

Architektur und Implementierung von Datenbanksystemen WS 05/06

Dr. Jens-Peter Dittrich

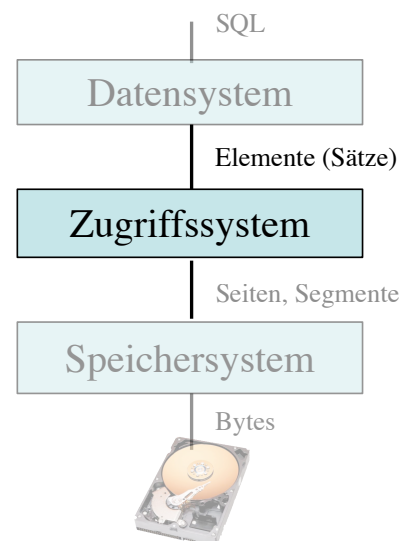
jens.dittrich@inf

www.inf.ethz.ch/~jensdi

Institut für Informationssysteme

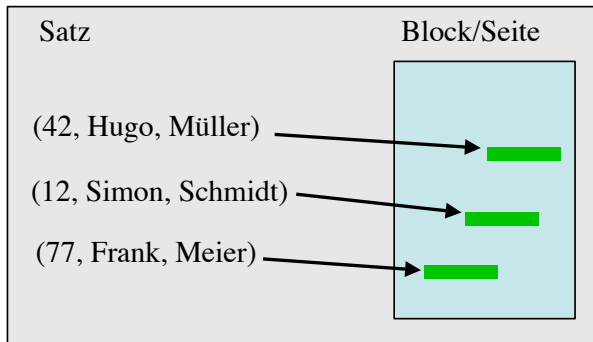
Zugriffssystem

1. Speicherungsstrukturen



Satzverwaltung

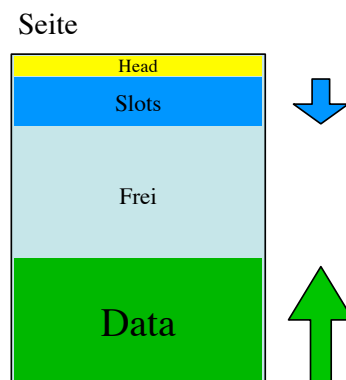
Aufgabe: Abbildung von Sätzen auf Blöcke/Seiten



- Agenda
 - Aufbau einer Seite
 - Satzadressierung
 - Abbildung von Sätzen
 - Satzlayout
 - Speichermodelle
 - NSM
 - DSM
 - PAX
 - Komprimierung
 - Lange Felder
 - Freispeicherverwaltung

Aufbau einer Seite

- Eine Seite besteht aus drei Teilen:
 - fixer Seitenkopf (Metadaten, z.B. Seitennummer, LSN)
 - Slots (Verweise auf einzelne Tupel)
 - Data (Repräsentation der Sätze)
- Slot = (Zeiger, Länge)
- Platz für Slots wird von vorne nach hinten allokiert
- Platz für Sätze wird von hinten nach vorne allokiert



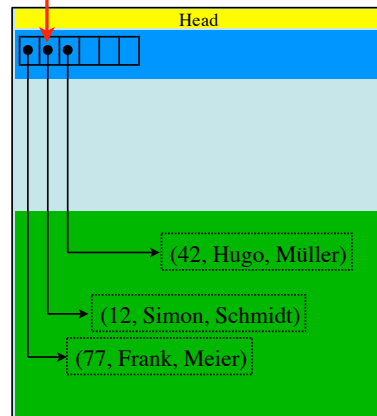
Vorteil: Sätze können problemlos innerhalb einer Seite migrieren

TID-Konzept

- Indirektion über Tuple-ID (TID)
TID = (Seite, Slot)

TID

42	2
----	---

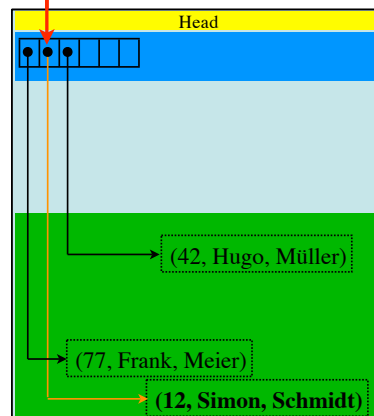


Seite 42

Verschieben von Tupeln innerhalb der Seite

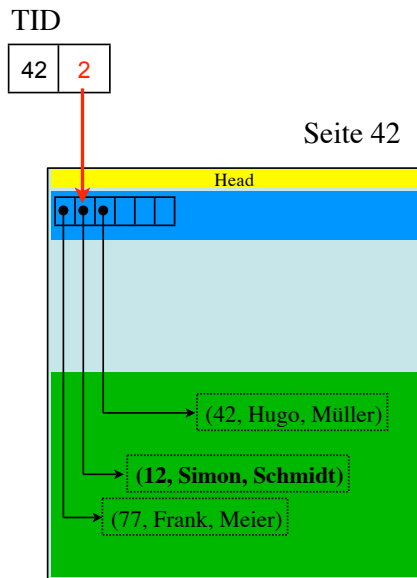
TID

42	2
----	---



Seite 42

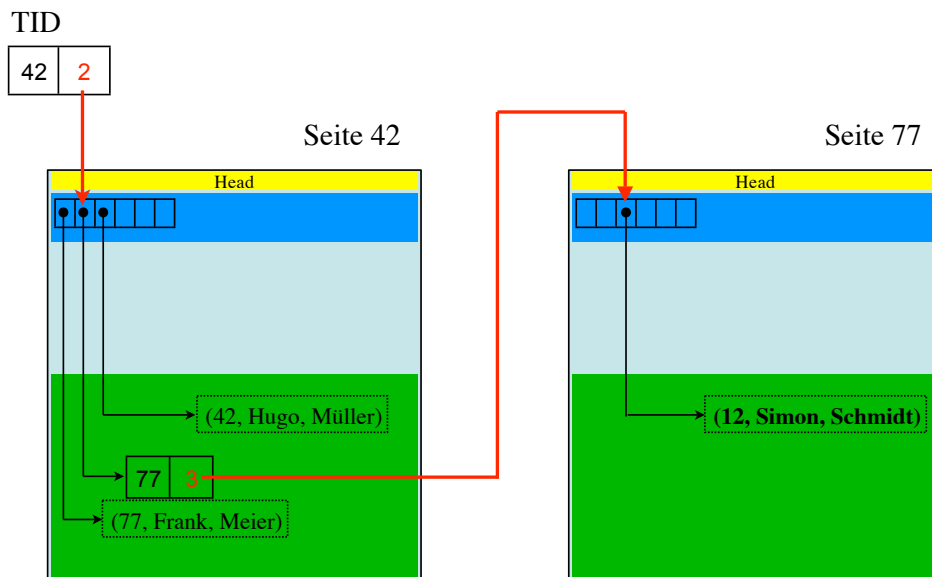
Verschieben von Tupeln von einer der Seite



11. November 2005

Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf

Verschieben von Tupeln von einer der Seite

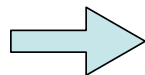


11. November 2005

Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf

TID-Konzept

- Anmerkungen
 - Zugriff trivial, wenn Satz nicht auf eine andere Seite migriert
 - Verschieben von Sätzen über forward TIDs
 - bei weiterem Verschieben:
Abänderung der ersten forward TID



maximal eine Indirektion durch forward TID

- Bewertung
 - minimal 1 Seitenzugriff erforderlich
 - maximal 2 Seitenzugriffe erforderlich (bei forward)

Indirekte Adressierung: Zuordnungstabelle

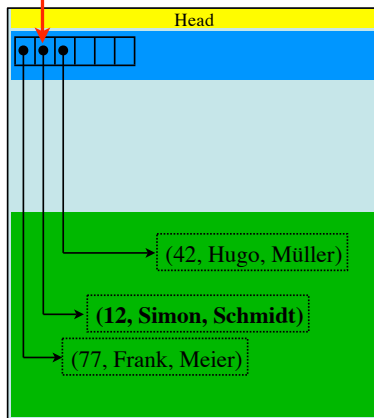
- Idee:
 - 1. halte separate Zuordnungstabelle
 - 2. gib nur logische Satzadressen nach aussen
 - keine forwards
 - bei Verschieben von Sätzen Eintrag in Tabelle anpassen

Verschieben von Tupeln: Zuordnungstabelle

Tabelle

S11	42	2
S43	..	

Seite 42

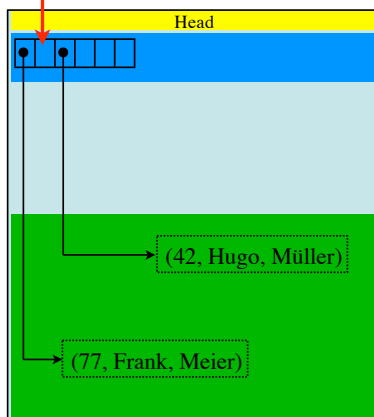


Verschieben von Tupeln: Zuordnungstabelle

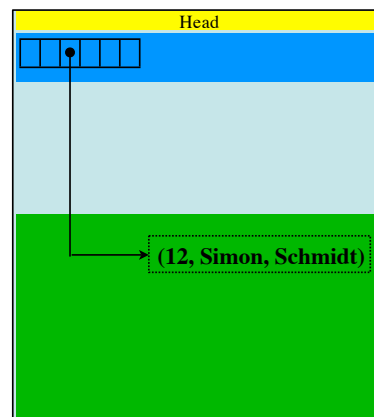
Tabelle

S11	42	2
S43	..	

Seite 42



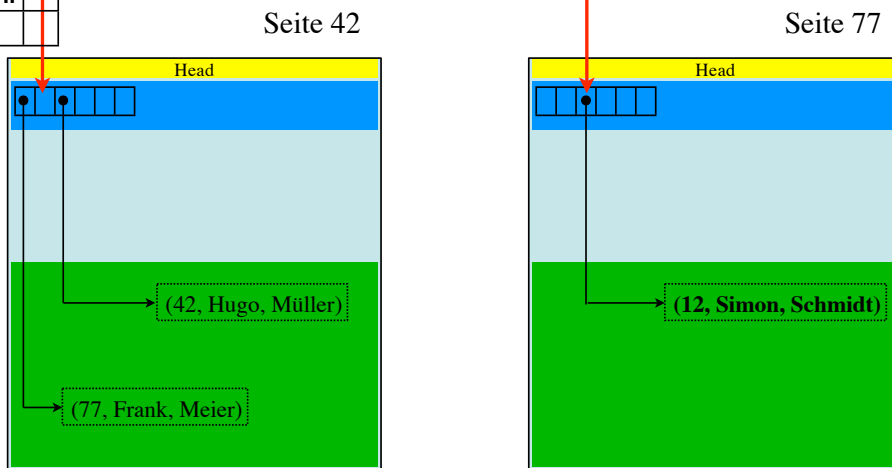
Seite 77



Verschieben von Tupeln: Zuordnungstabelle

Tabelle

S11	77	3
S43	..	



11. November 2005

Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf

Zuordnungstabelle

- Nachteil der Zuordnungstabelle:
 - 2 Seitenzugriffe (1 Zuordnungstabelle + 1 Seite)
- Vorteile:
 - kein Verschnitt in Seiten durch forwards

11. November 2005

Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf

14

Zuordnungstabelle (optimiert, aka PPP)

- Nachteil der Zuordnungstabelle:
 - **2** Seitenzugriffe (1 Zuordnungstabelle + 1 Seite)
- Optimierung:
 - Zugriff auf Zuordnungstabelle kann man einsparen, indem man häufig referenzierte Sätze in separatem Index im Hauptspeicher puffert

S11	42	2
S43	..	

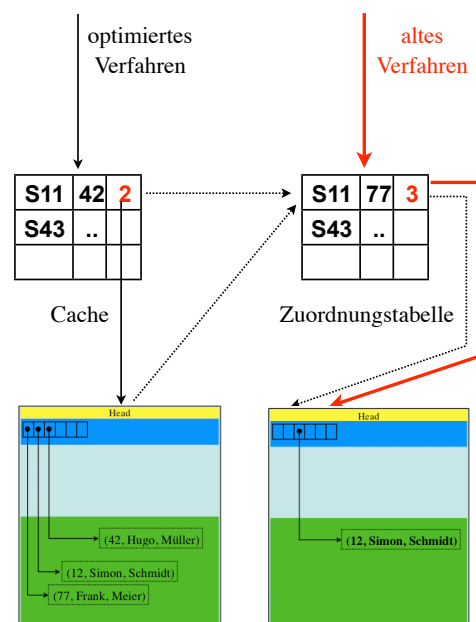
Cache

S11	77	3
S43	..	

Zuordnungstabelle

Zuordnungstabelle (optimiert, aka PPP)

Algorithmus zur Suche:
 Falls Satz-Adresse im Cache
 kein E/A Zugriff auf Zuordnungstabelle
 Adresse = Cache-Adresse
 Falls Satz nicht an Adresse gefunden:
 E/A-Zugriff auf Zuordnungstabelle
 Sonst
 E/A-Zugriff auf Zuordnungstabelle



Weitere Optimierung

- Beobachtung: Zuordnungstabelle entspricht grosser Indexseite!
- Warum nicht die Abbildung

logische Satzadresse → physische Satzadresse

in einer Indexstruktur speichern, z.B. einem B⁺-Baum?

- Bewertung
 - Vorteil: Sortierung der Einträge garantiert
 - Vorteil: keine zusätzliche Freispeicherverwaltung notwendig
 - Nachteil: teurer Zugriff durch mehrstufigen B⁺-Baum (für jeden Satzzugriff!)
 - Nachteil: Speicherauslastung schlecht

Abbildung von Sätzen

- Trennung von Metadaten und Nutzdaten
 - **Metadaten:** Daten im Katalog
 - Attributname
 - Typ
 - **Nutzdaten:** Daten auf der Seite
 - Wert
- N.B.
 - In der XML-Welt werden Metadaten und Nutzdaten gemeinsam abgespeichert:

```
<satz>  
  <vorname> hugo </vorname>  
  <name> müller </name>  
</satz>
```

Satzlayout

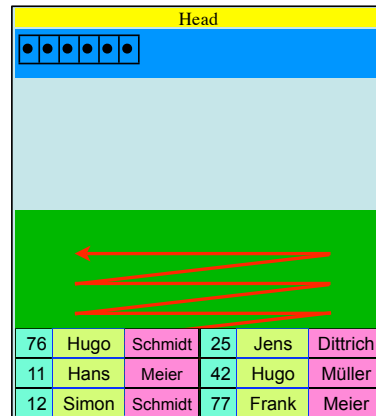
- Teil fixer Länge
 - speichert alle Werte, die einen Typ fixer Länge haben
 - z.B. numeric(10,2), date, char[42]
 - Vorteil: direkte Adressberechnung
$$\text{address} = \text{sizeof}(\text{type}) * \text{pos}$$
- Teil variabler Länge
 - z.B. varchars
 - speichert Länge und Zeiger im fixen Teil
 - speichert Wert im variablen Teil
 - Nachteil: indirekte Adressberechnung
$$\text{address} = \text{Zeiger}$$
 - Wichtig: die Anwesenheit von varchars stört den direkten Zugriff auf Typen fixer Länge nicht!

Satzlayout

- NULL-Werte
 - kleine Bitmap fixer Länge am Anfang des Satzes
 - "1" falls Attribut den Wert NULL hat, "0" sonst
 - Vorteil: einfach und schnell

Beispiel für Satzlayout

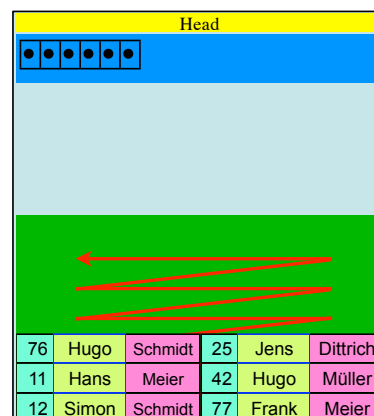
Key	Vorname	Name
77	Frank	Meier
12	Simon	Schmidt
42	Hugo	Müller
11	Hans	Meier
25	Jens	Dittrich
76	Hugo	Schmidt



zeilenweises Auffüllen
der Seite

n-ary Storage Model (NSM)

Key	Vorname	Name
77	Frank	Meier
12	Simon	Schmidt
42	Hugo	Müller
11	Hans	Meier
25	Jens	Dittrich
76	Hugo	Schmidt

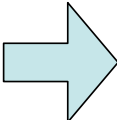


- Sätze werden sequentiell abgespeichert
- Alle Attribute eines Satzes werden lokal benachbart gespeichert

Frage: Was passiert in der Speicherhierarchie?

Decomposition Storage Model (DSM)

RID	Key	Vorname	Name
1	77	Frank	Meier
2	12	Simon	Schmidt
3	42	Hugo	Müller
4	11	Hans	Meier
5	25	Jens	Dittrich
6	76	Hugo	Schmidt



RID	Key
1	77
2	12
3	42
4	11
5	25
6	76

RID	Vorname
1	Frank
2	Simon
3	Hugo
4	Hans
5	Jens
6	Hugo

RID	Name
1	Meier
2	Schmidt
3	Müller
4	Meier
5	Dittrich
6	Schmidt


- Teile Ausgangstabelle in 2-attributige Untertabellen auf
- RID muss ggf. nicht separat gespeichert werden

Decomposition Storage Model (DSM)

RID	Key
1	77
2	12
3	42
4	11
5	25
6	76

RID	Vorname
1	Frank
2	Simon
3	Hugo
4	Hans
5	Jens
6	Hugo

RID	Name
1	Meier
2	Schmidt
3	Müller
4	Meier
5	Dittrich
6	Schmidt



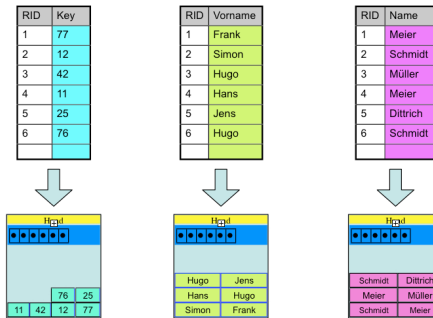
Head			
•	•	•	•
11	42	12	77

Head	
•	•
Hugo	Jens
Hans	Hugo
Simon	Frank

Head	
•	•
Schmidt	Dittrich
Meier	Müller
Schmidt	Meier

Decomposition Storage Model (DSM)

- Optimiert für wenige Attribute
- Vorteil: Sehr Effizient bei Zugriff auf nur ein oder wenige Attribute
- Nachteil: Ineffizient bei Zugriff auf viele Attribute
- Nachteil: Satz verstreut über mehrere Seiten.



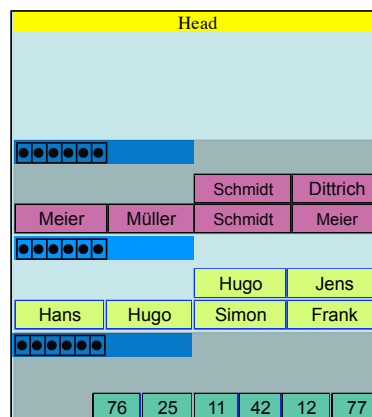
➔ Änderung der Satzverwaltung notwendig!

- Literatur:
 - Don S. Batory: On Searching Transposed Files. ACM Trans. Database Syst. 1979.
 - George P. Copeland, Setrag Khoshafian: A Decomposition Storage Model. SIGMOD 1985

Partition Attributes Across (PAX)

- Idee: Daten **innerhalb** der Seite aufteilen

Key	Vorname	Name
77	Frank	Meier
12	Simon	Schmidt
42	Hugo	Müller
11	Hans	Meier
25	Jens	Dittrich
76	Hugo	Schmidt



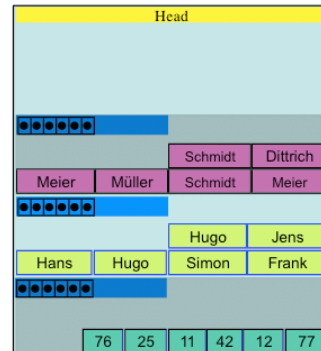
Unterseite 1

Unterseite 2

Unterseite 3

Partition Attributes Across (PAX)

- Vorteile
 - verbessert Lokalität innerhalb eines Attributs
 - Daten werden nur innerhalb der Seite anders angeordnet
 - ➔ keine Änderung der Satzverwaltung notwendig
 - Satzrekonstruktionskosten gering
 - 15%-2x Performanzgewinne gegenüber NSM
- Nachteile
 - für DSS nicht so gut wie DSM
- Literatur: Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, Marios Skounakis: Weaving Relations for Cache Performance. VLDB 2001.



Komprimierung

- Orthogonal zum verwendeten Satzlayout
- einzelne Felder vs. ganze Sätze (vs. komplette Seite vs. Extent vs. ...)
- sehr effizient für DSM
- Literatur (Auswahl):
 - Meikel Pöss, Dmitry Potapov: Data Compression in Oracle. VLDB 2003.
 - Balakrishna R. Iyer, David Wilhite: Data Compression Support in Databases. VLDB 1994.
 - Till Westmann, Donald Kossmann, Sven Helmer, Guido Moerkotte: The Implementation and Performance of Compressed Databases. SIGMOD Record 29(3) (2000)
 - Managing Gigabytes: Compressing and Indexing Documents and Images by Ian H. Witten, Alistair Moffat, and Timothy C. Bell. 2nd edition. Morgan Kaufmann Publishing. 1999.

Realisierung langer Felder

- Problem:
 - Was ist mit Datensätzen, die grösser sind als eine Seite?
 - Bsp.: Blobs (Binary Large Objects)
- 1. Lösung
 - Teile Satz in seitengrosse Stücke auf
 - Lege Index (byteoffset → Block) für Stücke an als Hash-Tabelle oder B⁺-Baum
- 2. Lösung
 - Lagere Satz aus in extra-Bereich (z.B. Dateisystem des BS)
 - Speichere nur Verweis auf Satz in DB-Seite
 - (siehe SQL/MED 1999-Standard)

Freispeicherverwaltung

- Aufgabe:
Finde Platz für neuen Satz
- Native Realisierung (Append Only):
 - Betrachte immer nur die zuletzt erzeugte Seite
 - Wenn Satz in diese Seite passt OK
 - Sonst: erzeuge neue Seite
- Bewertung
 - sehr schnelles Einfügen
 - schlechte Speicherauslastung

Append Only(n)

- Verallgemeinerung von Append Only:
 - Betrachte immer nur die n zuletzt erzeugten Seiten
 - Wenn Satz in eine dieser Seiten passt OK
 - Sonst: erzeuge neue Seite
- Bewertung
 - sehr schnelles Einfügen
 - schlechte Speicherauslastung

Best Fit, First Fit, Next Fit

- Best Fit:
 - suche geeignetste Seite
 - Aufwand = lineare Suche in Liste
- First Fit:
 - nimm erste Seite die passt
- Next Fit:
 - wie First Fit, aber: starte Suche bei letzter Position

Hybride Verfahren HY(n,u)

- Prinzip:
 - Falls Speicherauslastung besser als u:
 - verwende Append Only(n)
 - Sonst
 - verwende Next-Fit
- Bewertung: guter Kompromiss

Freispeichertabelle

- Idee: Speichere für jede Seite verfügbaren Speicherplatz
- Realisierung: Tabelle auf Segmentebene
 - mit exakter Speicherbelegung

Seite	1	2	3	4	5	
Bytes frei	f ₁	f ₂	f ₃	f ₄	f ₅	

2 Byte pro Eintrag

2 Byte pro Eintrag

- mit unscharfer Speicherbelegung

Seite	1	2	3	4	5	
Bytes frei	f ₁	f ₂	f ₃	f ₄	f ₅	

2 Byte pro Eintrag

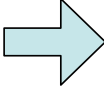
k Bits pro Eintrag

freier Platz $\leq (f_i / 2^k) * \text{Seitengröße}$

Space Map

- Nachteil der Freispeichertabelle: lineare Suche nach geeigneter Seite
- Lösung "Space Map":
 - Invertieren der Tabelle

Seite	1	2	3	4	5	
Bytes frei	f ₁	f ₂	f ₃	f ₄	f ₅	



Bytes frei	f ₄	f ₂	f ₃	f ₅	
Seite	4	1,2	3	5	

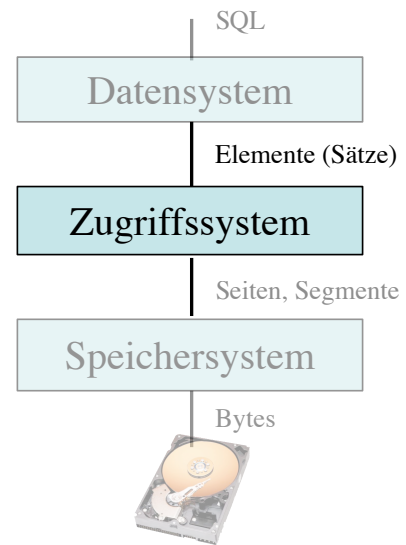
- Index über "Bytes frei" Attribut
- binäre Suche
- einfachste Variante: 0 oder 1, Seite hat Platz oder nicht
- grobgranulare Variante: 00, 01, 10, 11, Seite hat 0%, 25%, ... Platz
- (Diskussion analog zur Freispeichertabelle)

Space Map

- Trade-off: Granularität vs. Speicherverbrauch
- Trade-off: Granularität vs. Performanz
- Vorteil: effiziente Suche für Best Fit: $O(1)$
- Nachteil: update-Kosten für Space-Map

Zugriffssystem

2. Eindimensionale Zugriffspfade (Teil 1)

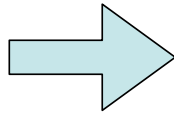


Motivation

- **Anfragen:**
 - Wie ist die Adresse von Student mit Matrikelnummer 424342?
 - Welche Studenten besuchen weniger als 2 Vorlesungen dieses Semester?
 - Welche Studenten wohnen nicht im Kanton Zürich?
- **Wie beantwortet das DBMS diese Anfragen?**
 - Alle Studenten-Datensätze anschauen (Sequentieller Zugriffspfad, siehe letzte VL-Stunde)
 - Daten geschickt organisieren, so dass die benötigten Sätze schnell gefunden werden (Baumstrukturierte Zugriffspfade, heute)

Was heisst indizieren?

- Abbildung:
 - Schlüssel → Menge von Sätzen
 - Schlüssel muss nicht mit Primärschlüssel der Relation übereinstimmen



Zugriffspfade sorgen für die effiziente Implementierung dieser Abbildung

Zugriffspfade

- Oft kann auf denselben Satz über verschiedene "Pfade" zugegriffen werden
- Zugriffspfad = Möglichkeit des Zugriffs auf einen Satz
- Zugriffspfade haben **sehr grossen** Einfluss auf die Effizienz der Anfragebearbeitung des DBMS

Primärer vs. Sekundärer Zugriffspfad

- Zwei Klassen von Zugriffspfaden
 1. Zugriffspfade für Primärschlüssel

Beispiel:

```
SELECT *  
FROM Mitarbeiter  
WHERE Personalnummer = 42
```

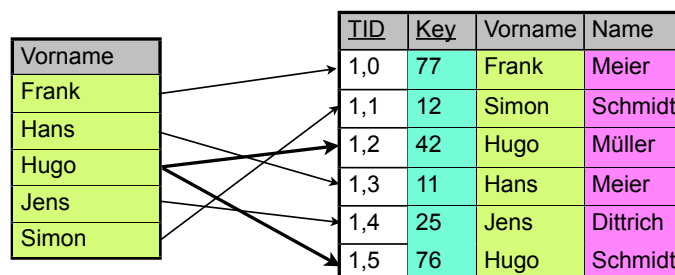
2. Zugriffspfade für Sekundärschlüssel

Beispiel:

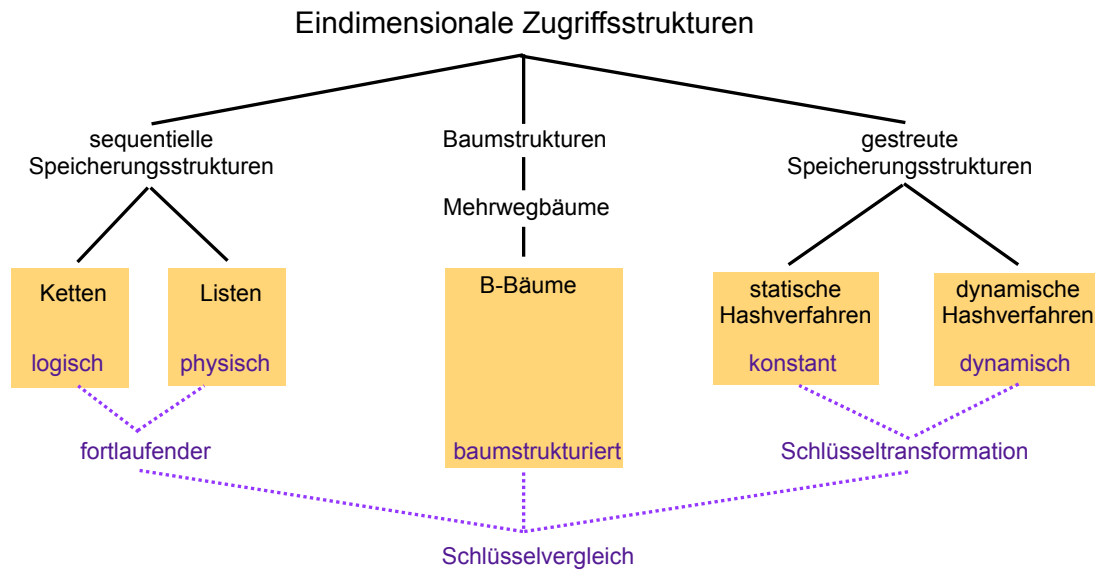
```
SELECT *  
FROM Mitarbeiter  
WHERE Wohnort = 'ZH'
```

Sekundäre Zugriffspfade und Invertierung

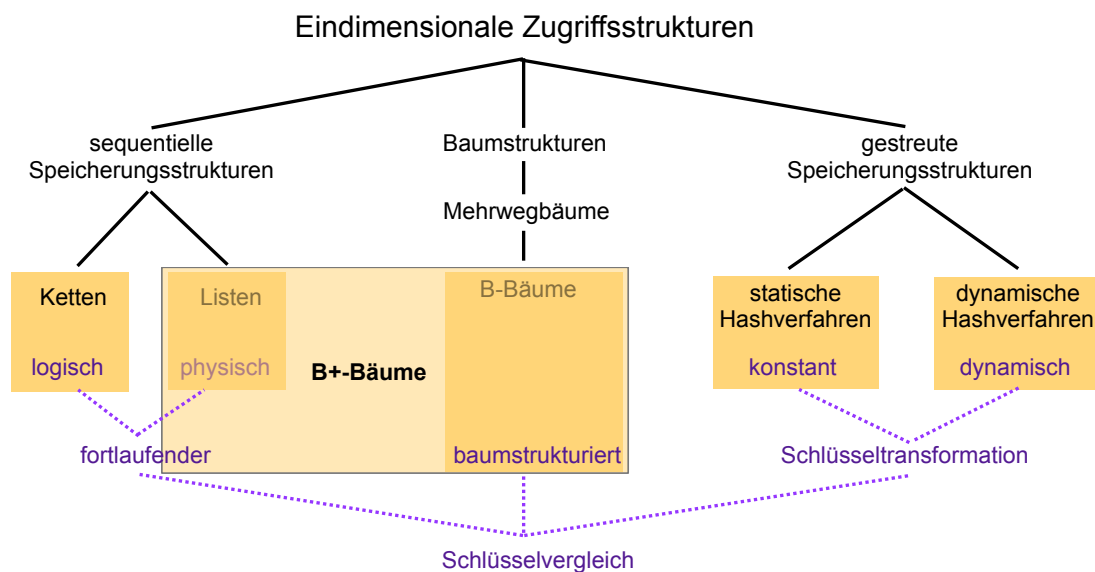
- Suche über Sekundären Zugriffspfad kann mehr als ein Ergebnis liefern (1:n Beziehung zwischen Schlüsseln und Ergebnissen)



Eindimensionale Zugriffspfade

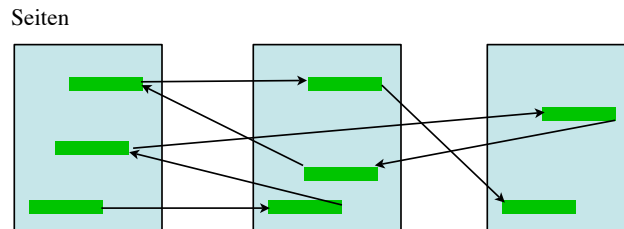


Eindimensionale Zugriffspfade



Sequentielle Zugriffspfade: Ketten

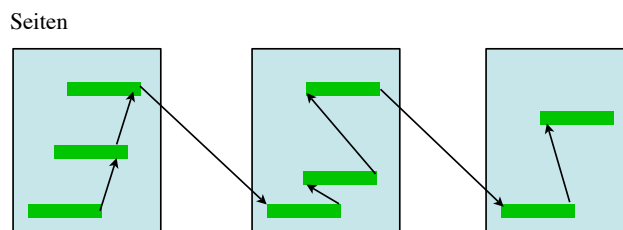
- Liste von Sätzen ohne Rücksicht auf Seiten



- Bewertung
 - sehr schlechtes E/A-Verhalten
 - Worst Case: 1 wahlfreier Zugriff pro Satz
 - spielt keine Rolle in DBMS

Sequentielle Zugriffspfade: Liste

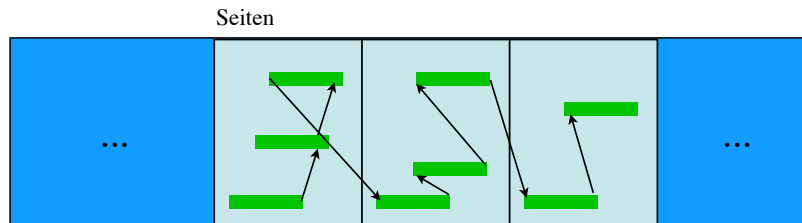
- Liste von Sätzen gruppiert in Seiten
- Sätze sind physisch **geclustert**



- Bewertung
 - besseres E/A-Verhalten
 - Worst Case: 1 wahlfreier Zugriff **pro Seite**
 - spielt eine Rolle in DBMS

Sequentielle Zugriffspfade: Sequenz

- Liste von Sätzen gruppiert in Seiten
- Sätze **und** Blöcke sind physisch geclustert



- Bewertung
 - bestmögliches E/A-Verhalten
 - Worst Case: 1 wahlfreier Zugriff + sequentielle Reads
 - sehr wichtig für DBMS
 - Nachteil: schwierig aufrecht zu erhalten bei vielen inserts und updates

Bäume

- Binärbäume
 - Nicht geeignet für DBMS
 - Knoten lassen sich nur schwer auf Seiten abbilden
- Digitalbäume
 - spezielle Anwendungen
 - wichtig für nicht-relationale Daten
- B-Bäume
 - wichtigste Datenstruktur im Datenbankbereich
 - Vorteil: extrem vielseitig, lässt sich leicht erweitern
 - seit über 30 Jahren Forschung, immer noch wichtige Neuerungen
 - viele Indexstrukturen mit ähnlichen Ideen (R-Baum, M-Baum)

B-Bäume

- Grundlagen: siehe Algorithmen und Datenstrukturen (Widmayer)
- Agenda (für nächste Woche):
 - Grundlagen (Wiederholung)
 - ISAM
 - Bulk-Loading
 - Präfix B⁺-Bäume
 - Präfix/Suffix-Komprimierung
 - Cache-sensitive B⁺-Bäume
 - Primär vs. Sekundärindex
 - Clustered Index
 - Sekundärindexe und Google
 - ...

Nächste Woche

Zugriffssystem

2. Eindimensionale Zugriffspfade (Teil 2)

