FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Datenbanksysteme II

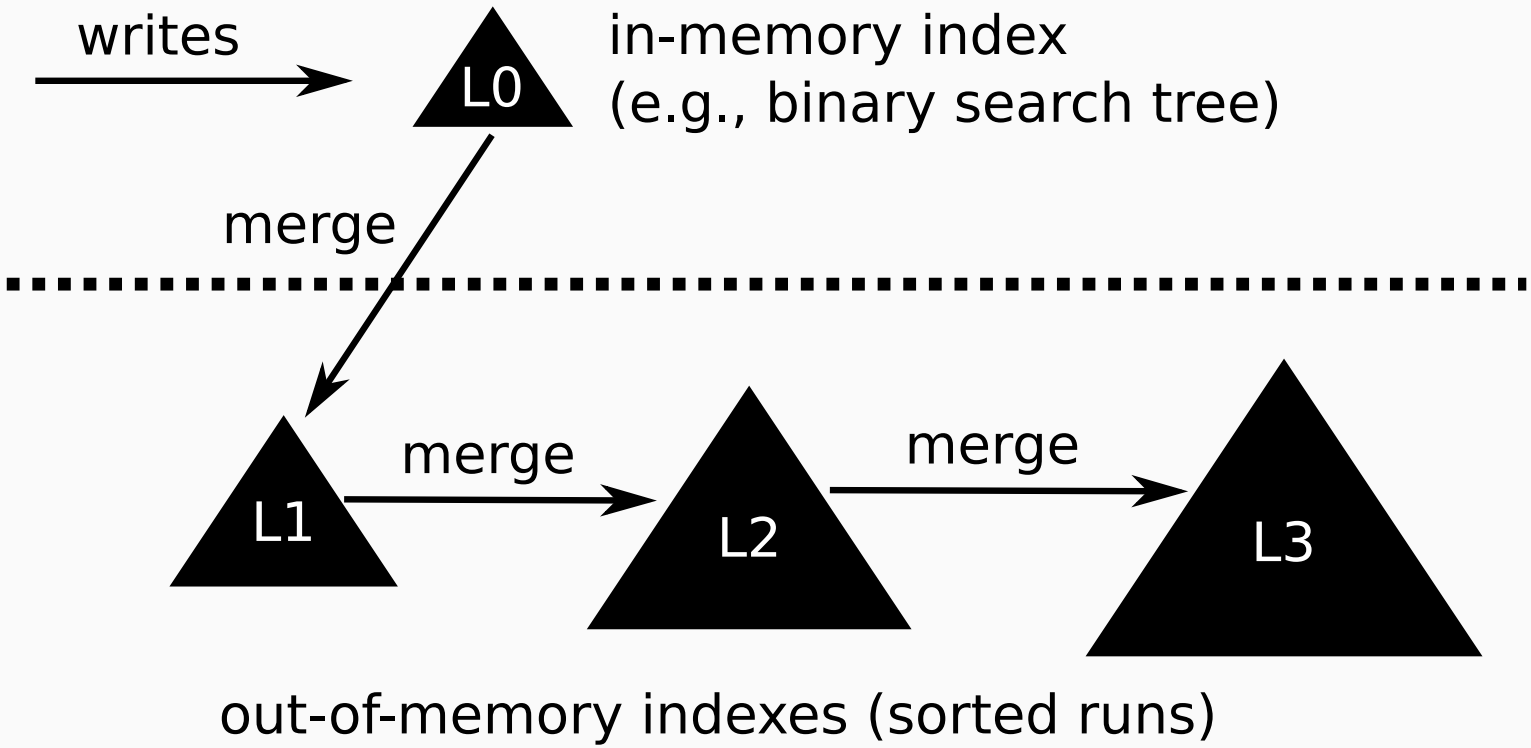
Prof. Dr. Viktor Leis

Professur für Datenbanken und Informationssysteme

Spezielle Datenstrukturen

Log-Structured Merge-Tree (LSM)

- neue Werte werden in in-memory Ebene eingefügt
- Löschen wird zu “Anti-Materie”
- Modifikation wird zu Löschen und Einfügen
- Leseoperation müssen alle Ebenen durchsuchen



LSM Eigenschaften

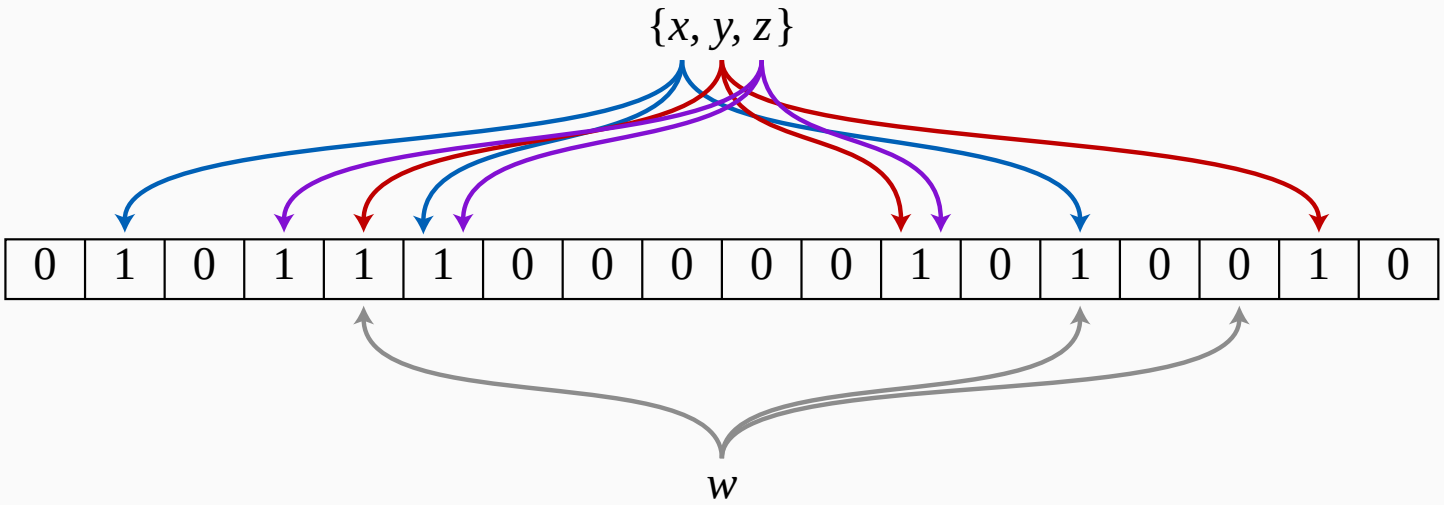
- Einfügen/Löschen/Modifikation ist zunächst sehr effizient
- Speicherung auf Platte ist kompakt
- Merge ist teuer
- Suche ist langsam
- Suche kann durch Bloom filter beschleunigt werden
- (kurze) Bereichssuche ist teuer

Bloom Filter

- platzsparende Repräsentation einer Menge
- probabilistisch
- einseitiger Fehler (“false positive”)
- viele Anwendungen (z.B. Verteile Joins, LSM Bäume, Caching)

Bloom Filter

- Anzahl der Elemente: n
- Größe in bits: m
- Anzahl Hashfunktionen: k
- z.B. $n = 3$, $m = 18$, $k = 3$:



Konfiguration

- Annahme: gute Hashfunktion (Hashwerte sind gleichverteilt)
- für eine gewünschte Fehlerrate p (“false positive rate”) ergibt sich:
  - $m \approx -1.44 log_2 p \times n$
  - $k = -log_2 p$

p	m/n	k
0.1	4.8	3.3
0.01	9.6	6.6
0.001	14.0	10.0

Bloom Filter: Zusammenfassung

- Fehlerrate ist konfigurierbar
- prinzipiell ist es mögliche weiter Elemente einzufügen (aber Fehlerrate erhöht sich)
- Größe ist proportional zur Anzahl der Elemente in der Menge
- es gibt auch optimierte Varianten (z.B. Cuckoo Filter)

Count-Distinct Problem

- Problem: Bestimmung der Anzahl von eindeutigen Werten
- man kann dies mit einer Hashtabelle in $O(n)$ exakt lösen, benötigt aber $O(n)$ Speicher
- lässt sich approximativ/probabilistisch mit einer konstanten Menge an Speicher lösen
- keine gute Lösung: Extrapolation von einer Stichprobe konstanter Größe

Flajolet–Martin Algorithmus: Idee

- Sketch basiert auf Hashing
- je mehr eindeutige Werte es gibt, desto wahrscheinlicher ist es, dass man einen Hashwert erwischt, der am Ende aus mehreren Nullen besteht
- es gibt einen statischen Zusammenhang zwischen der Anzahl der Nullen am Ende und der Anzahl der eindeutigen Werte
- als Sketch braucht man also anstatt allen Hashwerten nur die maximale Anzahl von Nullen am Ende zu speichern

Flajolet–Martin Algorithmus (2)

- basiert wie der Bloom filter auf der Annahme, dass Hashwerte gleichverteilt sind
- man kann Pech haben und Ausreißer hat zufällig viele Nullen am Ende (obwohl die Menge klein ist)
- deswegen partitioniert man die Werte nach anderen Hashbits und merkt sich mehrere Maxima
- man hat also mehrere Sketches, die man zu einer Schätzung kombiniert (z.B. Durchschnitt oder Median)
- das erhöht die Genauigkeit deutlich

Count-Distinct Sketches: Zusammenfassung

- Schätzung der Anzahl von eindeutigen Elementen
- im Gegensatz zum Bloom filter ist die Größe für beliebig große Mengen konstant
- Genauigkeit kann eingestellt werden (Anzahl der Partitionen)
- verbesserte Variante: HyperLogLog
- sehr effizient (etwa 10 CPU Instruktionen), niedriger Speicherverbrauch (z.B. 64 bytes), ziemlich genau (z.B. 1%Fehler)

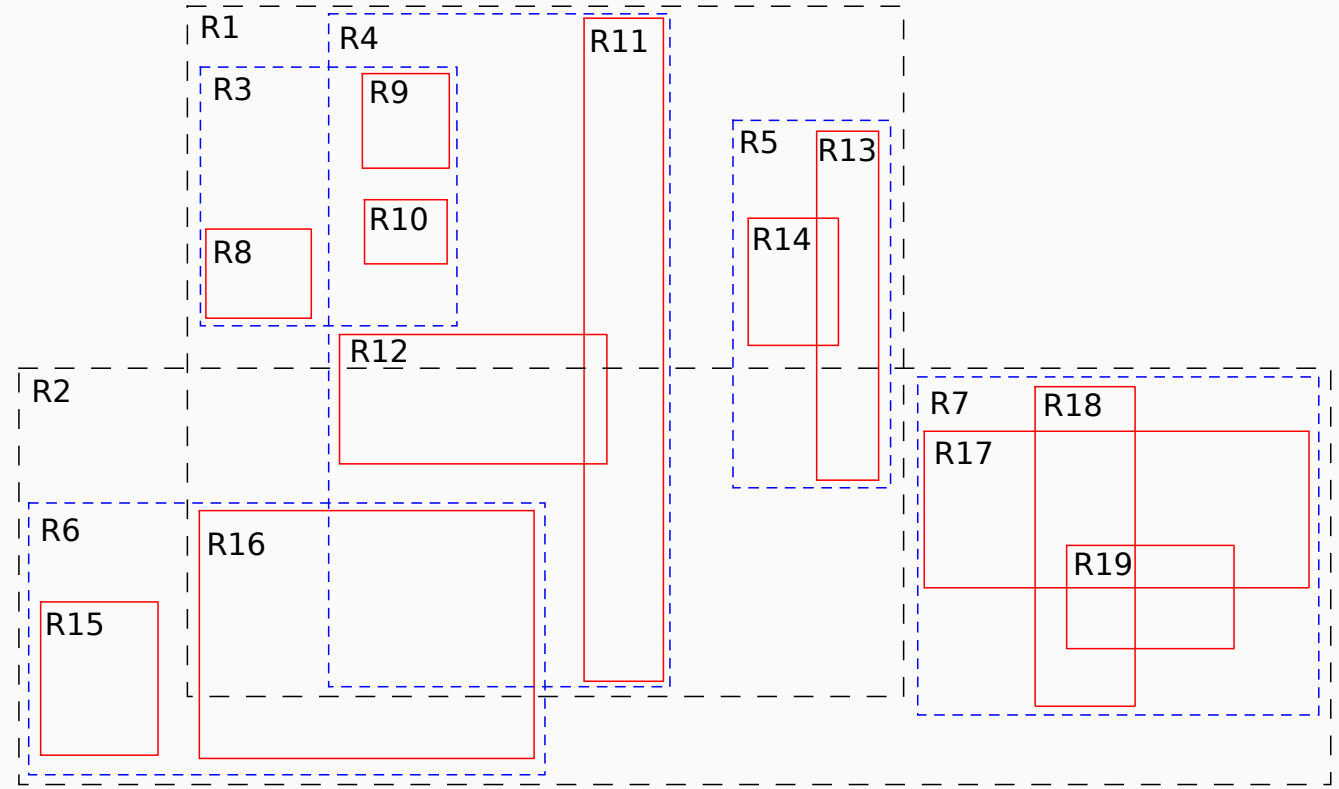
Mehrdimensionale Indexierung

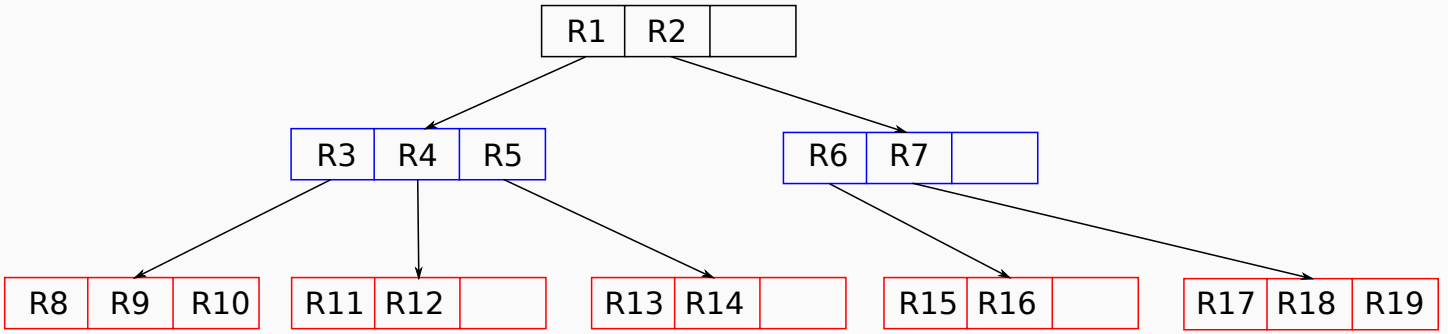
- klassische Indexstrukturen wie B-Bäume erlauben effiziente (z.B. $\log n$) eindimensionale Punkt- und Bereichsabfragen (“z.B. finde alle Mitarbeiter, die mehr als X verdienen”)
- bei mehrdimensionalen Anfragen hilft das aber nur bedingt
- Beispiele:
  - geografische Nachbarschaftsanfragen (“finde alle Restaurants in der Nähe”)
  - mehrere Bereichsprädikate (“finde alle Mitarbeiter, die mehr als X verdienen und jünger als Y sind”)
- erfordert spezielle, mehrdimensionale Indexstrukturen

R-Baum

- Idee: Unterteilung der Daten in geschachtelte Rechtecke (“bounding box”)
- Speicherung auf innere und Blattknoten fester Größe (ähnlich wie beim B-Baum)
- keine schönen theoretischen Laufzeitgarantien, funktioniert in der Praxis aber meist gut

R-Baum (2)


R1
R4
R11
R3
R9
R5
R13
R10
R14
R8
R12
R2
R7
R18
R17
R6
R16
R19
R15


R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19

R-Baum: Einfügen

- wenn ein Knoten voll ist, wird er in zwei Knoten aufgeteilt, deren Boxen nicht überlappen
- dabei sollen die beiden neuen Boxen möglichst klein sein
- exakte Berechnung der optimalen Aufteilung ist teuer, deswegen werden Heuristiken eingesetzt

R-Baum: Suche

- Vergleich der gesuchten Box bzw. des gesuchten Punktes mit allen Boxen im aktuellen Knoten
- wenn mehrere Überlappungungen gefunden wurden, müssen allen nachgegangen werden

R-Baum: Zusammenfassung

- kann auf mehr als zwei Dimensionen generalisiert werden
- funktioniert aber nur bei wenigen Dimensionen gut
- verfügbar in PostgreSQL zur Indexierung geografischer Daten (GiST Indexe)
- es gibt auch Alternativen (z.B. Quad-tree)

Indexierung für Volltextsuche

- ein weitere Art von Anfrage, die nicht direkt mit klassischen Indexen beantwortet werden kann, ist Volltextsuche (`select * from table where str like '%foo%'`)
- Lösung: Indexierung aller Suffixe
- um $O(n^2)$ Speicherverbrauch zu vermeiden, wird nicht der komplette Suffix gespeichert, sondern ein Offset
- bekannt als Suffix array bzw. Suffix tree

Zusammenfassung

- neben B-Bäumen werden in Datenbanksystemen auch weitere Datenstrukturen eingesetzt:
  - LSM-Baum (schreib-optimiert)
  - Bloom Filter (kompakte Menge mit einseitigen Fehlern)
  - Count-Distinct Sketch (Schätzung der Kardinalität einer Menge)
  - R-Baum (mehrdimensionale Indexierung)
  - Suffix-Baum (Volltextsuche)