - Bsp.: Berechnung von Durchschnittsgehalt, Umsatzsumme ...
- Zusätzliche Einsparungen durch komprimierte Speicherung von Attributwerten

PNR	ANR	W-ORT	GEHALT		PNR	ANR	W-ORT	GEHALT
12345	K02	L	45000	0	12345	K02	L	45000
23456	K02	M	51000	1	23456	K02	M	51000
34567	K03	L	48000	2	34567	K03	L	48000
45678	K02	M	55000	3	45678	K02	M	55000
56789	K03	F	65000	4	56789	K03	F	65000
67890	K12	L	50000	5	67890	K12	L	50000

## Vorteile / Nachteile

- Vorteile Column Store
  - I/O-Einsparungen falls nur wenige Attribute benötigt
  - effiziente Aggregationsmöglichkeiten
  - OLAP-orientiert
  - Oft deutlich bessere Trefferraten im CPU-Cache durch Fokussierung auf relevante Daten
- Nachteile Column Store (-> Vorteile Row Store)
  - Noch relativ neue Technologie
  - Ungünstig für Operationen, die (fast) alle Attribute von Tupeln betreffen, z.B. für Änderungen / effizientes Einfügen neuer Tupel
  - Weniger günstig für OLTP

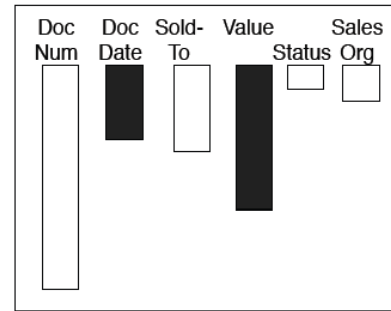
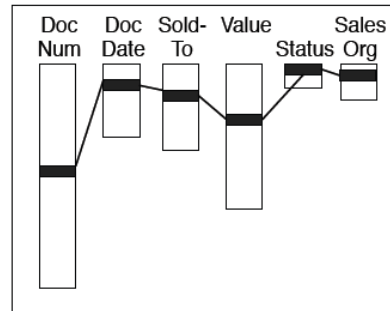


# Anfragen – Column vs. Row Store \*

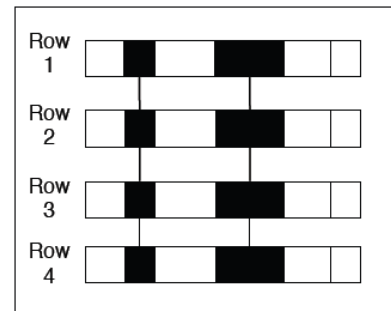
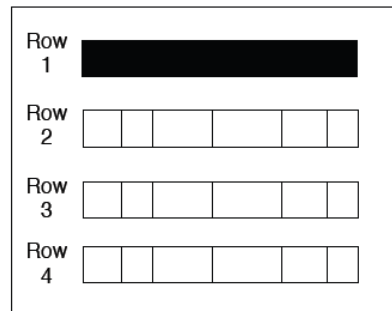
```
SELECT *  
FROM Sales Orders  
WHERE Document Number = '95779216'
```

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

Column Store



Row Store



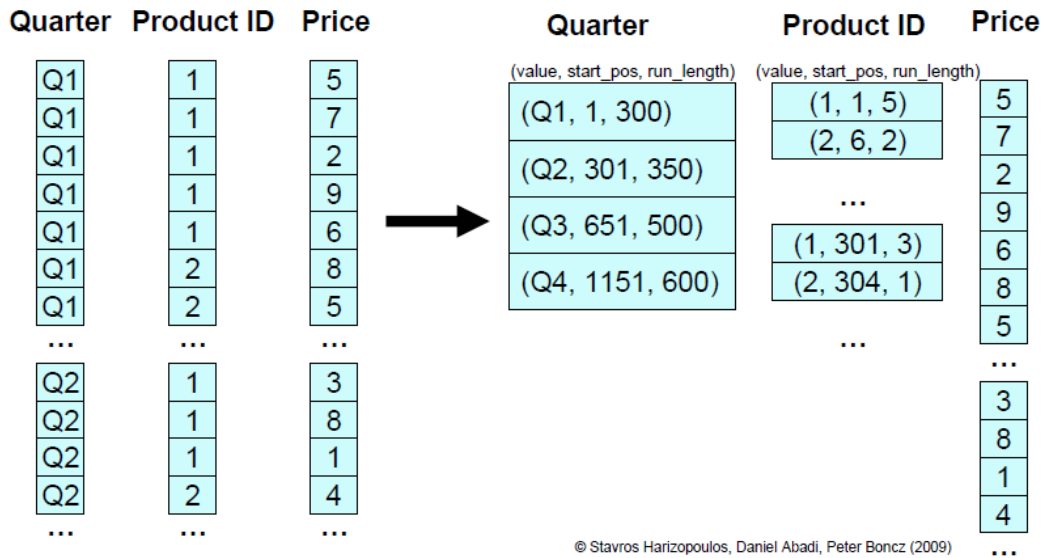
\* Quelle: Hasso Plattner: Enterprise Applications – OLTP and OLAP – Share One Database Architecture

## 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

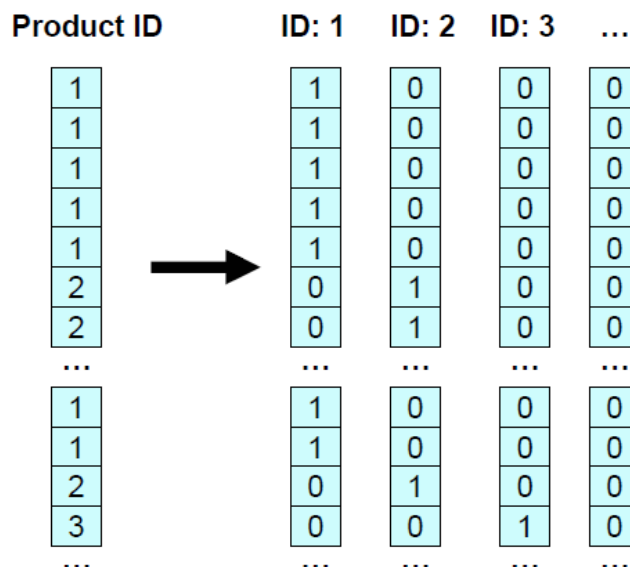


© Stavros Harizopoulos, Daniel Abadi, Peter Boncz (2009)



## Bit Vector Encoding

- 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



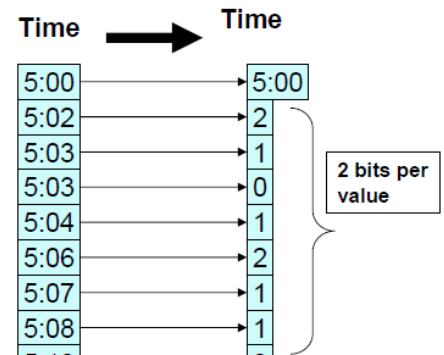
© Stavros Harizopoulos, Daniel Abadi, Peter Boncz (2009)



# Weitere Kompressionstechniken

## ■ Delta / Difference Coding

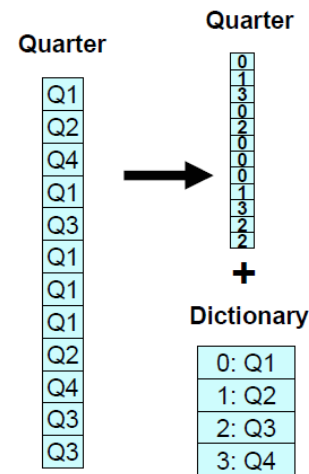
- Speicherung der Differenz zu Vorgängerwerten
- Oft kompakter, v.a. bei Sortierung



## ■ Wörterbuch-Kodierung

- Erfordert keine Sortierung
- Günstig v.a. bei wenigen und langen Attributwerten
- Auswertung auf komprimierten Daten möglich
- Lookup zur Dekomprimierung

Dictionary		IndexVector	
pos	value	pos	value
0	Aachen	0	0
1	Karlsruhe	1	0
2	Leipzig	2	0
3	Münster	3	1
		4	0
		5	0
		6	2
		7	3



© Stavros Harizopoulos, Daniel Abadi, Peter Boncz (2009)



# Zusammenfassung

## ■ Freispeicherinformation auf verschiedenen Ebenen

- Segment (Datei), Seite

## ■ Abbildung von Sätzen:

- meist festes Format, variable Länge
- Spansätze, 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

## ■ Speicherung großer Objekte (BLOBs, "long fields")

- große sequentielle Listen (Clusterung): hohe E/A-Leistung
- B\*-Baum-Technik: flexible Darstellung, moderate Zugriffsgeschwindigkeit

## ■ Column Store-Techniken hilfreich für beschleunigte Anfragen

- Kompressionsverfahren: RLE, Bit Vector- / Delta- / Dictionary-Kodierung

