

Datenbanken: **Indexe**

Motivation und Konzepte

Motivation

Warum sind Indexstrukturen überhaupt wünschenswert?

- Bei Anfrageverarbeitung werden Tupel aller beteiligter Relationen nacheinander in den Hauptspeicher geladen
- Nachteil: Sehr zeitintensiv
 - Zugriffslücke
 - ~ 10 ms Zugriffszeit
- Aber:
 - Meist erfüllt ein Großteil der Tupel nicht die Anfragebedingung
 - Festplatten erlauben wahlfreien Zugriff
 - Anfragen haben oft ähnliche Prädikate

Lösungsansatz

Wie kann man diese Umstände ausnutzen um Anfragen effizienter auszuführen?

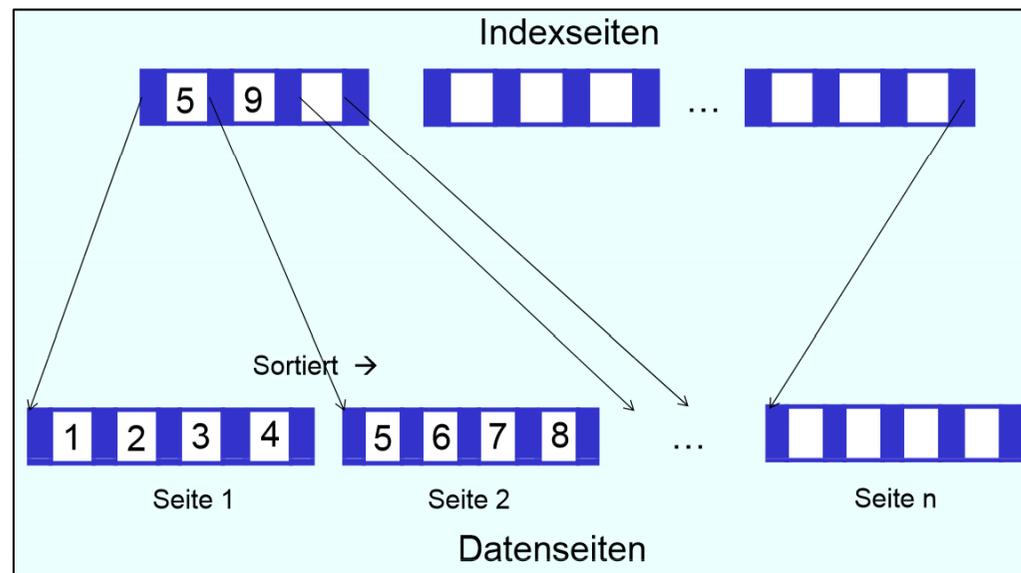
- Nur Teil der Daten der wirklich gebraucht wird soll in den Hauptspeicher geladen werden
- Dadurch kann das transferierte Datenvolumen klein gehalten werden (Geschwindigkeitsgewinn)

- Umsetzung durch „Nachschlagverzeichnisse“ (Indexe) für wichtigste(s) Attribut(e)
- Verschiedene Konzepte, u.a.:
 - Baumstrukturen (Hierarchisch)
 - Hashing (Partitionierung)

Umsetzung 1: ISAM

Ein erster Versuch der Umsetzung mit einer Ebene Indexseiten

- ISAM (Index-Sequential Access Method)
- Vorgänger von B-Bäumen
- Idee:
 - Sortieren der Tupel auf dem gewünschten (indexierten) Attribut
 - Indexdatei anlegen die auf entsprechende Tupel verweist



Quelle: Folienskript zur Vorlesung

Umsetzung 1: ISAM

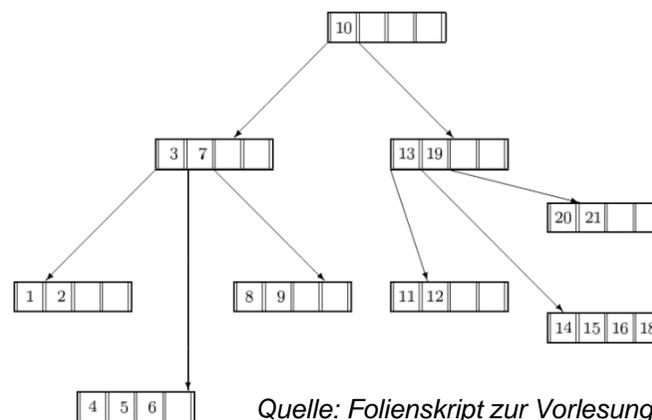
Vorteile und Nachteile

- Vorteile:
 - Anzahl Indexseiten \ll Anzahl Datenseiten daher weniger Disk I/O
 - Bereichsanfragen können (wegen Sortierung) beantwortet werden (Chained I/O)
- Nachteile:
 - Instandhaltung teuer (unflexibel, einfügen sehr aufwändig)
 - Relativ viele Indexseiten

Umsetzung 2: B-Bäume

Ein Index für den Index

- Einrichten einer Indexstruktur für die Indexseiten (s. ISAM)
- Für B-Bäume gelten bestimmte Eigenschaften:
 - Jeder Pfad von Wurzel (erster Knoten) zu einem Blatt (Knoten ohne Nachfolger) hat gleiche Länge
 - Jeder Knoten bis auf die Wurzel hat zwischen i und $2i$ Einträge ($i = \text{Grad des Baumes}$)
 - Einträge sind sortiert
 - Jeder Knoten bis auf die Blätter hat Kinder entsprechend der Anzahl seiner Einträge + 1
 - Knoten enthalten Daten und Verweise
 - Für ein Datum in einem inneren Knoten gilt: Daten in seinem linken Teilbaum sind immer kleiner und Daten in seinem rechten Teilbaum sind immer größer als dieses Datum
 - Teilbäume folgen den gleichen Regeln wie der gesamte Baum



Umsetzung 2: B-Bäume

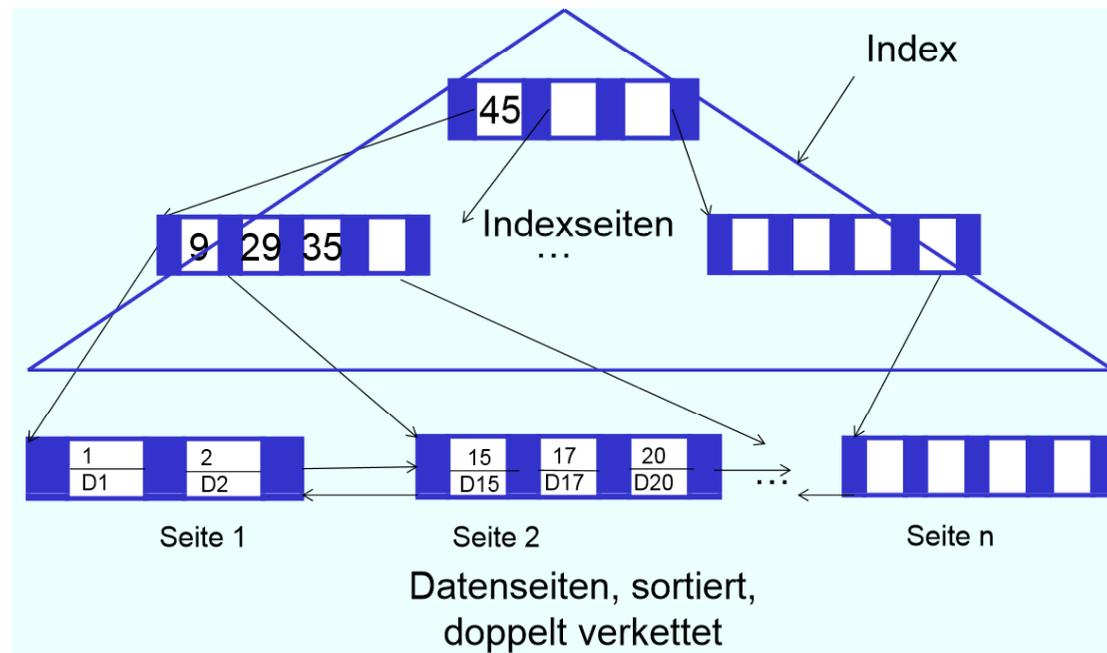
Vorteile und Nachteile

- Vorteile:
 - Schnelles Erreichen der Daten
 - Komplexität bestimmbar
- Nachteile:
 - Löschen- und Einfügealgorithmen relativ komplex zu implementieren
 - Performanz abhängig vom Verzweigungsgrad
 - Je höher der Verzweigungsgrad desto effizienter die Suche
 - Aber: Verzweigungsgrad wird reduziert durch Speichern von Daten in inneren Knoten

B+-Bäume

Der verbesserte B-Baum

- Optimierte Eigenschaften im Vergleich zum B-Baum:
 - Daten nur in Blättern (Dadurch höherer Verzweigungsgrad)
 - Verkettete Blattknoten (für sequentielle Abfrage → s. ISAM)
- Weitere Verbesserungsmöglichkeit: Schlüsselpräfixe

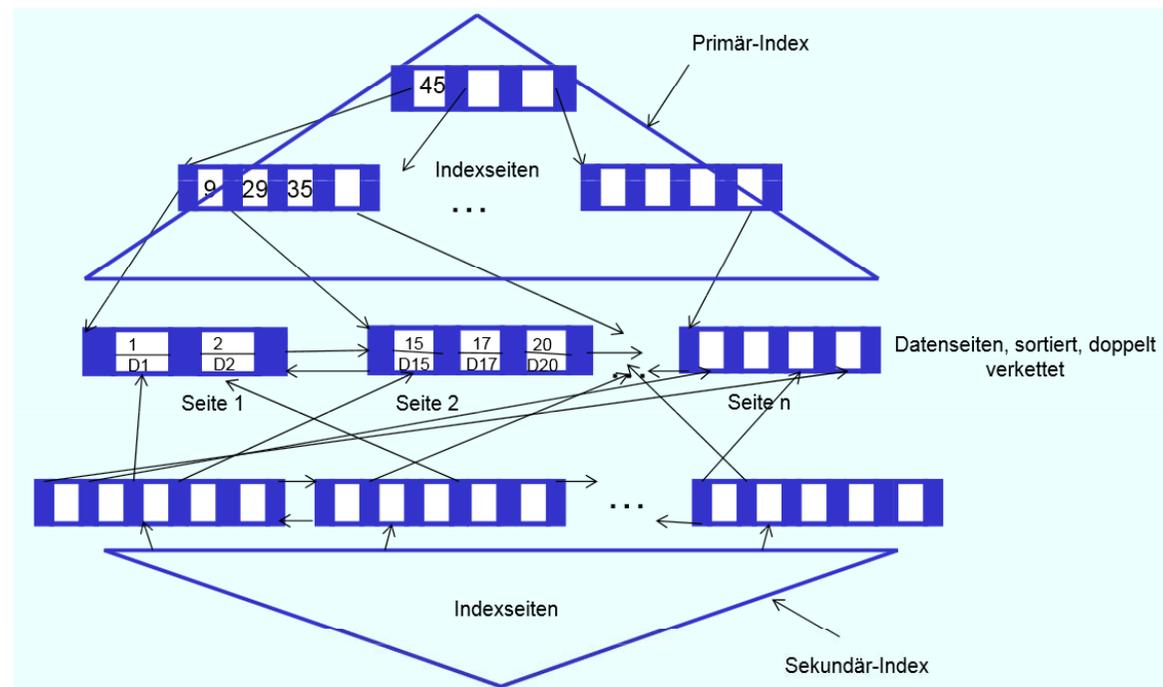


Quelle: Folienskript zur Vorlesung

Sekundärindexe

Indexierung auf mehreren Attributen

- Nützlich wenn mehrere Attribute wichtig und häufig in Anfragen enthalten sind
 - Indexe auf weiteren Attributen können erstellt werden
 - Daten können im Plattenspeicher nur nach einem Kriterium sortiert abgelegt werden
 - Index auf weiteren Attributen daher weniger effizient

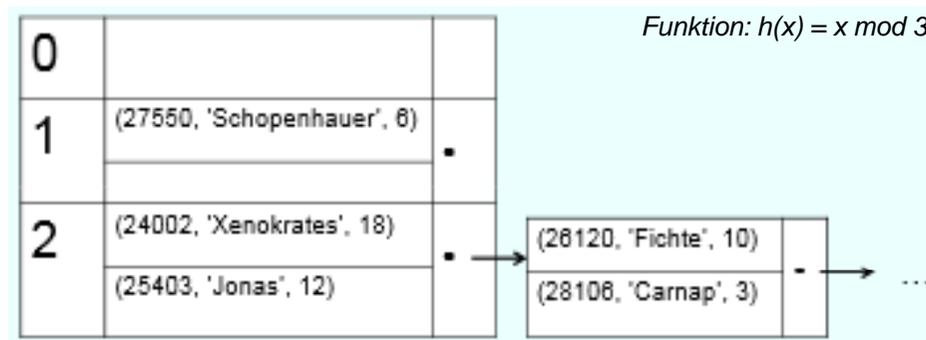


Quelle: Folienskript zur Vorlesung

Hashing

Daten zerhacken

- Daten werden durch „Hash-Funktion“ auf festgelegte Anzahl von Werten abgebildet
- Jedem Funktionswert wird ein Speicherbereich zugewiesen
 - In diesem Speicherbereich werden die jeweiligen Tupel abgelegt
 - Im Schnitt 2 Seitenzugriffe nötig ($\rightarrow O(1)$)
 - Vergleich: Komplexität bei Zugriff über Bäume: $O(\log n)$
- Nachteile:
 - Keine Bereichsabfragen möglich
 - Teils komplizierte Hashing-Funktionen notwendig (im Optimalfall: injektiv & surjektiv)
 - Datenmenge darf nicht zu dynamisch sein
 - Kollisionsbehandlung notwendig \rightarrow (zu) viele Datensätze auf einen Wert abgebildet (im Beispiel unten, wenn mehr als zwei Datensätze auf den gleichen Hashwert abgebildet werden)
 - Vorreservierung des Speichers nötig



Quelle: Folienskript zur Vorlesung