Dirk Habich

Data Management Foundations

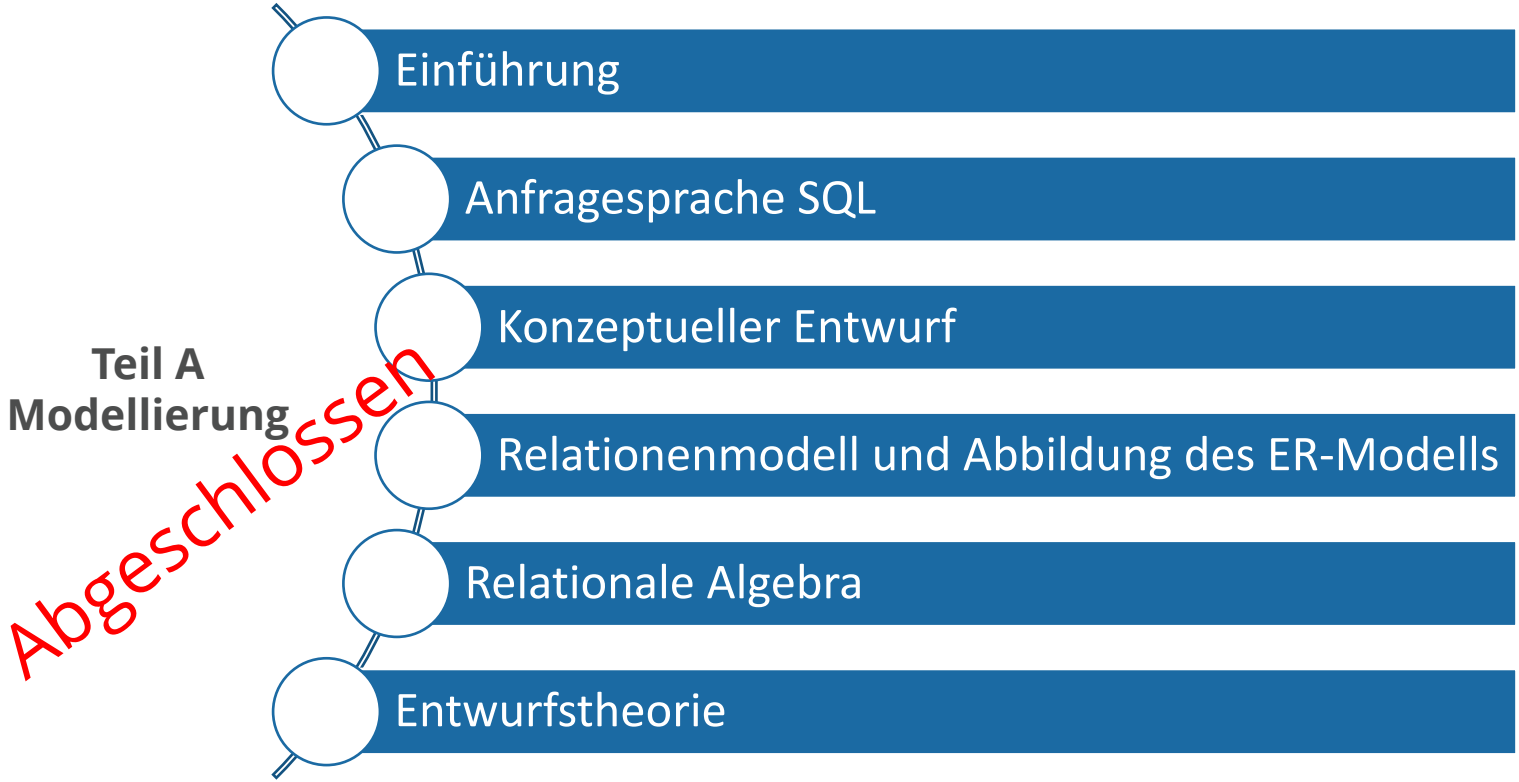
Architekturüberblick

TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Research Group

Gliederung der Lehrveranstaltung

Dresden Database Research Group

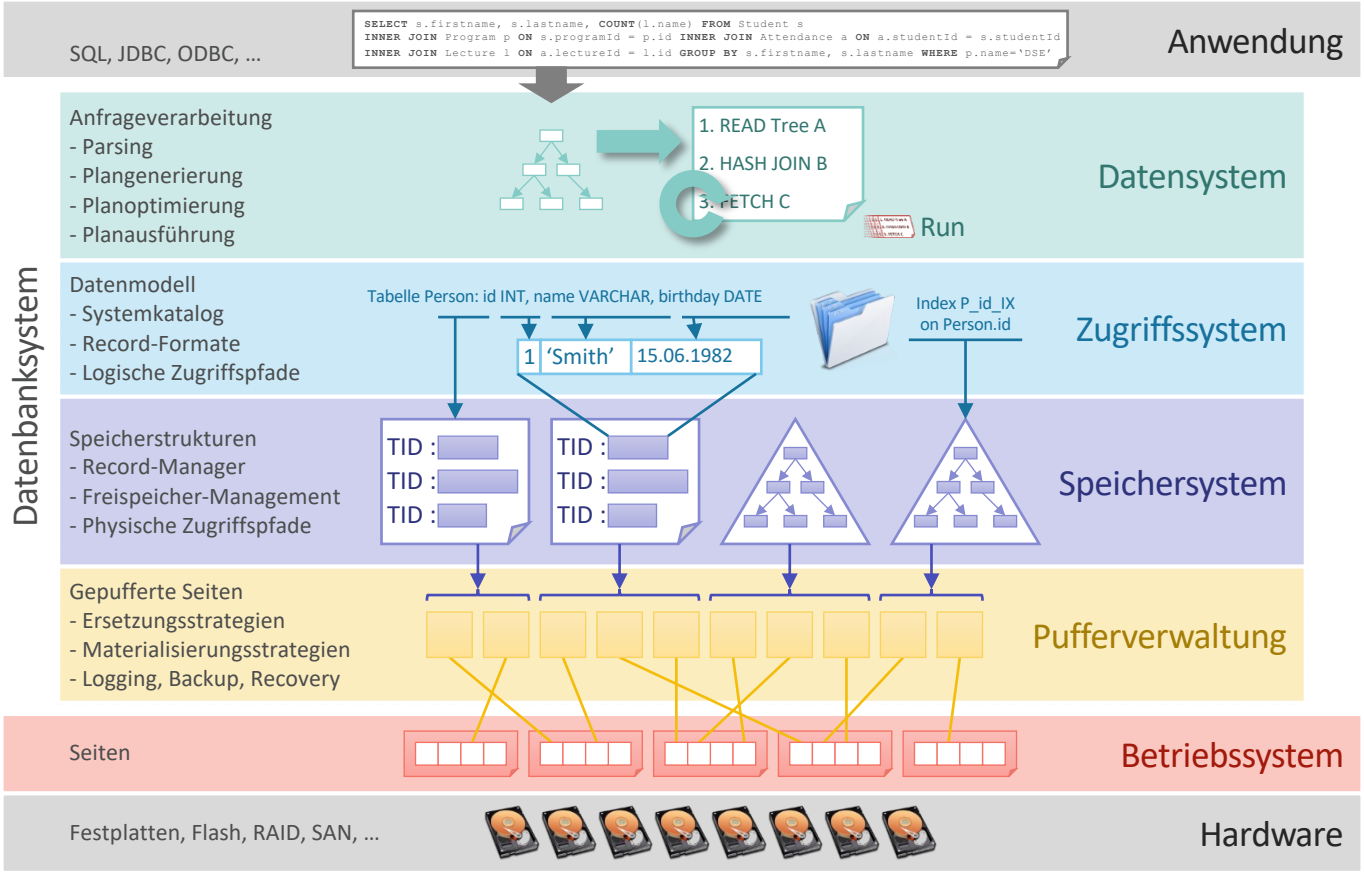


Gliederung der Lehrveranstaltung

Dresden Database Research Group

Teil B System

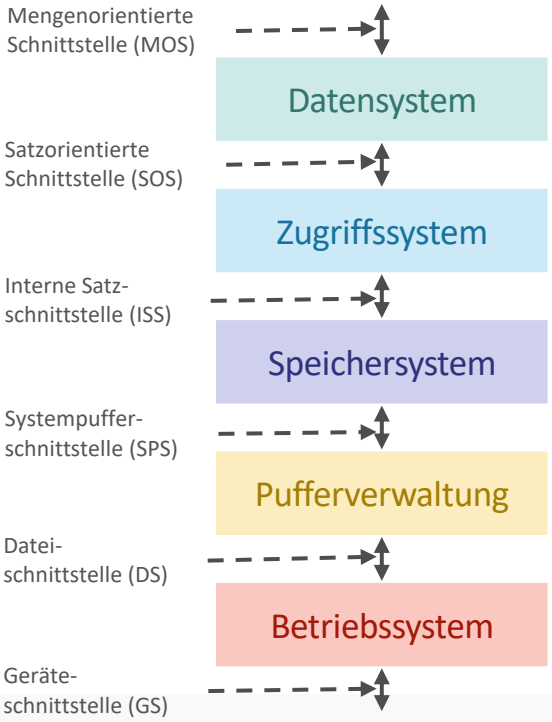
Datenbankarchitektur – Blue Print



5-Schichten-Architektur

Dresden Database Research Group

Übersicht über Datenobjekte und Objektstrukturen



Datensystem	Relationen Sichten	SQL: select … from
Zugriffssystem	Externe Sätze, Scans, Indexstrukturen	FIND NEXT satz STORE satz
Speichersystem	interne Sätze, Bäume, Hashtabellen	Speichere internen Satz INSERT in B-Baum
Pufferverwaltung	Segmente Seiten	Bereitstellen Seite j Freigeben Seite j
Betriebssystem	Dateien Blöcke	Lies Block k Schreibe Block k
	Zylinder Spuren	Treiber

5-Schichten-Architektur (2)

Dresden Database Research Group

MOS: mengenorientierte Schnittstelle

- Deklarative Datenmanipulationssprache auf Tabellen und Sichten (etwa SQL)

SOS: satzorientierte Schnittstelle

- Navigierender Zugriff auf interner Darstellung der Relationen
- Manipulierte Objekte: typisierte Datensätze und interne Relationen sowie logische Zugriffspfade (Indexe)
- Aufgaben des Datensystems: Übersetzung und Optimierung von SQL-Anfragen

ISS: interne Satzschnittstelle

- Interne Tupel einheitlich verwalten, ohne Typisierung
- Speicherstrukturen der Zugriffspfade (konkrete Operationen auf B-Bäumen und Hash-Tabellen), Mehrbenutzerbetrieb mit Transaktionen

SPS: Systempufferschnittstelle

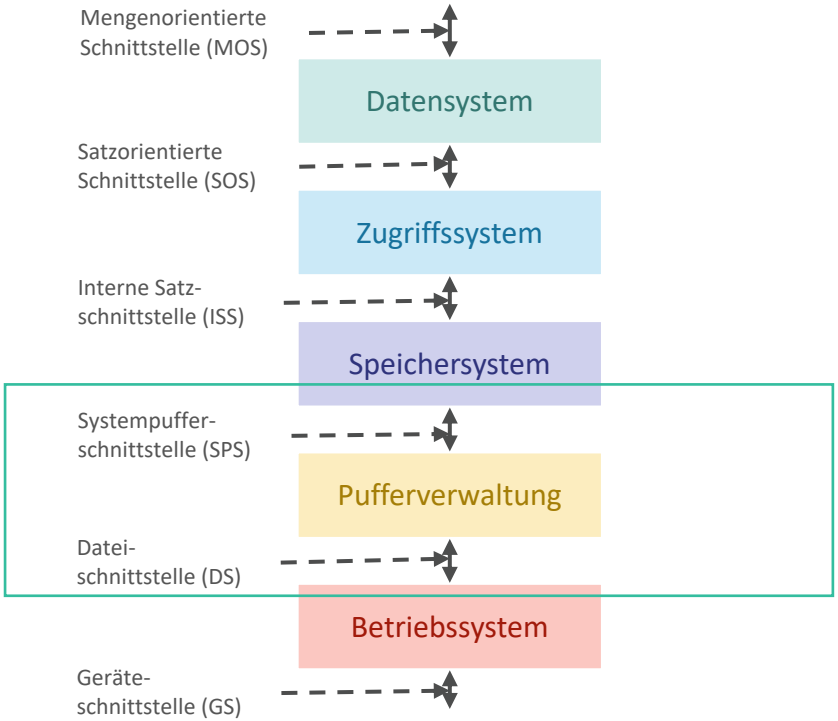
- Umsetzung auf interne Seiten eines virtuellen linearen Adressraums
- Typische Operationen: Freigeben und Bereitstellen von Seiten, Seitenwechselstrategien, Sperrverwaltung, Schreiben des Protokolls

DS: Dateischnittstelle

- Umsetzung auf Geräteschnittstelle

Pufferverwaltung

Einordnung in die Schichtenarchitektur

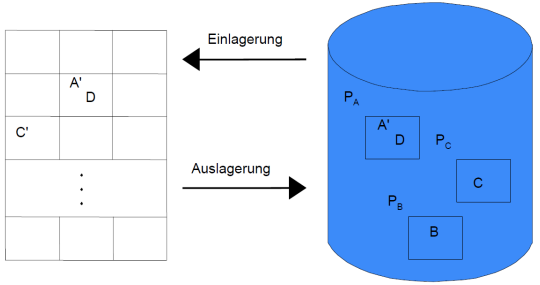


Arbeitsweise und Eigenschaften

Dresden Database Research Group

Arbeitsweise und Rolle der Pufferverwaltung in einem DBS

- Lese- und Schreiboperationen werden über einen Systempuffer abgewickelt
- Puffer kann nur einen Bruchteil der gesamten Datenbank aufnehmen
- Puffer besteht aus
  - Seitenstrukturierten Segmenten (Adressräume)
  - Pufferkontrollblock (Aufnahme von Verwaltungsinformation)

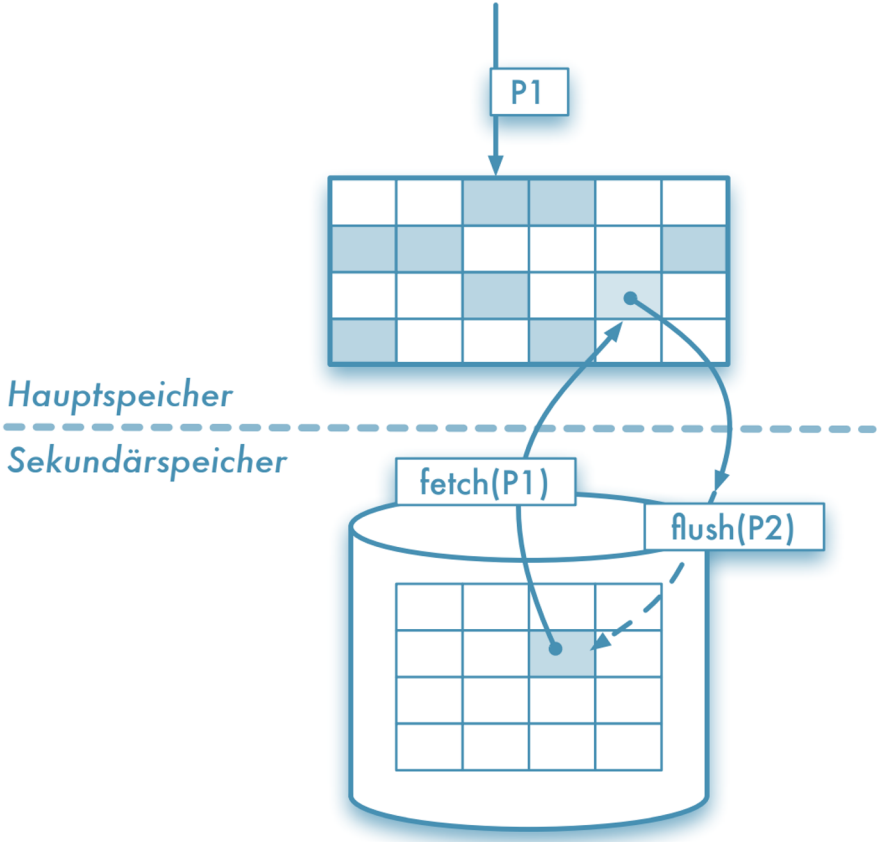


Eigenschaften einer DBS Systempufferverwaltung

- Im Prinzip: normale Pufferverwaltung mit diversen Verdrängungsstrategien
- Aber: beim DB-Puffer ist der Nutzer bekannt (die darüber liegende Schicht!), so dass Anwendungswissen (Kontextwissen) in die Pufferverwaltung einfließen kann

Lokalität als Maß für Seitenverdrängung

Seitenersetzung schematisch



Dienste eines Systempuffers

Dresden Database Research Group

Systematik der Aufrufe

- Bereitstellen (Logische Referenz)
  - Bereitstellen der angeforderten Seite im Puffer; dabei evtl. Verdrängen einer „älteren” Seite gemäß der Ersetzungsstrategie und physisches Einlesen der neuen Seite
- FIX
  - Festhalten einer Seite im Puffer, so dass die Adressierbarkeit des Seiteninhalts gewährleistet ist (oft mit Bereitstellen automatisch durchgeführt)
- UNFIX
  - Aufheben eines FIX; macht die Seite frei für die Ersetzung
- Änderungsvermerk
  - Eintragen eines Vermerks in die Seite, dass sie beim Verdrängen aus dem Puffer physische geschrieben werden muss
  - evtl. Sicherstellen eines Before-Image, falls dies die Einbring-Strategie erfordert (der Änderungsvermerk muss VOR Ausführen der Änderung gemacht werden)
- Schreiben
  - Sofortiges Ausschreiben der Seite aus dem Puffer in die Datenbank

Seitenersetzungsstrategien

Dresden Database Research Group

Klassifikation der Verfahrensklassen

Preplanning	Prefetching	Demand Fetching
▪ Programmanalyse ▪ Vorabuntersuchung der Anforderung → Große Fehlerrate	▪ Physische Datenstrukturierung ▪ Clusterbildung ▪ Einbezug von Verarbeitungs-wissen → Datenmodellbezogen → Spekulative Entscheidungen	▪ Keine Vorausaktion ▪ Reaktion auf Anforderung → Lokalitätserhaltung im DB-Puffer

Grundannahme

- Referenzverhalten der nächsten Zukunft ist ähnlich dem der jüngsten Vergangenheit

Merkmale gängiger Strategien

Dresden Database Research Group

Alter einer Seite

- Alter seit der Einlagerung (globale Strategie (G))
- Alter seit dem letzten Referenzzeitpunkt (Strategie des jüngsten Verhaltens (J))
- Alter wird nicht berücksichtigt (-)

Referenzierung einer Seite

- Berücksichtigung aller Referenzen (globale Strategie (G))
- Berücksichtigung der letzten Referenzen (Strategie des jüngsten Verhaltens (J))
- Berücksichtigung keiner Referenz (-)

Ziel

- Annäherung an optimale Strategie

Klassifikation gängiger Strategien

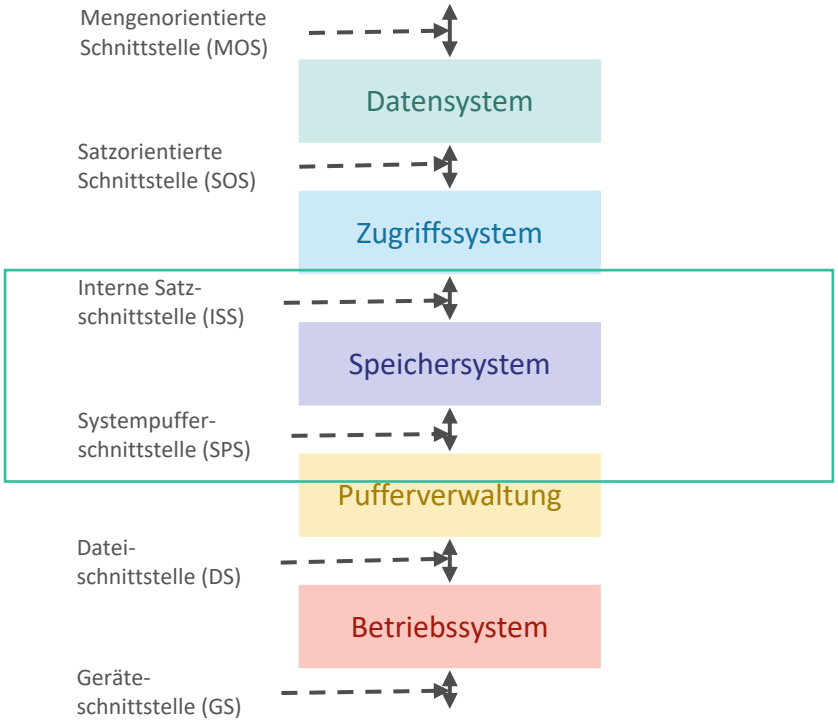
Dresden Database Research Group

Verfahren	Prinzip	Alter	Anzahl
FIFO	Älteste Seite ersetzt	G	-
LFU (least frequently used)	Seite mit geringer Nutzungshäufigkeit ersetz	-	G
LRU (least recently used)	Seite ersetzen, die am längsten nicht referenziert wurde	J	J
DGCLOCK (dyn. generalized clock)	Protokollierung der Ersetzungs-häufigkeiten wichtiger Seiten	G	JG
LRD (least reference density)	Ersetzung der Seite mit geringster Referenzdichte	JG	G

G → globale Strategie
J → Strategie des jüngsten Verhaltens
- → nicht berücksichtigt
JG → Beides

Speichersystem - Record Management

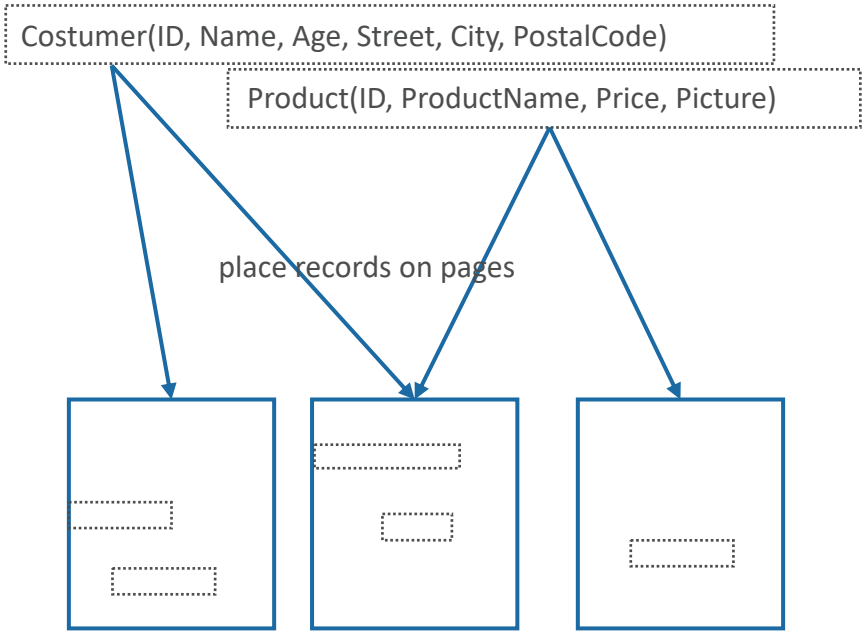
Einordnung in die Schichtenarchitektur



Record Management

Dresden Database Research Group

Graphical Presentation



Key Aspects

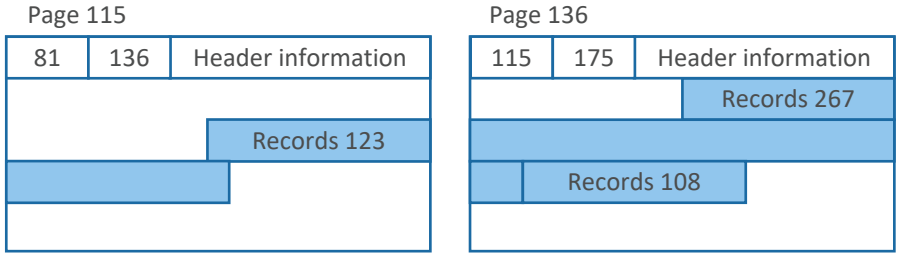
- Representation of complete records on pages
- Organization of structure formats for records
- Addressing of Records
  - Allocation Table
  - TID-concept
- Heap management (“Freispeicherverwaltung”)

Organization of Pages

Dresden Database Research Group

Concatenation

- Pages are linked together by double linked lists
- Recording of free pages: heap management (Freispeicherverwaltung)



L_P = page size
L_{PH} = length of page header

Page Header

- Information about previous and and following page
- Optionally also number of the page itself
- Information about the type of record (Table Directory)
- Information about free space

Abbildung von Sätzen auf Seiten

Dresden Database Research Group

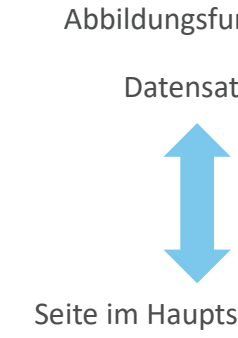
Wichtig

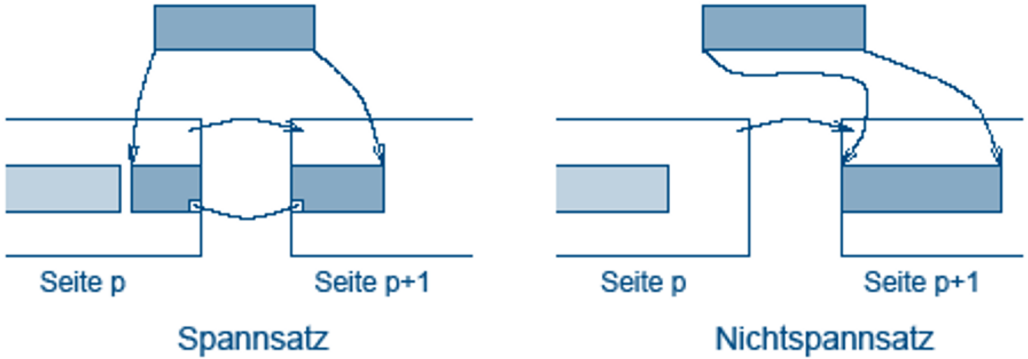
- **Seite** fester Länge als ‘Verarbeitungseinheit’
- **Datensatz** beliebiger Länge als ‘Verarbeitungseinheit‘

Entkopplung von

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

Spann bzw. Nicht-Spannsatz





Satzadressierung

Dresden Database Research Group

Problem

- Eindeutige und lebenslange Adressierung von Sätzen
- 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

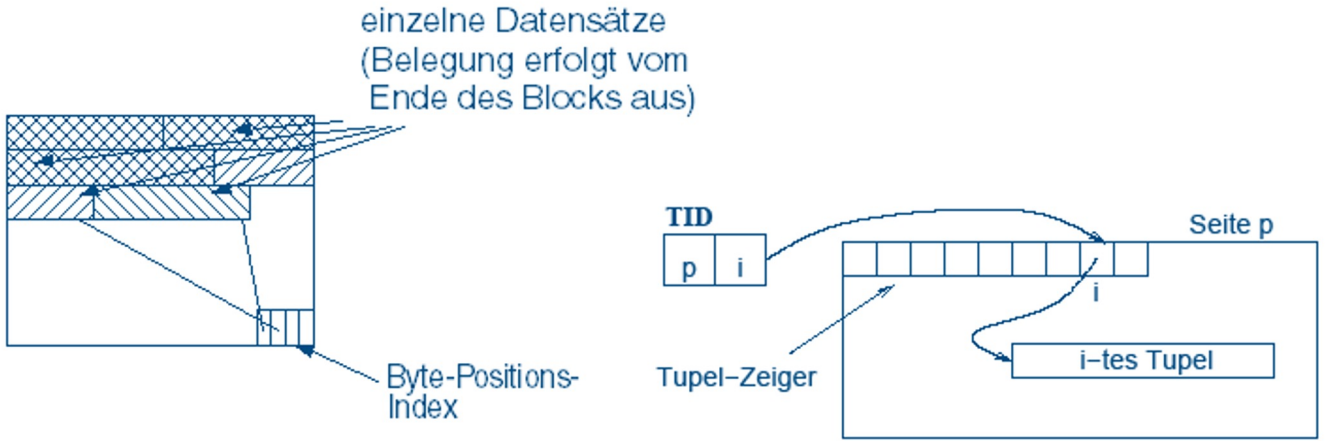
- Schneller, möglichst direkter Satzzugriff
- Hinreichend stabil gegen geringfügige Verschiebungen (Verschiebungen innerhalb einer Seite ohne Auswirkungen)
- Seltene oder keine Reorganisationen

TID-Konzept

Satzadressierung über Indirektion innerhalb einer Seite

- Array mit Byte-Positionen der Sätze in dieser Seite
- Adresse ist das Paar bestehend aus Seitennummer und Index in diesem Array (“TID = Tuple IDentifier”)
- Für den Zugriff auf einen Satz wird nur ein Seitenzugriff benötigt

Struktur einer Seite



TID-Konzept (2)

Dresden Database Research Group

Löschen eines Satzes

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

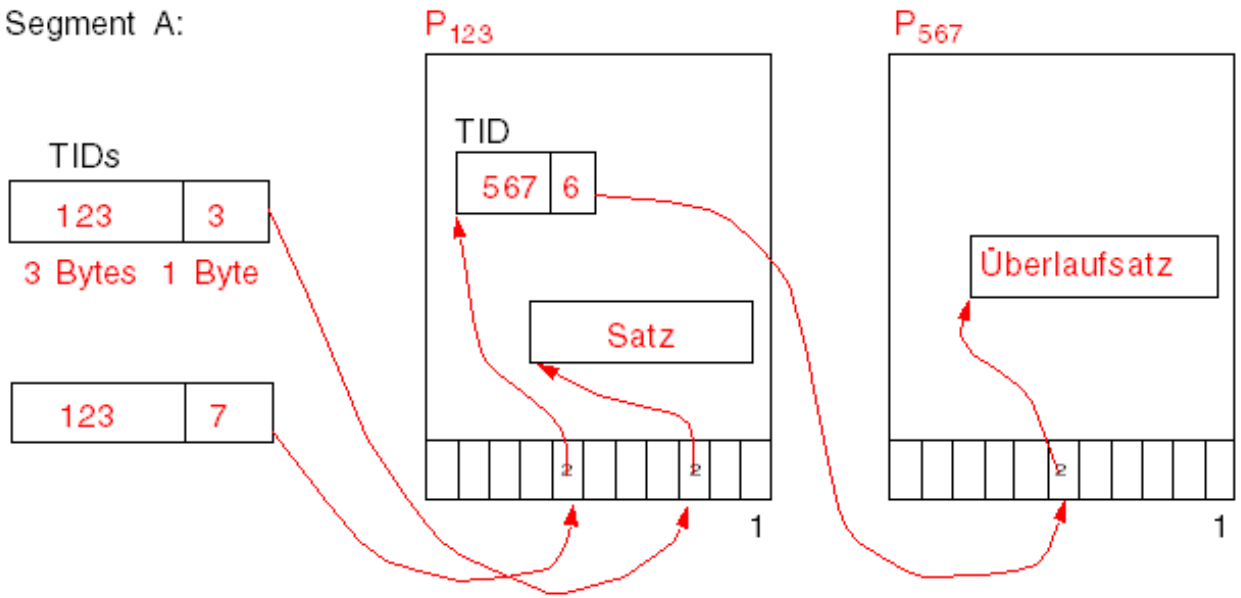
Aktualisierungsoperation auf einen Datensatz

- Es kann sich die Länge eines Datensatzes verändern !!!
- Datensatz schrumpft oder wird größer (**ohne Überlauf**): alle Sätze werden innerhalb der Seite verschoben und der Byte-Positionsindex wird angepasst
- Datensatz wird größer und der freie Platz in der Seite reicht nicht mehr für die Speicherung des jetzt größeren Datensatzes (**Überlauf**)
  - Verschiebung des Datensatzes in eine andere Seite!

Überlaufbehandlung TID-Konzept

Dresden Database Research Group

Verlagerung eines Satzes



Überlaufbehandlung TID-Konzept (2)

Dresden Database Research Group

Vorgehen

- In der alten Seite verbleibt an der Stelle des Originalsatzes eine neue Satzadresse, die auf die neue Seite verweist
  - In diesem (seltenen) Fall müssen also zwei Seiten gelesen werden.
  - Wird der Satz ein weiteres Mal verlagert, so wird die Satzadresse in der ersten Seite 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 platziert werden.

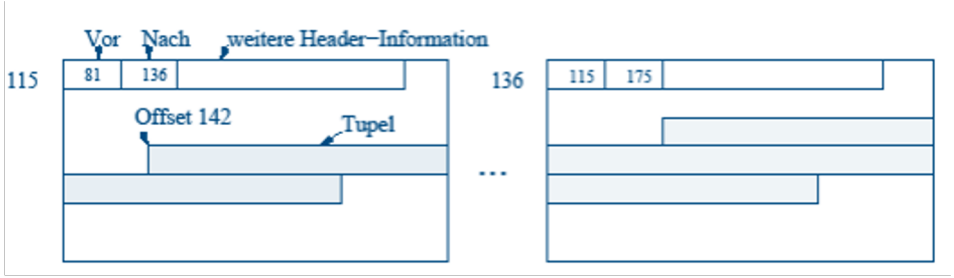
Vorteile

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

Seitenorganisation

Verkettung

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

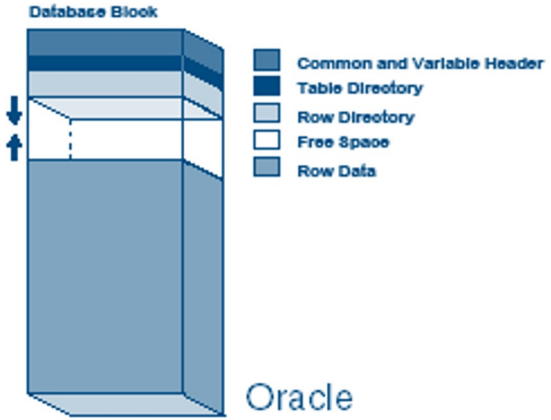


Seiten-Header

- 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

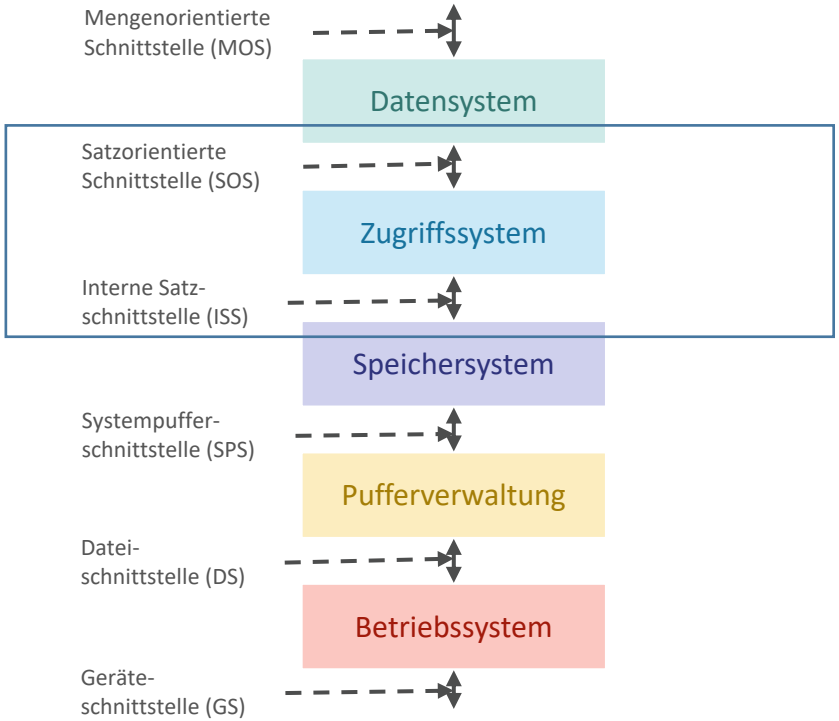


TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Research Group

Zugriffssystem - Indexstrukturen

Einordnung in Schichtenarchitektur



Motivation von Zugriffspfaden

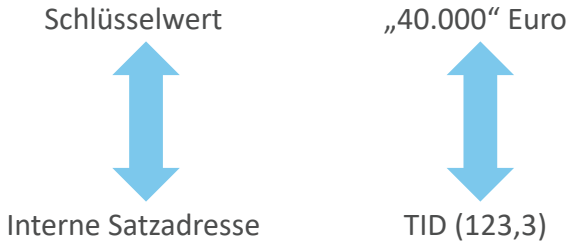
Dresden Database Research Group

Arten von Zugriffen

- Sequentieller Zugriff auf alle Sätze eines Satztyps (Scan)
- Sequentieller Zugriff in Sortierreihenfolge eines Attributes
- Direkter Zugriff über den Primärschlüssel (z.B.: Kennzeichen = “DD-EK 2332”)
- Direkter Zugriff über Sekundärschlüssel (z.B. Farbe = “silber” und Automarke = “VW”)
- Direkter Zugriff über zusammengesetzte Schlüssel und komplexe Suchausdrücke (Wertintervalle, ...)
- Navigierender Zugriff von einem Satz zu einer dazugehörigen Satzmenge desselben oder eines anderen Satztyps

Anforderungen an Zugriffspfade

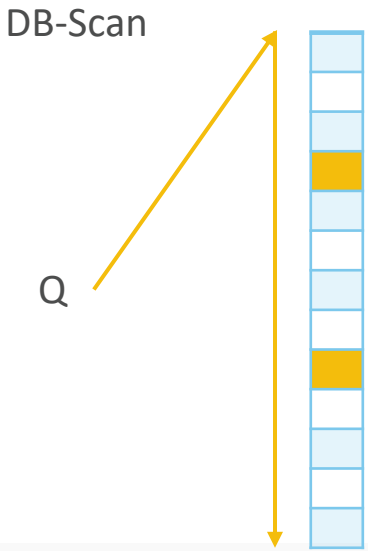
- Effizientes (direktes) Auffinden von Datensätzen bzgl. inhaltlichen Kriterien
- Vermeiden von sequentiellem Durchsuchen aller Datensätze
- Erleichterung von Zugriffskontrollen durch vorgegebene Zugriffspfade (constraints)
- Erhaltung topologischer Beziehungen

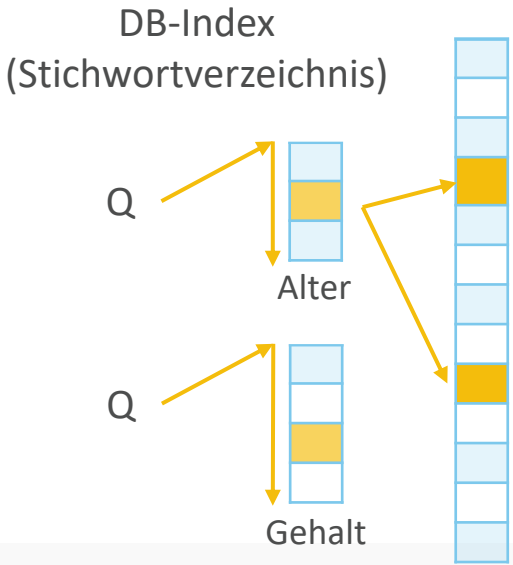


Nutzung eines Index

Idee

- Einführung eines Zwischenschrittes
- Pers(PID, Name, Alter, Gehalt, …)





DB-Scan versus Index-Nutzung

Dresden Database Research Group

DB-Scan

- Alle Seiten müssen gelesen und alle Sätzen in den eingelesenen Seiten müssen hinsichtlich dem Suchkriterium untersucht werden
- Wird von allen DBMS unterstützt
- Ist ausreichend / effizient bei:
  - Kleinen Satztypen (z. B. $\leq$ 5 Seiten)
  - Anfragen mit großen Treffermengen (z. B. > 1 %)
- DBMS kann Prefetching zur Scan-Optimierung nutzen

Index

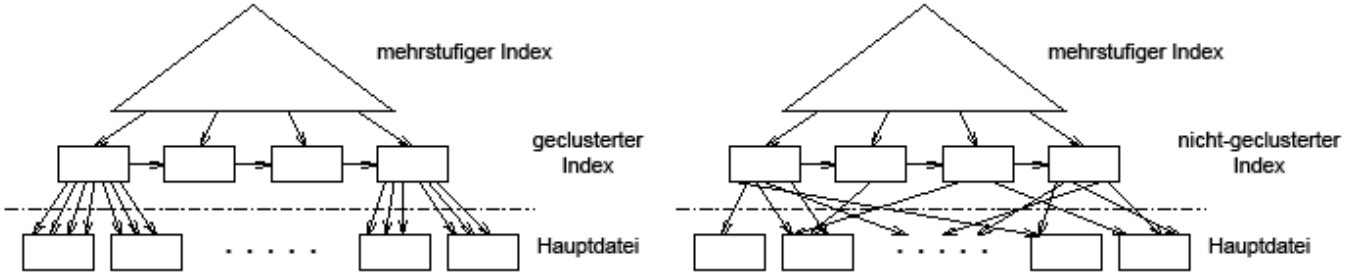
- Zwei Klassen von Indexstrukturen
  1. Schlüsselwerte werden transformiert um die betreffenden Seiten/Blöcke zu ermitteln
  2. Schlüsselwerte werden redundant in einer eigenen Struktur gehalten und mit dem Suchkriterium verglichen
- ... wenn kein geeigneter Zugriffspfad vorhanden (oder dessen Nutzung nicht ökonomischer) ist, müssen alle Zugriffsarten mittels eines Scans durchgeführt werden

Primär- versus Sekundärindex

Dresden Database Research Group

Klassifikation

- (Primär-)Index: bestimmt Dateiorganisationsform
  - Unsortierte Speicherung von Tupeln: Heap-Organisation
  - Sortierte Speicherung von internen Tupeln: sequentielle Organisation
  - Gestreute Speicherung von internen Tupeln: Hash-Organisation
  - Speicherung in mehrdimensionalen Räumen: mehrdimensionale Dateiorganisationsformen
  - Normalfall: Primärindex über Primärschlüssel / geclusterter Index
- Sekundärindex
  - Redundante Zugriffsmöglichkeit, zusätzlicher Zugriffspfad



Index-Beschreibung in SQL

Dresden Database Research Group

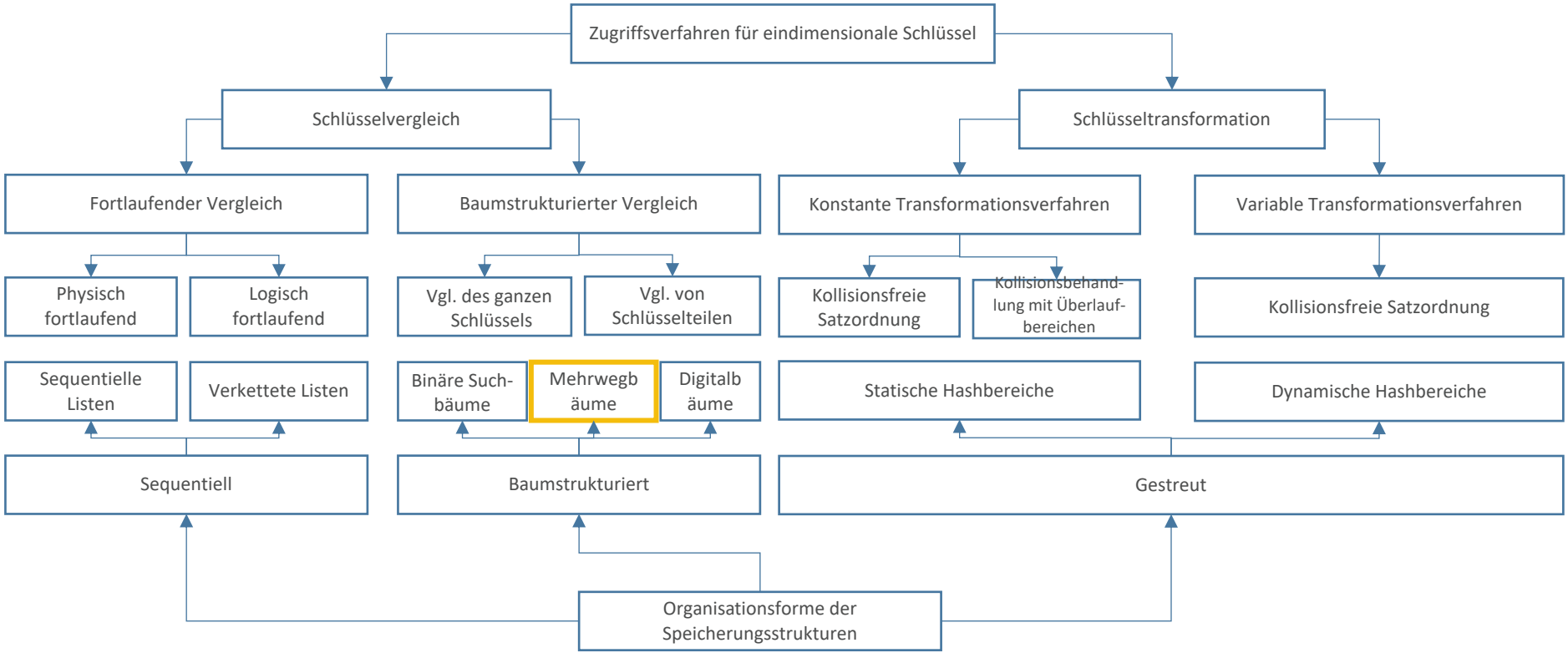
Formulierung in SQL

- `CREATE UNIQUE INDEX pnr_idx ON pers (pnr) ALLOW REVERSE SCANS`
  - Ermöglicht bidirektionale Index-Scans (Standard)
- `CREATE UNIQUE INDEX pnr_idx ON pers (pnr) INCLUDE (pname)`
  - Zusätzliche Spalten zur Vermeidung des Zugriffs auf Relation
- `CREATE INDEX pgehalt_idx ON pers (gehalt)`
- `CREATE INDEX alt_geh_idx ON pers (alter, gehalt)`
- wichtig: verschieden zu
  `CREATE INDEX geh_alt_idx ON pers (gehalt, alter)`

Berechnete Indexe

- Definition von Indexstrukturen auf Funktionen (z.B. Oracle)
- Benutzung von Anfragen, die auf „äquivalenten" algebraischen Ausdruck zurückgreifen
- `CREATE INDEX idx ON table_x (a + b * (c - 1), a, b);`
  wird benutzt, um folgende Anfrage zu unterstützen
  `SELECT a FROM table_1 WHERE a + b * (c - 1) < 100;`

Klassifikation der Verfahren



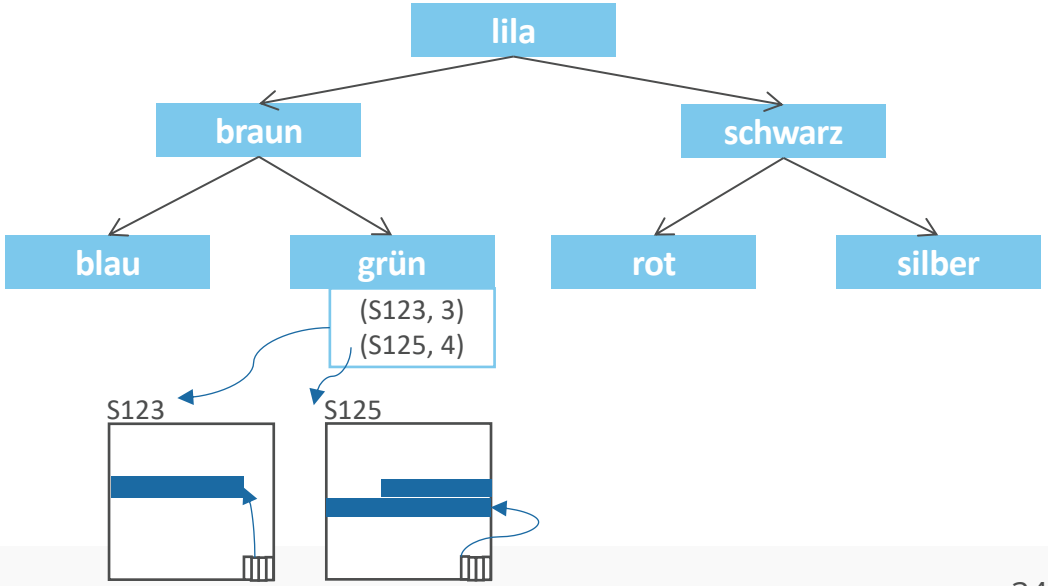
Liste versus Binärbaum

Dresden Database Research Group

Verwaltung der Indexeinträge

- Variante 1: Liste von Einträge
- Variante 2: Organisation als Binärbaum / Binärer Suchbaum
  - Baumstruktur mit einem linken und rechten Kind
  - Ausgeglichener balancierter Suchbaum

blau	
braun	
grün	(S123, 3) (S125, 4)
lila	
rot	
schwarz	
silber	



Erweiterung des binären Suchbaumes

Dresden Database Research Group

Mehrwegbaum

- Baumstruktur mit mehreren Kindern
- Idee: Die maximale Größe eines Knotens entspricht exakt der Speicherkapazität einer Seite

B-Baum

- Variante eines Mehrwegbaumes zur Abbildung von Schlüsselwerten auf interne Satzadressen
- Entworfen für den Einsatz in Datenbanksystemen (Bayer, McCreight, 1972)

Funktion

- Dynamische Reorganisation durch Splitten und Mischen von Seiten
- Direkter Schlüsselzugriff
- Sortierter sequentieller Zugriff (insbes. B*-Baum)

B-Bäume

"It could be said that the world's information is at our fingertips because of B-trees"

Goetz Graefe Microsoft, HP Fellow, now Google ACM Software System Award

B-Baum Eigenschaften

Dresden Database Research Group

Definition

Ein B-Baum vom Typ (k, h) ist ein Baum mit folgenden drei Eigenschaften

- Jeder Pfad von der Wurzel zum Blatt hat die gleiche Länge h
- Jeder Knoten mit der Ausnahme der Wurzel als Blatt hat mindestens k und höchstens 2k Einträge (Schlüssel)
- Jeder Knoten (außer Wurzel und Blätter) hat mindestens k + 1 Nachfolger und höchstens 2k + 1 Nachfolger
- Die Wurzel ist ein Blatt oder hat mindestens 2 Nachfolger

Seitenformat

- (K_i, D_i, P_i) = Eintrag
  - K_i = Schlüssel
  - D_i = Daten des Satzes oder Verweis auf den Satz (materialisiert oder referenziert)
  - P_i = Zeiger zu einer Nachfolgerseite

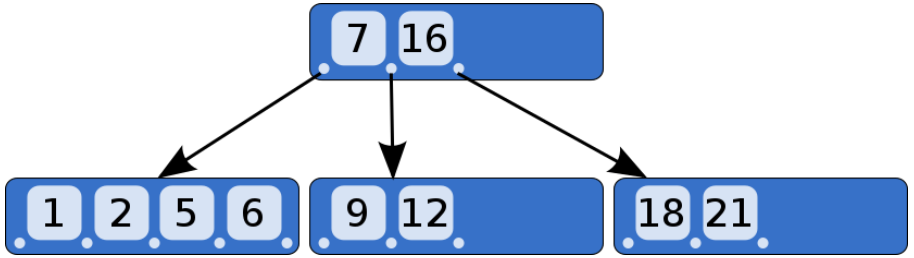
B-Baum Parameter

Bedeutung der Zeiger P_i (i = 0, 1, ... p)

- P_0 weist auf einen Teilbaum mit Schlüsseln kleiner als K_1
- P_i (i = 1, 2, ..., l - 1) weist auf einen Teilbaum, dessen Schlüssel zwischen K_i und K_{i+1} liegen
- P_p weist auf einen Teilbaum mit Schlüsseln größer als K_p
- In den Blattknoten sind die Zeiger nicht definiert

Parameter h (Höhe des Baumes)

- Ergibt sich aus der Anzahl der gespeicherten Datenelemente und der Einfügereihenfolge



Berechnung der Höhe

Dresden Database Research Group

Minimale und maximale Höhe

- B-Baum der Ordnung k mit n Schlüsseln
- Level 2 hat ≥ 2 Knoten
- Level 3 hat ≥ 2(k+1) Knoten
- Level 4 hat ≥ $2(k+1)^2$ Knoten
- Level h+1 hat n+1 ≥ $2(k+1)^{h-1}$ (äußere) Knoten

- und somit:

$$\lceil \log_{2k+1}(n+1) \rceil \leq h \leq \left\lceil \log_{k+1}\left(\frac{n+1}{2}\right) + 1 \right\rceil$$

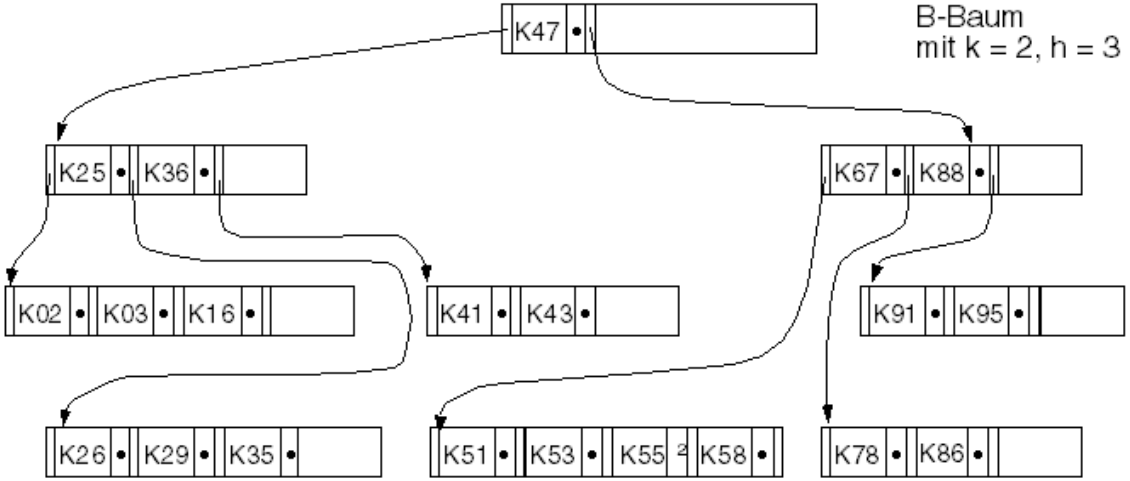
Beobachtung

- Jeder Knoten (außer der Wurzel) ist mindestens mit der Hälfte der möglichen Schlüssel gefüllt
  → **Speicherplatzausnutzung ≥ 50 %**!

Beispiel eines B-Baumes

Dresden Database Research Group

B-Baumstruktur als Zugriffspfad für den Primärschlüssel ANR



Operationen

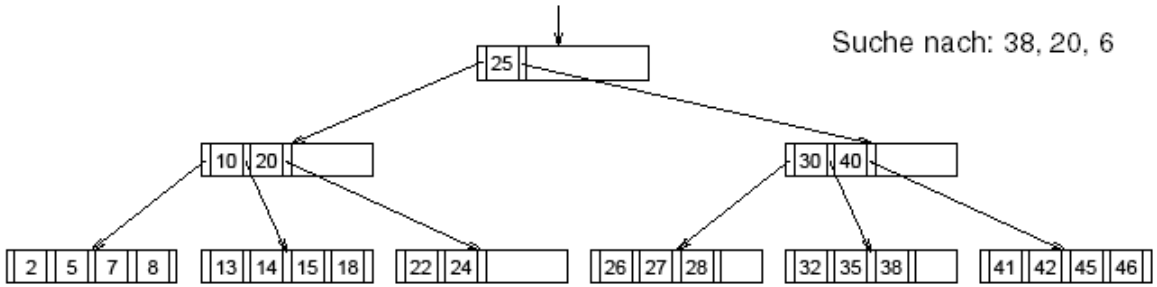
- Suchen eines Datensatzes mit vorgegebenem Schlüsselwert
- Einfügen und Löschen eines Datensatzes

Suche im B-Baum

Dresden Database Research Group

Suche

- Beginnend mit dem Wurzelknoten, wird ein Knoten jeweils von links nach rechts durchsucht
  - 1) Stimmt K_i mit dem gesuchten Schlüsselwert überein, ist der Satz gefunden. (Weitere Sätze mit gleichem Schlüsselwert befinden sich ggf. in dem Teilbaum, auf den P_{i-1} zeigt.)
  - 2) Ist K_i größer als der gesuchte Wert, wird die Suche in der Wurzel des von P_{i-1} identifizierten Teilbaums fortgesetzt.
  - 3) Ist K_i kleiner als der gesuchte Wert, wird der Vergleich mit K_{i+1} wiederholt.
  - 4) Ist auch K_{2k} noch kleiner als der gesuchte Wert, wird die Suche im Teilbaum von P_{2k} fortgesetzt.
- Ist der weitere Abstieg in einen Teilbaum (2. oder 4.) nicht möglich (Blattknoten):
  - Suche abbrechen, kein Satz mit gewünschtem Schlüsselwert vorhanden.



Einfügen im B-Baum

Dresden Database Research Group

Regel

- Eingefügt wird nur in Blattknoten!

Vorgehen

- Zunächst Abstieg durch den Baum wie bei Suche (S sei der Schlüssel des einzufügenden Satzes)
  - $S \leq K_i$: folge P_{i-1}
  - $S > K_i$: prüfe K_{i+1}
  - $S > K_{2k}$: folge P_{2k}
- Im so gefundenen Blattknoten:
  - Satz entsprechend der Sortierreihenfolge einfügen
  - Sonderfall: Blattknoten ist schon voll (enthält 2k Sätze)

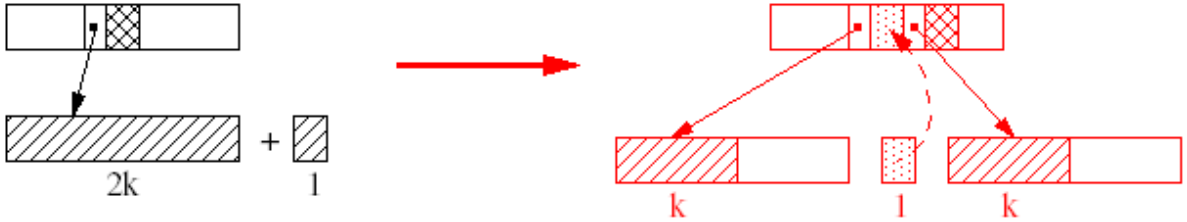
  → Splitt des Blattknotens

Splitt beim Einfügen im B-Baum

Dresden Database Research Group

Vorgehen beim Splitt

- Einen neuen Blattknoten erzeugen
- Die 2k+1 Sätze (in Sortierordnung!) halbe-halbe zwischen altem und neuem Blattknoten aufteilen
  - Die ersten k Sätze in die erste (die linke) Seite
  - Die letzten k Sätze in die zweite (die rechte) Seite
- Den mittleren (k+1-ten) Satz als neuen “Diskriminator” (als Verzweigungsinformation bei der Suche) in den eine Stufe höheren Knoten einfügen, der auf den Blattknoten verweist



Splitt beim Einfügen im B-Baum (2)

Dresden Database Research Group

Zwei mögliche Situationen nach einem Splitt

- Der übergeordnete Knoten ist voll → Splitt auf dieser Ebene wiederholen
- Ausreichend Platz → Fertig

Weiterer Sonderfall

- Splitt des Wurzelknotens
  → Erzeugung von zwei neuen Knoten
  → Neue Wurzel mit zwei Nachfolgeknoten
- Höhe des Baums wächst um 1
  (Man sagt bildlich: Der Baum “reißt von unten nach oben auf”)

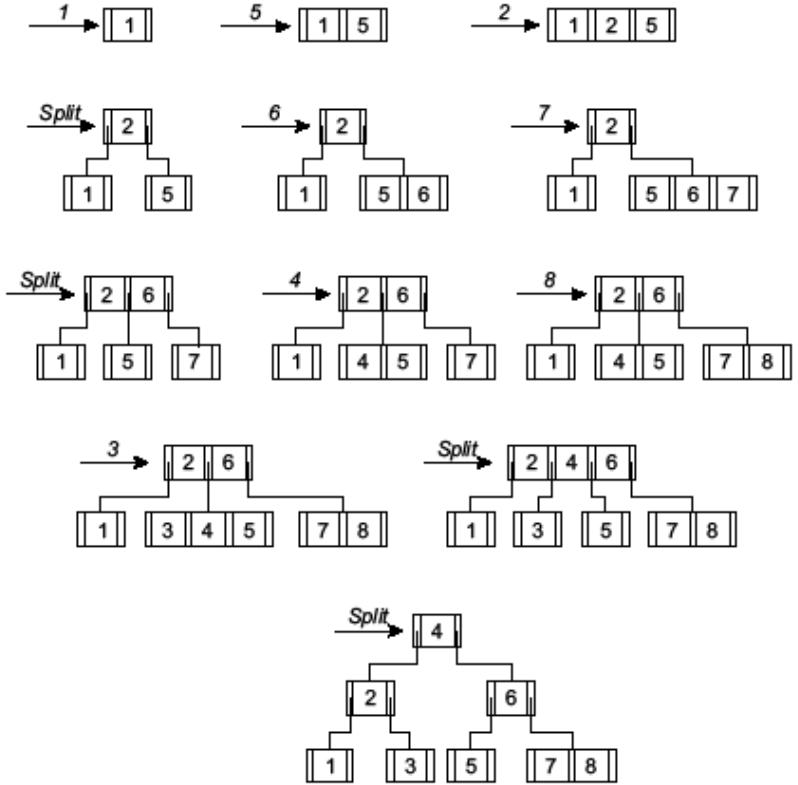
Dynamische Reorganisation

- Kein Entladen und Laden erforderlich
- Baum immer balanciert

Einfügen im B-Baum: Beispiel

Einfügen

- k=1



Einfügen und Löschen im B-Baum

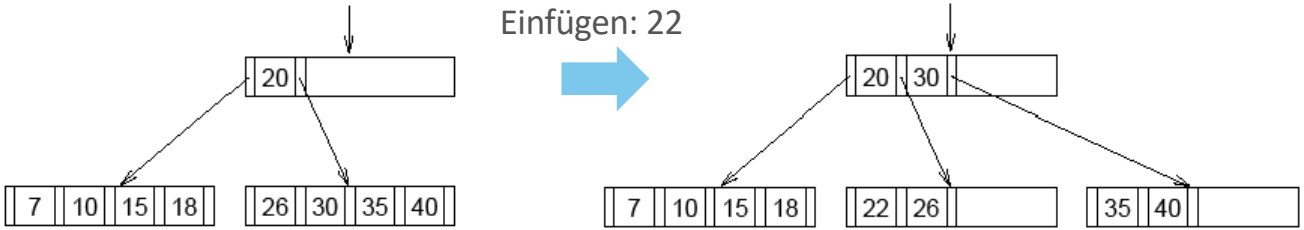
Dresden Database Research Group

Problem

- Einfügen kann **Überlauf** zur Folge haben
- Löschen kann **Überlauf** und **Unterlauf** zur Folge haben

Beispiel, k=2

- Einfügen des Schlüssels 22 → Überlauf → Splitten



- Löschen des Schlüssels 22 → **Unterlauf**, Zugriff auf alle 4 Knoten erforderlich, Wiederherstellung des Ursprungszustands

Löschalgorithmus

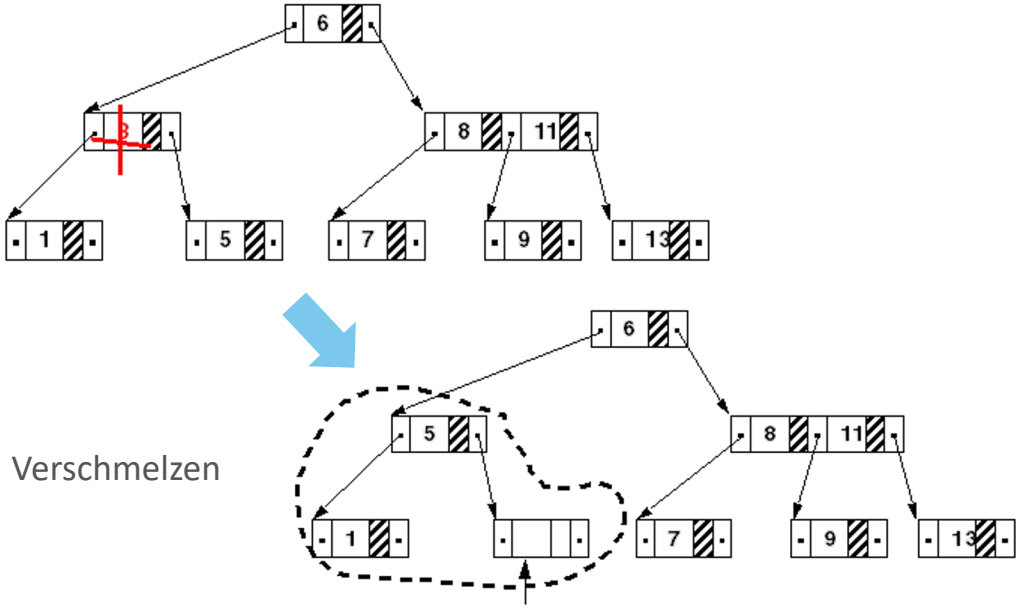
- Suche den Knoten der den zu löschenden Schlüssel K enthält
- Falls K in einem Blattknoten, löschen den Schlüssel und behandele den potentiell resultierenden Unterlauf durch ein Verschmelzen mit dem Geschwisterknoten
- Falls K in einem inneren Knoten, hole einen neuen Diskriminator aus einem der Nachfolger
  - Analysiere welcher Nachfolgeknoten von K mehr Elemente hat: der Rechte oder der Linke, falls beide gleichviele haben entscheide für einen
  - Ersetze den zu löschenden Schlüssel K mit dem linken K’ bzw. dem rechten Nachfolgerschlüssel K”
  - Lösche K’ oder K” aus dem entsprechenden Nachfolgerknoten
  - Fahre rekursiv fort

Löschen im B-Baum

Dresden Database Research Group

Beispiel

- Ordnung k = 1, n=2k
- Lösche Schlüssel 3



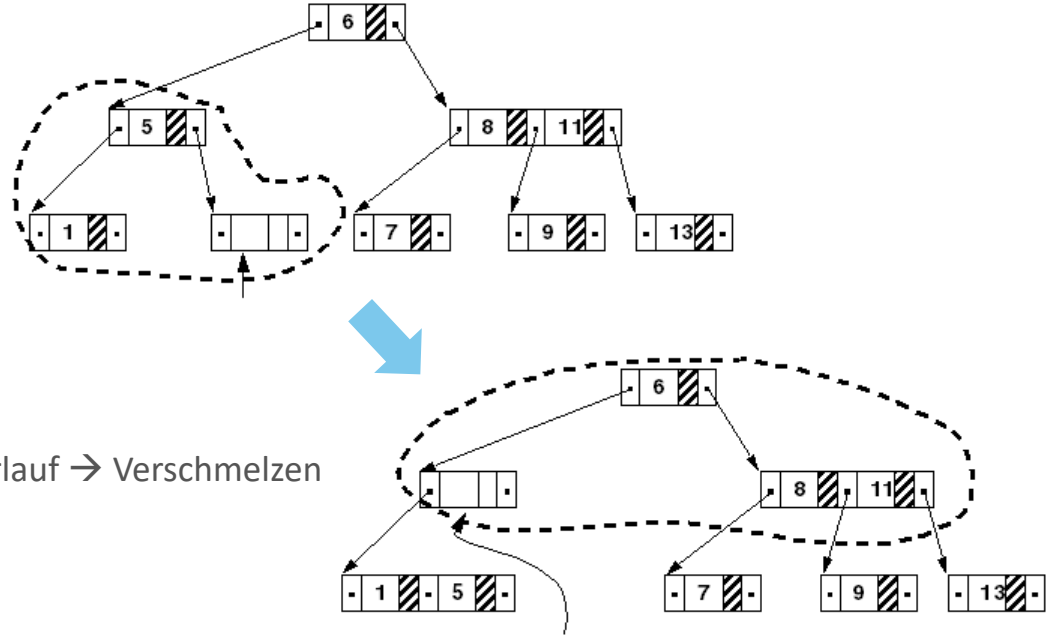
Unterlauf → Verschmelzen

Löschen im B-Baum (2)

Dresden Database Research Group

Beispiel (Fortsetzung)

- Ordnung k = 1, n=2k
- Lösche Schlüssel 3



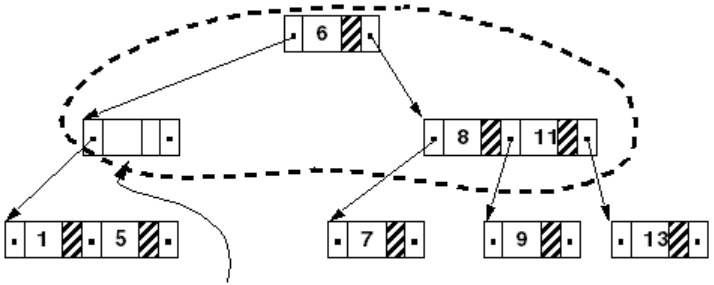
Löschen im B-Baum (3)

Dresden Database Research Group

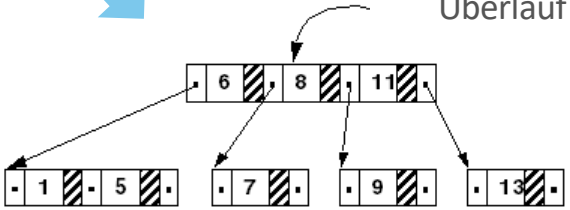
Beispiel (Fortsetzung)

- Ordnung k = 1, n=2k
- Lösche Schlüssel 3

Unterlauf → Verschmelzen



Überlauf → Splitten

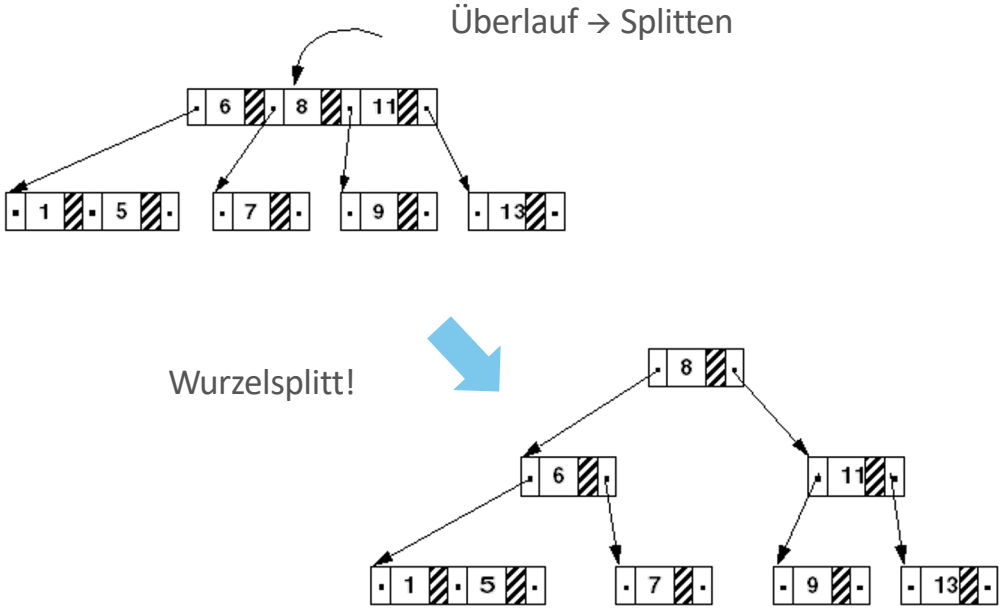


Löschen im B-Baum (4)

Dresden Database Research Group

Beispiel (Fortsetzung)

- Ordnung k = 1, n=2k
- Lösche Schlüssel 3



B*-Baum

B*-Baum

Dresden Database Research Group

Eigenschaften und Unterschiede zum B-Baum

- Alle Sätze (bzw. Schlüsselwerte mit TID's) werden in den Blattknoten abgelegt.
- Innere Knoten enthalten nur Verzweigungsinformation (also u.U. auch Schlüsselwerte, die in keinem Satz vorkommen), aber keine Daten
- Aufbau von B*-Baum-Knoten:

Innerer Knoten

P_0	R_1	P_1	R_2	P_2	...	R_p	P_p	freier Platz

R_i = Referenzschlüssel, $k \leq p \leq 2k$

Blatt-knoten

M	S_1	D_1	S_2	D_2	...	S_j	D_j	freier Platz	N

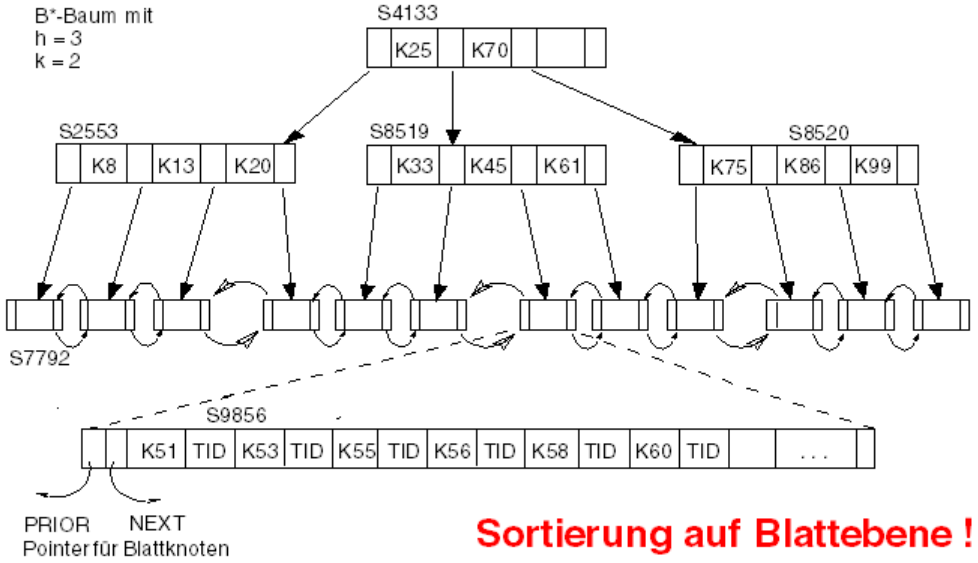
M = PRIOR-Zeiger, N = NEXT-Zeiger, $k^* \leq p \leq 2k^*$

B*-Baum für Primärschlüssel

Dresden Database Research Group

Beispiel

- ANR ist Primärschlüssel in der Relation ABT(ANR, ORT, MNR)

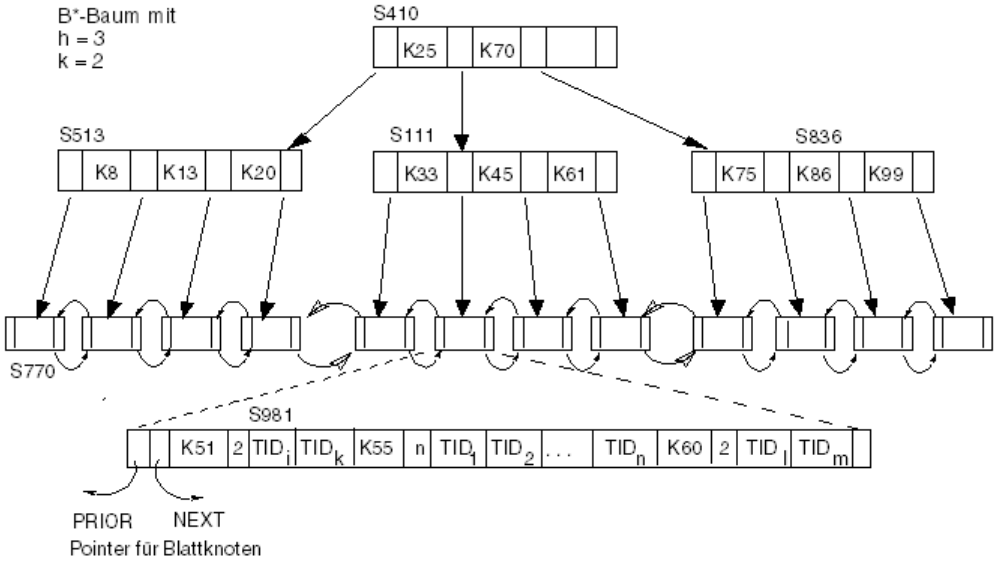


B*-Baum für Sekundärschlüssel

Dresden Database Research Group

Beispiel

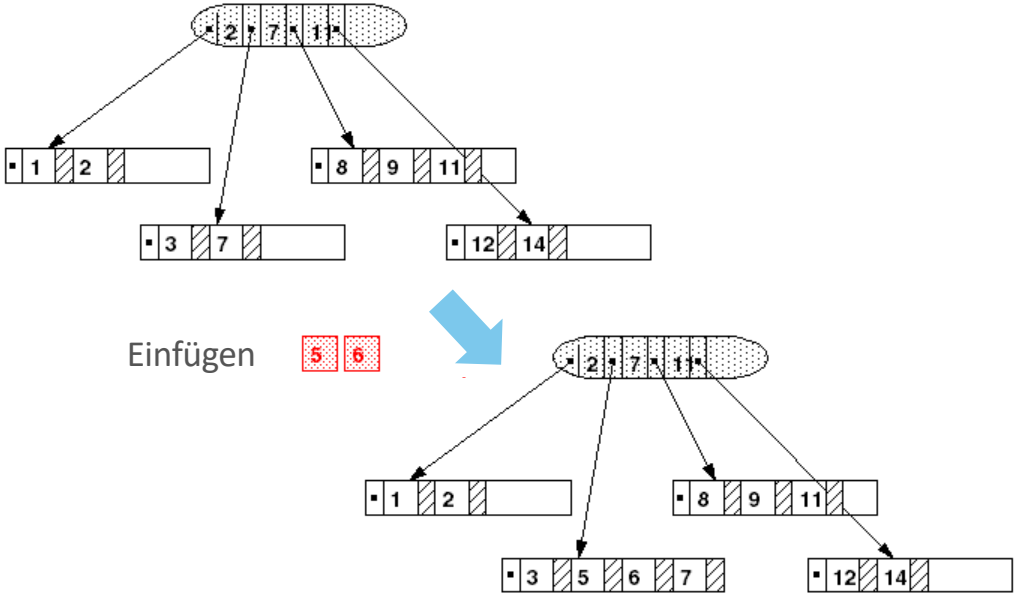
- ANR ist Sekundärschlüssel in der Relation PERS(PNR, NAME, ALTER, ANR)



Einfügeoperation im B*-Baum

Dresden Database Research Group

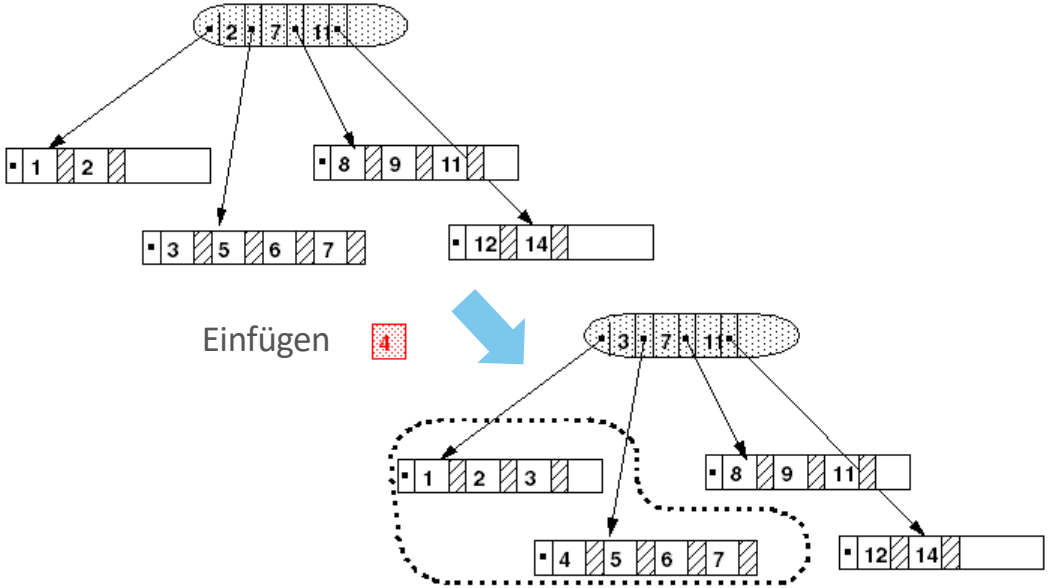
… am Beispiel



Balancierung im B*-Baum

Dresden Database Research Group

...am Beispiel



Löschen von Sätzen aus B*-Baum

Dresden Database Research Group

1. Suche den zu löschenden Eintrag im Baum

2. Entsteht durch das Löschen ein Unterlauf? (#Einträge < k?)

- Nein
  - Entferne den Satz aus dem Blatt (Aktualisierung des Diskriminators im Vaterknoten ist nicht erforderlich!)
- Ja
  - Mische das Blatt mit einem Nachbarknoten:
  - Ist die Summe der Einträge in beiden Knoten größer als 2k?
    - Nein
      - Fasse beide Blätter zu einem Blatt zusammen
      - falls dabei ein Unterlauf im Vaterknoten entsteht: mische die inneren Knoten analog
    - Ja
      - Teile die Sätze neu auf beide Knoten auf, so dass ein Knoten jeweils die Hälfte der Sätze aufnimmt
      - Der Diskriminator im Vaterknoten ist entsprechend zu aktualisieren

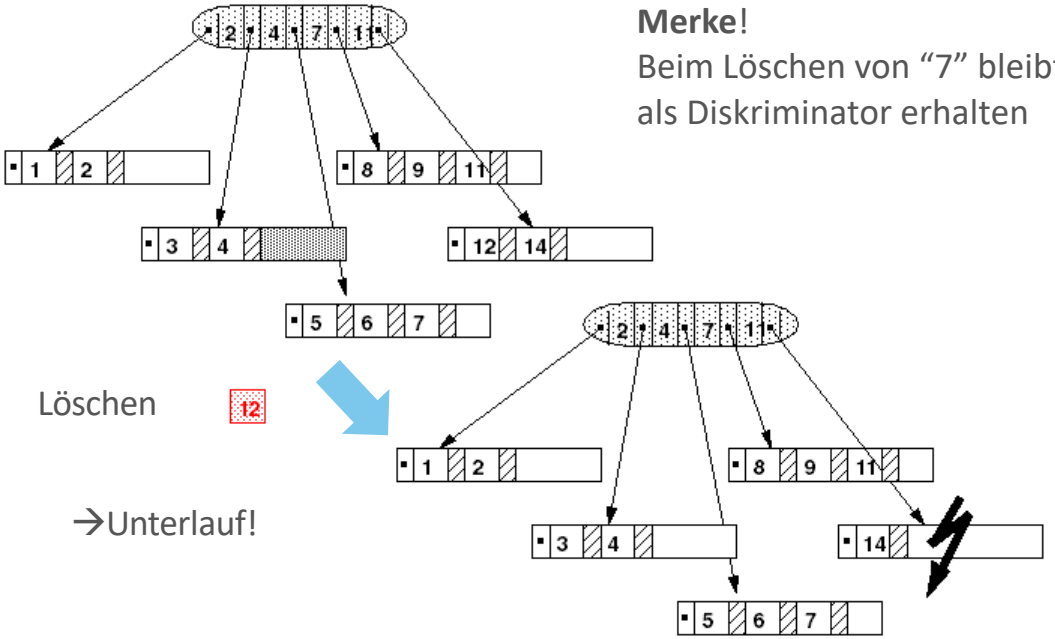
Anmerkung

- Vielzahl von Varianten bzgl. Aufteilung nach UDI-Operationen (UDI = Update/Delete/Insert)

Löschen im B*-Baum

Dresden Database Research Group

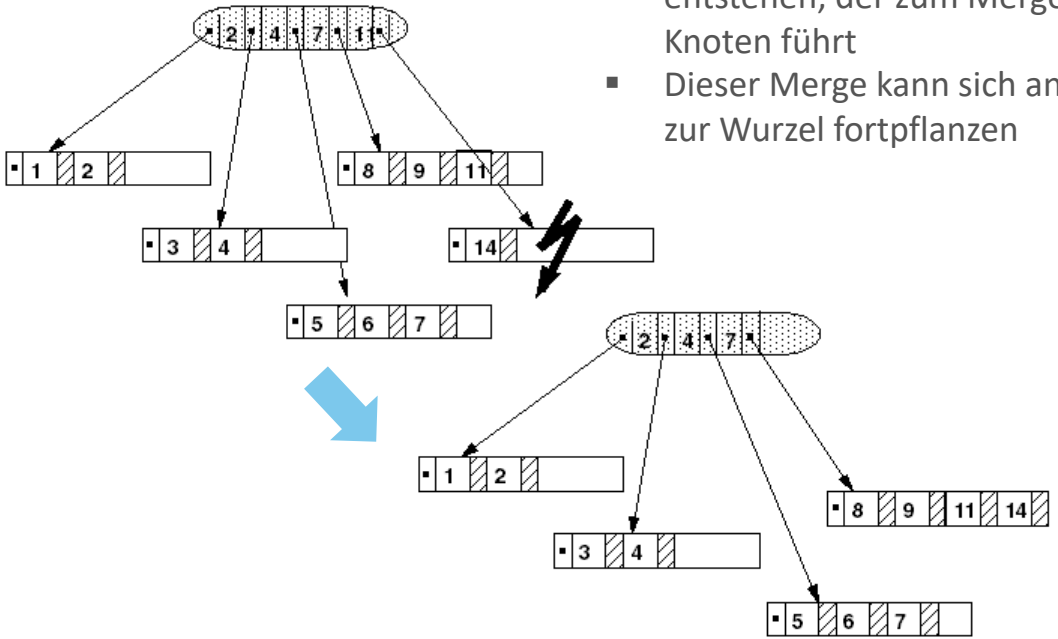
… am Beispiel



Merke!
Beim Löschen von “7” bleibt “7” als Diskriminator erhalten

Löschen im B*-Baum (2)

… am Beispiel



- Durch den Merge zweier Blätter kann auch in den inneren Knoten ein Unterlauf entstehen, der zum Merge von inneren Knoten führt
- Dieser Merge kann sich analog zum Split bis zur Wurzel fortpflanzen

Vergleich B- und B*-Baum

Dresden Database Research Group

B-Baum

- Keine Redundanz
- Lesen aller Sätze sortiert nach Schlüsselwert nur mit Verwaltung eines Stack der max. Tiefe = Baumhöhe h

- Bei Einbettung der Datensätze geringe Verzweigungszahl („Grad“ oder „fan-out“), daher größere Höhe
- Einige wenige Sätze (die in der Wurzel) werden mit einem Zugriff erreicht

B*-Baum

- Schlüsselwerte teilweise redundant gespeichert
- Kette der Blattknoten liefert alle Sätze nach Schlüsselwert sortiert

- Hohe Verzweigung in den inneren Knoten, daher geringe Höhe

TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Research Group

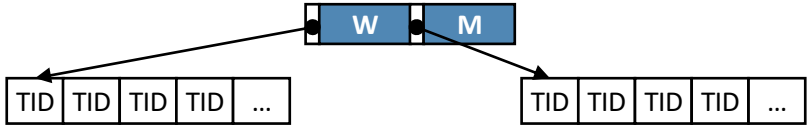
Indizierung von Spalten mit geringer Kardinalität

Spalten mit geringer Kardinalität

Dresden Database Research Group

Problem

- Beispiel: B-Baum auf Attribut Geschlecht der Kundentabelle mit 1 Mio. Tupeln resultiert in zwei Liste mit jeweils ungefähr 500.000 Einträgen



- Anfrage nach alle weiblichen Kunden benötigt 500.000 wahlfreie Seitenzugriffe (Sekundärindex!)
  → Tabellen-Scan wäre in diesem Fall viel schneller

Fazit

- B-Bäume (aber auch Hashing) sind nur sinnvoll für Prädikate mit geringer Selektivität (Verhältnis zwischen Eingangs- und Ausgangskardinalität)
- Daumenregel: Grenztrefferrate bei ungefähr 5%
- Höhere Raten rechtfertigen den Aufwand eines Indexes nicht

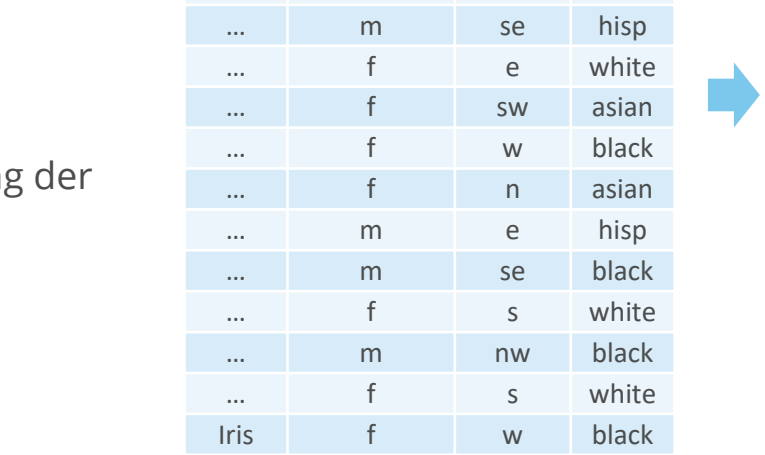
Bitmap-Index

Dresden Database Research Group

Idee

- Erzeugen einer Bitliste für jeden Attributwert
- Jedes Tupel der Tabelle wird einem Bit in der Bitliste zugeordnet
- Bitwerte
  - 1 → Attribut hat diesen Wert
  - 0 → Attribut hat diesen Wert nicht
- Notwendige Voraussetzung:
  Sequentielle Ordnung / Nummerierung der Tupel (TIDs)

Name	Geschlecht	Region	Race
Carol	f	n	white
…	m	e	black
…	f	e	asian
…	f	ne	white
…	m	se	hisp
…	f	e	white
…	f	sw	asian
…	f	w	black
…	f	n	asian
…	m	e	hisp
…	m	se	black
…	f	s	white
…	m	nw	black
…	f	s	white
Iris	f	w	black



Anfragen mittels Bitmap-Index

Dresden Database Research Group

Wichtigster Vorteil eines Bitmap-Index

- Joins sehr einfach und effizient realisierbar
- Nur die Daten lesen die für die Prädikate relevant sind

Beispiel: I/O Costs Estimation

- $\sigma_{Geschlecht='w' \wedge Region='n' \wedge Race='Asian'} R$ („asiatische Frauen der Region Nord")
- Selektivitäten: $\frac{1}{2} \cdot \frac{1}{8} \cdot \frac{1}{4} = \frac{1}{64}$
- N=10.000 Tupel, mit jeweils 400 Bytes (~ 10 Tupel pro Seite mit 4kB)
- Tabellen-Scan: 1.000 Seiten
- Bitmap-Zugriff: 10.000/64 → 156 Seiten (Worst case: jedes Tupel in einer anderen Seite) sowie 1 Seite für die Bitmaps

W		N		A		*
1		0		0		0
0		1		0		0
1		1		1		1
1		0		0		0
0		0		0		0
1		1		0		0
1		0		1		0
1	AND	0	AND	0	=	0
1		0		1		0
0		1		0		0
0		0		0		0
1		0		0		0
0		0		0		0
1		0		0		0
1		0		0		0

Zusammenfassung

Dresden Database Research Group

Architektur von Datenbanksystemen