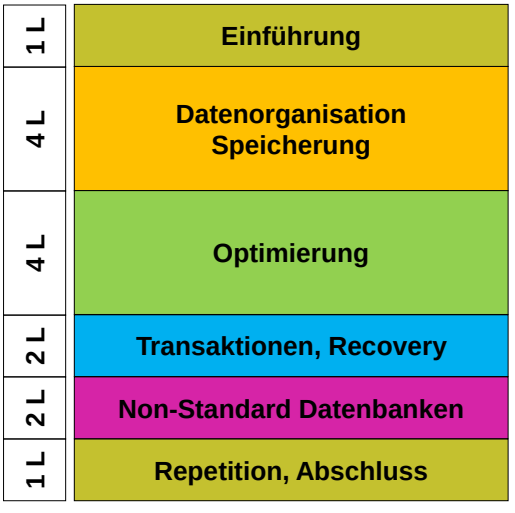
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Datenorganisation, Speicherung

Lehrbuch Kapitel 5.2 bis 5.4

Gliederung



"You are here"

Rückblick

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Aufgaben der Ebene Pufferverwaltung
- Grundaufbau von Indexen
- Grundlagen von Dateiorganisationsformen und Zugriffsstrukturen
- Speichertechniken
  - Klassierung
  - Aspekte von Indexen
    - dünn-/dicht-besetzt
    - Geclustert/nicht-geclustert
    - Ein-/mehrdimensional
    - …

Lernziele heute

- Verschiedene Zugriffsstrukturen (Dateiorganisationsform und/oder Zugriffspfad) verstehen
- Statische und dynamische Verfahren kennen

4

Zürcher Fachhochschule

Hier eine erste Übersicht über die Verfahren:

1. Statische Verfahren:
- Heap
- Sequentielle Datei
- Indexsequentielle Dateien
- Indexiert-nichtsequentielle Datei
- Statisches Hashing

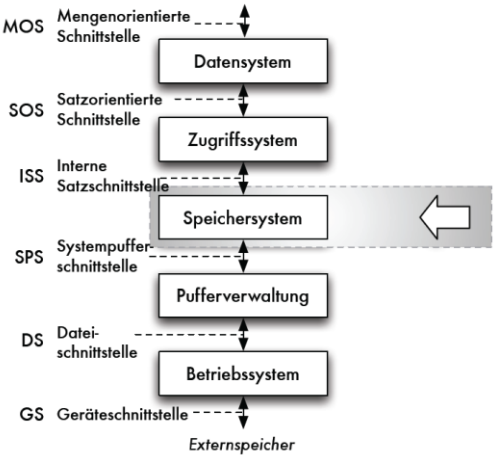
2. Dynamische Verfahren:
- B-Bäume
- Dynamisches Hashing (nicht behandelt in DAB2)

Speichersystem

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Letzten Lektion:
grundlegende Theorie zu Zugriffsstrukturen des Speicher-systems, sowie Aspekte von Indexen eingeführt.

Heue:
konkreten Verfahren.



Statische vs. dynamische Verfahren

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Statische Verfahren: Optimal nur bei bestimmter (fester) Anzahl von zu verwaltenden Datensätzen → periodische Reorganisation nötig
  - Heap
  - Sequentielle Datei
  - Indexsequentielle Dateien
  - Indexiert-nichtsequentielle Datei
  - Statisches Hashing
- Dynamische Verfahren: Unabhängig von der Anzahl der Datensätze optimal
  - Dynamische Indexverfahren: verändern dynamisch Anzahl der Indexstufen → in RDBMS üblich (B-Baum)
  - Dynamische Adresstransformationsverfahren: verändern dynamisch den Bildbereich (dynamisches Hashing)

6

Zürcher Fachhochschule

Begriff Heap im DB-Umfeld: Die HEAP-Speicherstruktur (Haufen) ist die sequentielle Speicherung der Daten in der Reihenfolge der Eingabe. Die Daten werden in Blöcken nacheinander auf der Platte abgelegt, die als lineare Liste miteinander verkettet sind. Gemäss Java-Spezifikation: «A heap file is an unordered set of records, stored on a set of pages».

Statische Zugriffsstrukturen sind Verfahren, die auf die Anzahl der Datensätze keine Rücksicht nehmen Z.B. ist ein dreistufiger Index immer dreistufig, egal ob 1 Datensatz oder Milliarden von Datensätze vorhanden sind. Ist die gewählte Zugriffstruktur für die aktuelle Anzahl unpassend, so muss eine ‘andere’ Struktur mittels Reorganisation gewählt, respektive erstellt werden (z.B. wenn die Datenmengen kontinuierlich anwachsen).

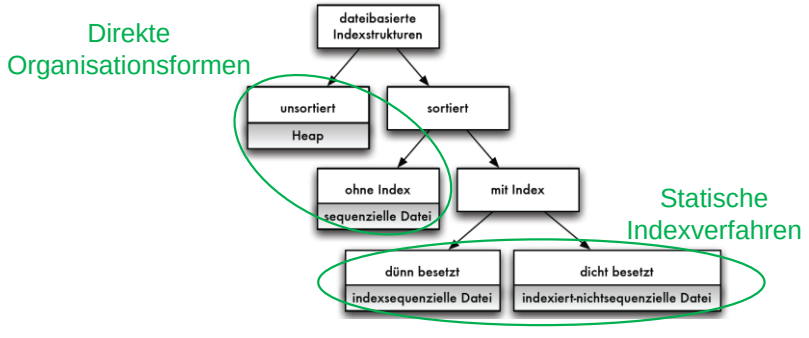
Dynamische Zugriffsstrukturen vergrössern oder verkleinern den Bildbereich des ‘Adresstransformationsverfahrens’ dynamisch in Abhängigkeit der Anzahl Datensätzen. So wird beim dynamischen Indexverfahren z.B. die Anzahl der Indexstufen zur Anwendungszeit entsprechend vergrössert oder verkleinert.

Da in Datenbanksystemen die Anzahl der Datensätze sich im Verlauf der Zeit meist deutlich ändert, sind die statischen Verfahren in der Mehrzahl der Fälle ungeeignet.

Statische Verfahren

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Direkte Organisationsformen: Keine Hilfsstruktur, keine Adress-berechnung (Heap, sequenzielle Datei)
- Statische Indexverfahren für Primärindex (indexsequenzielle Datei) und Sekundärindex (indexiert-nichtsequentielle Datei)
- Statisches Hashing (fehlt in Übersicht)



7

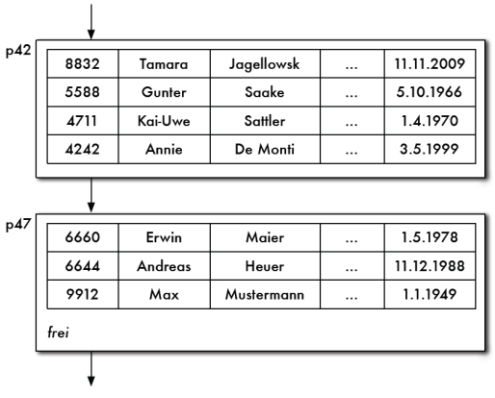
Zürcher Fachhochschule

Das Bild zeigt die statischen Verfahren im Überblick. Es fehlt allerdings noch das statische Hashing, welches auf diesen Folien als letztes der statischen Verfahren erläutert wird.

Heap ('Haufen')

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Daten werden unsortiert gespeichert (Heap, 'auf einem Haufen')
- Oft grundlegende Speichertechnik in RDBMS
- Z.B. in der zeitlichen Reihenfolge der Speicherung



8

Zürcher Fachhochschule

Die Abbildung zeigt einen Ausschnitt einer Heapdatei. Da eine sortierte Speicherung der Daten (Clustering, oder Heap) im Vergleich zur Führung von zusätzlichen Hilfsstrukturen (nach dem selben Sortierkriterium) aufwändiger ist, muss es gute Gründe geben, falls die Daten nicht als Heap gespeichert werden sollen. Ein solcher Grund ist z.B., wenn die Daten, die so auf die selbe physische Seite zu liegen kommen auch häufig gemeinsam verwendet werden (z.B. wenn häufig sortiert nach Primärschlüssel gelesen wird, oder wenn häufig Datensätze mit dem selben Fremdschlüssel gemeinsam gelesen werden).

Im SQL-Server werden die Daten der Hauptdatei per Default gemäss Primärindex geclustert (sortiert). Falls dies nicht erwünscht ist, muss manuell eingegriffen werden (das Clusterkriterium wird entfernt); dann kann eine unsortierte Datei erzeugt werden. Allerdings kann bei Heaps eine Heap-Fragmentierung auftreten (falls Datensätze verschoben werden müssen – z.B. weil sie länger werden – wird ein Zeiger auf den verschobenen Datensatz eingefügt). Wird die Hauptdatei sortiert (geclustert), entfällt das Problem der Heap-Fragmentierung. Die Vorteile der Sortierung überwiegen trotz Mehraufwand in aller Regel, so dass in SQL-Server nur in wenigen Ausnahmefällen ein Heap eingesetzt werden sollte.

Heap ('Haufen')

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Operationen:
  - Einfügen: Zugriff auf letzte Seite der Datei (z.B. in Data-Dictionary vermerkt).
  - Löschen: Suchen, dann Löschbit auf 0 setzen
  - Suchen: Sequenzielles Durchsuchen der Gesamtdatei
- Komplexitäten (n = Anzahl Datensätze):
  - Einfügen: O(1)
  - Suchen: O(n)
  - Löschen: O(n)

9

Zürcher Fachhochschule

Beim Einfügen muss, falls die letzte Seite voll ist, eine weitere Seite angefordert und verkettet werden.

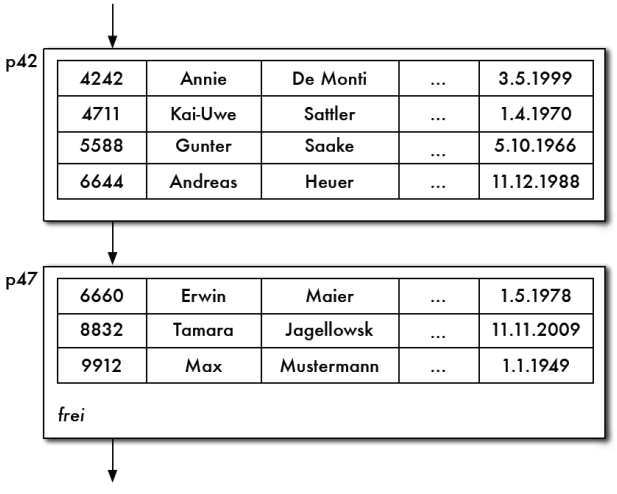
Während das Einfügen extrem Performant ist, ist das Löschen und Suchen sehr schlecht, der Aufwand wächst linear mit der Anzahl Tupel. Daher wird die Heap-Datei meist zusammen mit Sekundärindexen eingesetzt; oder für sehr kleine Relationen.

Sind beim Einfügen keine Duplikate erlaubt, ist vorher eine Suche notwendig. Da dies wie bereits erwähnt, sehr aufwändig ist, sollte hierzu ein Index gebildet werden.

Sequentielle Datei

School of Engineering
InIT Institut für angewandte Informationstechnologie

- Daten werden sortiert gespeichert



10

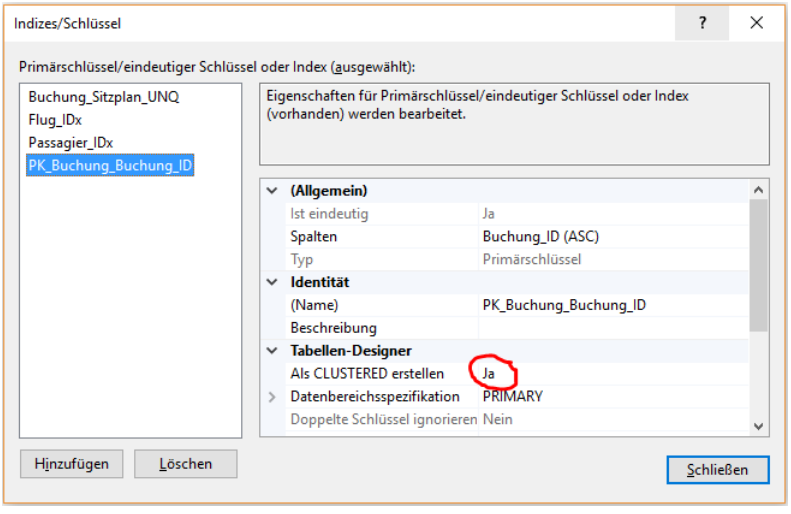
Zürcher Fachhochschule

Die Daten werden nach einem Attributwert oder einer Kombination von Attributwerten sortiert gespeichert (Clustering). Im Bild sind die Daten nach der Kundennummer sortiert.

Im SQL-Server ist die alleinige Sortierung der Datei nicht möglich, die Sortierung muss in Kombination mit einem «Index» erfolgen (der Index wird dabei durch einen B-Baum implementiert). Es kann daher bei genau einem Index angegeben werden, dass die Daten gemäss diesem Index geclustert werden sollen:

```
ALTER TABLE [dbo].[Buchung] ADD CONSTRAINT [PK_Buchung_Buchung_ID] PRIMARY KEY UNIQUE CLUSTERED
    ([Buchung_ID] ASC)
```

Oder mit dem SQL Server Management Studio:



Sequentielle Datei

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Operationen:
  - Einfügen: Seite suchen, Datensatz einsortieren: O(n)
  - Löschen: Suchen, dann Löschbit setzen: O(n)
  - Suchen: Sequenzielles Durchsuchen der Gesamtdatei: O(n/2)
    (Keine binäre Suche möglich, warum? Trotzdem schneller, warum?)

- Alle im folgenden besprochenen Dateiorganisationsformen bieten gegenüber Heap und sequentielle Datei:
  - Schnelleres Suchen
  - Mehr Platzbedarf (durch Hilfsstrukturen wie Indexdateien)
  - Mehr Zeitbedarf beim Einfügen und Löschen

11

Zürcher Fachhochschule

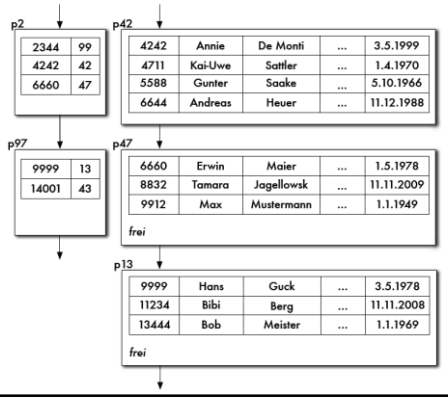
Beim erstmaligen sequenziellen Füllen einer Datei mit den Daten, wird mit Vorteil jede Seite nur bis zu gewissem Grad (z.B. 2/3) gefüllt. Dadurch wird das spätere Einfügen erleichtert. Sind die Seiten mehrheitlich gut gefüllt, ist eine Reorganisation der Datei angezeigt.

Beim Einsortieren müssen allenfalls Datensätze innerhalb der Seite verschoben werden. Da diese Operationen aber meist innerhalb einer Seite erfolgen, ist das Einfügen noch immer effizient.

Bei der sequentiellen Datei sind die Seiten mittels verketteter Liste verknüpft. Beim Suchen ist damit keine binäre Suche möglich, da nicht direkt auf Seiten zugegriffen werden kann (Sprung auf die Seite in der Mitte ist nicht möglich). Da die Datensätze sortiert sind, müssen im Schnitt aber nur die Hälfte der Seiten gelesen werden.

Indexsequentielle Datei

- Kombination von sequenzieller Hauptdatei und Indexdatei
- Indexsequentielle Dateiorgansationsform kann geclusterter, dünnbesetzter Index sein
- Mindestens zweistufiger «Baum». Die Blattebene ist Hauptdatei (die eigentlichen Datensätze), jede andere Stufe ist Indexdatei



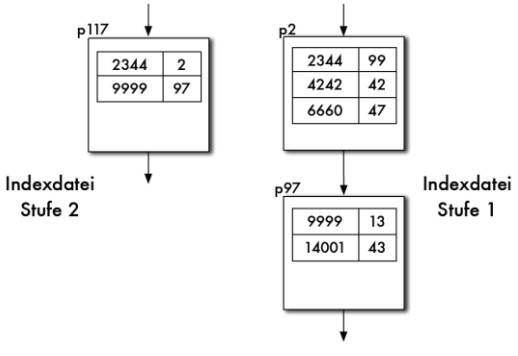
Die Indexdatei wird bei der indexsequentiellen Datei über das selbe Sortierkriterium gebildet, wie die Hauptdatei. Ein Index daher als geclustert bezeichnet, da dessen Sortierkriterium der Sortierung der Hauptdatei entspricht. In diesem Falle kann die Indexdatei dann auch dünn besetzt sein (wie im Bild).

Aufbau der Indexdatei: (Primärschlüsselwert, Seitennummer). Zu jeder Seite der Hauptdatei gibt es genau einen Index-Datensatz in der Indexdatei. Problem: Die «Wurzel» des Baumes umfasst bei einem einstufigen Index mehrere (verkettete) Seiten, die sequentiell durchsucht werden müssen.

Mehrstufiger Index: Die Indexdatei wird wieder indexsequenziell verwaltet. Idealerweise so, dass der Index höchster Stufe nur noch eine Seite umfasst (bester Performance-Fall). Die Anzahl Stufen ist aber weiterhin fix/statisch.

Der SQL-Server kennt keine Indexe im hier eingeführten Sinne. Im SQL-Server sind «Indexe» als B-Bäume implementiert, auch wenn diese nach aussen wie Indexe wirken.

Das Bild zeigt einen zweistufigen Index (Hauptdatei fehlt):



Indexsequentielle Datei

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Probleme bei indexsequentiellen Dateien:

- Anpassung Stufenanzahl Index ist nicht vorgesehen
- Index- und Hauptdatei-Seiten schrumpfen langsam
- Viele Mutationen führen zu unausgeglichenen Seiten in der Hauptdatei

13

Zürcher Fachhochschule

Bei wachsenden Dateien wächst die Zahl der linear verketteten Indexseiten, eine automatische Anpassung der Stufenanzahl ist aber nicht vorgesehen. Dadurch wächst beim Suchen die Anzahl zu lesender Seiten linear mit der Datenmenge.

Bei schrumpfende Dateien wird dagegen die Zahl der Index- und Hauptdatei-Seiten nur zögernd verringert (erst wenn diese vollständig leer sind werden sie entfernt). Dadurch wir die Zugriffzeit ebenfalls negativ beeinflusst.

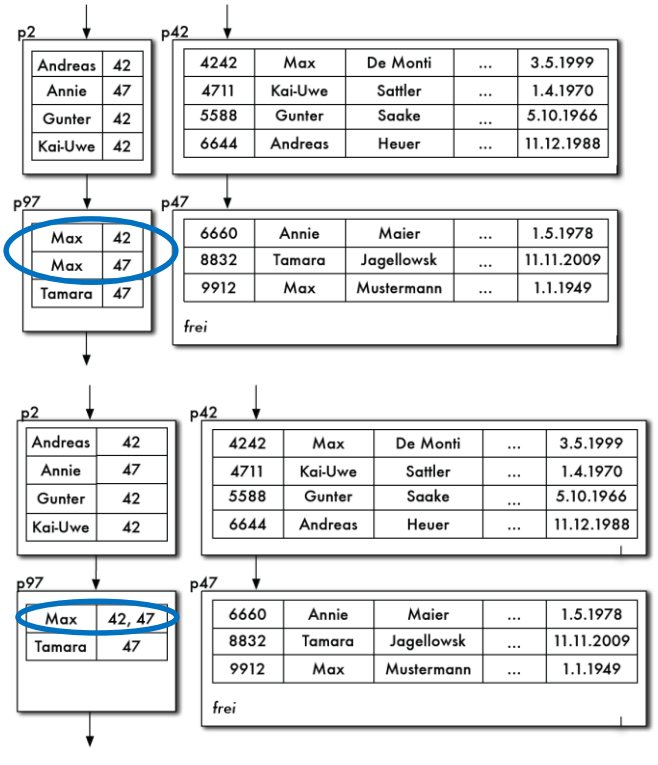
Viele Mutationen führen zu unausgeglichenen, schlecht gefüllten Seiten in der Hauptdatei, woraus unnötig hoher Speicherplatzbedarf und lange Zugriffszeiten resultieren.

Indexiert-nichtsequentieller Zugriffspfad

- Keine Dateiorganisationsform, Hauptdatei kann unsortiert sein:
  - Unterstützt Sekundärschlüssel
  - Mehrere Zugriffpfade möglich
  - Einstufig oder mehrstufiger Index
- Aufbau des Sekundärindex:
  - Dichtbesetzt und nicht-geclustert
  - Besitzt zu jedem Satz der Hauptdatei einen Indexdateisatz (w,s)
    - w: Sekundärschlüsselwert
    - s: zugeordnete Seite
  - Da ein Sekundärschlüsselwert mehrfach vorkommen kann, werden
    a. entweder für ein w mehrere Sätze in die Indexdatei aufgenommen, oder
    b. für ein w eine Liste von Adressen in der Hauptdatei angeben

Der indexierte-nichtsequentielle Zugriffspfad (= Sekundärindex) stellt keine Dateiorganisationsform dar. Die Hauptdatei kann als Heap, indexsequentielle Datei oder mittels Hashfunktion gebildet werden, ist aber nicht nach den dem Sekundärschlüssel sortiert. Je Sekundärschlüssel (und Datei) wird ein Sekundärindex erstellt. Der Sekundärindex muss immer dichtbesetzt sein und kann nicht-geclustert sein.

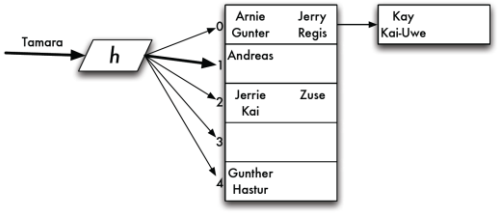
Der SQL-Server implementiert auch diese Indexe als B-Bäume.



Statisches Hashing

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Idee: Abbildung eines Ausgangsbereiches von Werten auf einen Zielbereich, so dass möglichst wenige Werte des Ausgangsbereiches auf denselben Zielwert abgebildet werden
- Benötigt 'Überlaufbehandlung' bei Kollisionen
- Im DB-Bereich: Abbilden von Schlüsselwerten auf Seitenadressen



15

Zürcher Fachhochschule

Die Grundlagen zum Hashing werden als bekannt vorausgesetzt: Vorlesung ADS (sonst selbständig erarbeiten anhand Lehrbuch, Kapitel 5.4).

Statisches Hashing

- Basis-Hashfunktion: h(k) = k mod m
- Anforderungen an h:
  - effizient berechenbar
  - Kollisionen minimieren
  - 'vernünftiger' Bildbereich
- Strategien zur Behandlung von Kollisionen:
  - Überlaufbereich
  - Sondieren (d.h. noch nicht benutzte Seitenadresse suchen)
    - linear → führt oft zu 'Werteclustern' (+1, +2, +3, …)
    - quadratisch (Seite: +1, +4, +9, …)
    - …
  - 'Doppeltes Hashen': Zum Sondieren wird eine weitere Hashfunktion benutzt

16

Zürcher Fachhochschule

Bei Hashfunktionen wird häufig die Modulo-Funktion in Kombination mit einer Primzahl verwendet. Diese Hashfunktion erfüllt die Anforderungen an h im allgemeinen gut. Aber: wieso werden Primzahlen verwendet?

Der berechnete Wert soll von der kompletten Eingabe abhängen, weil man hofft, dass so eher eine gleichförmige Verteilung erreicht wird. Werden z.B. CHF-Beträge (gerundet auf 5 Rappen) in einer Hashfunktion verwendet, dann würden mit Modulo 10 nur die Werte 5 und 0 berechnet. Dieses Problem stellt sich immer, wenn es einen grössten gemeinsamen Teiler grösser 1 zwischen dem gewählten Modul und den Eingabewerten gibt; der ist bei Primzahlen definitionsgemäss die Primzahl selbst und somit das Optimum.

Wenn die Eingabewerte endlich und bekannt sind (z. B. Postleitzahlen zwischen 1000 und 9999), kann es deutlich schneller zu berechnende Funktionen geben, z. B. die Postleitzahl – 1000 (ergibt Seitenadressen 0 bis 8999).

Statisches Hashing

- Modifikationen für RDBMS (Bildbereich ist nicht linear):
  - Abbilden von Primärschlüsselattributwerten auf sog. Bucket-Adressen (eine oder mehrere Seiten) mittels eines Hash-Verzeichnisses
  - Hash-Verzeichnis: Abbildung Hashwert h(k) auf Bucket-Adresse
- Suche ist nun sehr effizient: Minimal nur zwei Seitenzugriffe!
  Hashverzeichnis und erste Seite des Buckets
- Probleme:
  - Anpassung an wachsende/schrumpfende Datenmengen ('statisch')
  - grosser Abbildungsbereich → schlechte Speicherauslastung
  - kleiner Abbildungsbereich → viele Kollisionen → Überlaufseiten
  - Reorganisation teuer: 'Re-Hashen'
  - ungeeignet für Bereichsanfragen: 'alle Datensätze sortiert ausgeben'

17

Zürcher Fachhochschule

Hashing wird im Allgemeinen für die Adressierung im Hauptspeicher verwendet. Eine Adresse beinhaltet dabei häufig exakt ein Element.

Bei Datenbanksystemen verweist die berechnete Adresse aber auf eine oder mehrere Seiten (Bucket), welche wieder mehrere Datensätze enthalten kann. Diese Seiten, der Bucket muss beim Suchen so lange durchsucht werden, bis der gesuchte Datensatz gefunden wurde, oder bis man sicher ist, dass dieser nicht existiert.

Ein Problem ist, dass die Seiten, die für das Hashing verwendet werden, keinen linearen Adressraum bilden. Die Seiten sind verstreut. Es muss daher eine **zusätzliche Abbildung** zwischen dem mittels Hashing berechneten linearen Adressraum und den verstreuten Seiten erfolgen. Diese Abbildung erfolgt durch das zusätzliche **Hash-Verzeichnis**.

Der SQL-Server (ab Version 2014) kennt Hashing bei Indexen für Memory-optimierte Tabellen. Bei In-Memory-Datenbanken wird der Arbeitsspeicher des Computers als Datenspeicher genutzt (anstelle von Festplatten). Im SQL-Server sind dabei die ACID-Eigenschaften weiterhin vollständig erfüllt.

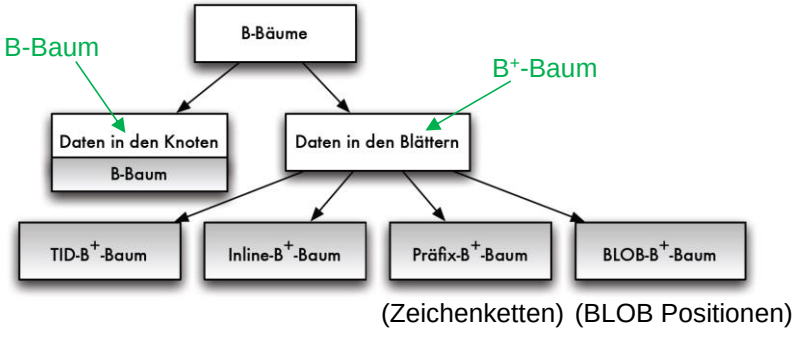
Dynamische Verfahren

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- B-Baum (= vollständig balancierter Baum)
- Dynamisches Hashing

B-Bäume

- Dynamisches Verfahren: Anpassung der Anzahl Stufen
- Allgegenwärtig im Datenbankbereich (nicht nur bei RDBMS)
- Viele Varianten:



19

Zürcher Fachhochschule

Die Grundlagen zu Bäumen werden als bekannt vorausgesetzt: Vorlesung ADS (sonst selbständig erarbeiten anhand Lehrbuch, Kapitel 5.4).

B-Bäume sind absolut zentral für Datenbanksysteme. Sie sind auch in nicht-relationalen Datenbanksystemen als Grundtechnik implementiert.

Präfix-B$^+$-Baum und BLOB-B$^+$-Baum sind für deren jeweiligen Datentypen optimiert. Wir werden diese Varianten in der Vorlesung nicht weiter vertiefen.

B-Bäume

- Wer hat's erfunden? Bayer & McCreight
- **B** steht für **B**ayer/**B**oeing, **b**alanciert, … (nicht: binär!)
- Datenbankbereich: Die Knoten der Suchbäume sind zugeschnitten auf die Seitenstruktur des Datenbanksystems → mehrere Zugriffsattributwerte auf einer Seite
- Balanciert (oder ausgeglichen):
  - Alle Wege von der Wurzel zu den Blättern sind gleich lang.
- Vollständig balanciert:
  - Alle Wege von der Wurzel zu den Blättern sind gleich lang.
  - jede Seite ausser der Wurzelseite enthält zwischen m und 2m Einträge, m = Ordnung des Baumes
  - Jede Seite ist entweder eine Blattseite oder hat i+1 Nachfolger, falls i die Anzahl ihrer Elemente ist

20

Zürcher Fachhochschule

In einer ersten Version wurde für den *vollständig* balancierten Baum verlangt, dass jeder Knoten gleich viele Indexeinträge enthält. Diese Forderung führt zu aufwändigeren Algorithmen, so dass die Forderung aufgeweicht wurde: jede Seite (ausser der Wurzel) enthält zwischen m und 2m Einträge. Da 2m Einträge 100% Füllung entspricht, ist der Baum mindestens zu 50% gefüllt. Für diese Anforderung gibt es effiziente Algorithmen zum Suchen, Einfügen und Löschen von Daten.

Leider wird die Ordnung des Baumes nicht einheitlich angegeben. Als Ordnung wird fallweise m oder 2m verwendet. In unserem Sinne hat ein Baum der Ordnung 2 zwischen 2 und 4 Einträge in einem Knoten.

B-Bäume

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Eigenschaften:

- Bei n Datensätzen und m Einträgen je Knoten braucht es $\log_m(n)$ Seitenzugriffe von der Wurzel zum Blatt
- Balancierung garantiert mindestens 50% Speicherplatzausnutzung
- Schnelle Verfahren zum Suchen, Einfügen und Löschen: $O(\log_m(n))$
- Geeignet für Primär- und Sekundärindexe

21

Zürcher Fachhochschule

Die Balancierung gewährleistet, dass 50% des Speichers genutzt werden, d.h. die Knoten sind mind. zur Hälfte gefüllt.

<u>Kleines Rechenbeispiel, nehmen wir an:</u>

- Ein Knoten des Baumes hat etwa 8 KByte (darin sind auch Header und weitere Daten abgelegt)
- Der Index verwendet ein Attribut das 30 Byte lang ist
- Die Tabelle enthält 10’000’000 Datensätze

Dann hat der vollständig gefüllte Baum etwa eine Tiefe von $\log_{(8000/30)}(10'000'000)=2.88$, also etwa 3. Das Suchen eines einzelnen Tupels benötigt damit etwa 4 Seitenzugriffe.

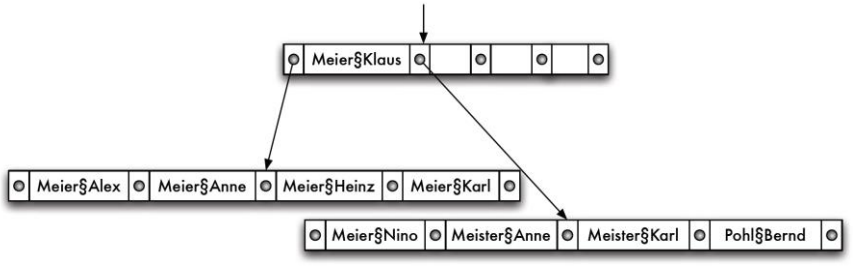
Hat es in der Tabelle 10’000 Datensätze ergibt sich eine Tiefe von 1.64. also etwa 3 Seitenzugriffe zum Suchen eines Tupels. Das Anwachsen der Datenmenge verschlechtert die Performance daher nur marginal.

Wird die Länge des Index aber auf 120 Byte vergrössert, ergibt sich bei 10’000’000 Datensätzen: $\log_{(8000/120)}(10'000'000)=8.50$. D.h. es sind im Schnitt 9 Seitenzugriff notwendig.

Es ist daher eine gute Idee, einen Index möglichst über kurze Attribute zu bilden!

B-Bäume – Varianten

- Mehr-Attribut B-Baum: B-Baum ist eine eindimensionale Indexstruktur, jedoch können mehrere Attribute als zusammengesetzter Schlüssel indexiert werden
- Beispiel: `CREATE INDEX NameIdx ON KUNDE(Name, Vorname)`
- Allerdings: Attribute werden bei partial-match-Anfragen nicht gleich behandelt!



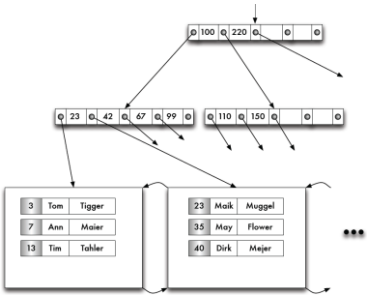
22

Zürcher Fachhochschule

Partial-match-Anfragen sind Anfragen, bei denen nicht alle Schlüsselwerte, sondern nur eine Teilmenge der Schlüsselwerte zur Suche angegeben werden. Im Bild gezeigten Baum, ist z.B. die Suche nach Nachnamen (ohne Vornahme) sehr effizient, während die Suche nach Vornahme (ohne Nachnahme) sämtlich Knoten des Baumes überprüfen muss (immer noch schneller, als die gesamte Hauptdatei zu durchsuchen).

B-Bäume – Varianten

- B^+-Baum: nur Blattebene enthält Daten → Baum ist 'hohl'
- In der Praxis am häufigsten eingesetzte Variante des B-Baumes:
  - In inneren Knoten nur noch Attributwert und Zeiger auf nachfolgende Knoten (Seite) der nächsten Stufe, keine Daten
  - Nur Blattknoten enthalten neben Attributwert die Daten (Datensätze der Hauptdatei oder Verweise darauf)
  - Knoten der Blattebene sind für effiziente Unterstützung von Bereichsanfragen untereinander verkettet



Datensätze innerhalb Blätter:
Inline-B^+-Baum

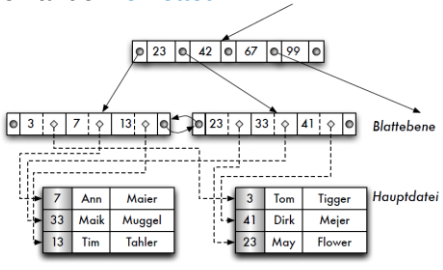
Beim B^+-Baum enthalten die inneren Knoten **keine** Daten oder Referenzen auf die Daten. Die Daten, oder in einer Variante die Verweise auf die Daten, sind nur in den Blättern des Baumes abgelegt. Bei einer Suche muss daher immer bis zu den Blattseiten des Baumes gesucht werden, erst dort werden die Daten, respektive eine Referenz auf die Daten gefunden.

Sind die eigentlichen Datensätze in den Blättern des Baumes gespeichert, sprechen wir auch vom Inline-B^+-Baum (siehe Bild auf Folie), was uns stark an die indexsequentielle Datei erinnert. Sind die Datensätze ausserhalb des Baumes gespeichert (siehe nächste Folie), in den Blättern also nur Referenzen auf die Daten abgelegt, sind diese Datensätze nicht geclustert und die sequenzielle Verarbeitung ist damit viel aufwändiger.

Im SQL-Server entspricht der Inline-B^+-Baum einer Tabelle mit geclustertem Index.

B-Bäume – Varianten

- B^+-Baum: nur Blattebene enthält Daten → Baum ist 'hohl'
- In der Praxis am häufigsten eingesetzte Variante des B-Baumes:
  - In inneren Knoten nur noch Attributwert und Zeiger auf nachfolgende Knoten (Seite) der nächsten Stufe, keine Daten
  - Nur Blattknoten enthalten neben Attributwert die Daten (Datensätze der Hauptdatei oder Verweise darauf)
  - Knoten der Blattebene sind für effiziente Unterstützung von Bereichsanfragen untereinander verkettet



Datensätze ausserhalb Blätter: TID-B^+-Baum

24

Zürcher Fachhochschule

Im SQL-Server ist jeder Index der nicht geclustert ist ein TID-B^+-Baum.

Soweit die Theorie zu B-Bäumen. Im täglichen Einsatz des SQL-Server gibt es weitere Punkte die zu beachten sind. Die Forderung, dass die Knoten zu mindestens 50% gefüllt sind, bedeutet im schlechtesten Fall, dass doppelt so viele Seiten gelesen werden müssen und dass doppelt so viel Speicher benötigt wird, wie im optimalen Fall (vollständig gefüllt). In der Praxis wird versucht, diesen mangelhaften Füllgrad zu verhindern. Der SQL-Server hat daher verschiedene Mechanismen um den Füllgrad höher zu halten. Hier soll das wichtigste davon angesprochen werden.

Beim Erstellen eines Index kann im SQL-Server der Füllgrad der Knoten angegeben werden. Dieser ist per Default 100% und sollte nur in wenigen, begründeten Fällen etwas kleiner gewählt werden (z.B. 90%), um das Einfügen neuer Datensätze zu beschleunigen.

Durch den täglichen Gebrauch (Ändern und Löschen von Daten) können B-Bäume aber fragmentiert werden. Dabei treten zwei Arten von Fragmentierung auf:

- Interne Fragmentierung: Misst den Füllgrad der Seiten, welche die Knoten speichern. So bedeutet der Füllgrad von 70%, dass 70% des Speicherplatzes belegt ist. Was dann entsprechend mehr Zugriffe und Speicherplatz bedeutet.
- Externe Fragmentierung: Falls die physische Reihenfolge der Seiten des Index nicht in der logischen Reihenfolge des Index entspricht, spricht man von externer Fragmentierung (analog der Fragmentierung einer Datei). Dies hat dann Einfluss auf die Performance, wenn die Daten in der Reihenfolge dieses Indexes sequentielle von einer Harddisk gelesen werden sollen (der Lesekopf der Harddisk muss dann öfter springen).

In Wartungsjobs kann dieser Füllgrad überwacht werden und im Bedarfsfall der Index neu erstellt werden (Aufgabe des DB-Administrators): `ALTER INDEX <index_name> [ REBUILD | REORGANIZE]`.

Zusätzlich müssen die statistischen Informationen, die der Optimizer nutzt, periodisch aufgefrischt werden – dazu mehr in späteren Lektionen.

Dynamisches Hashing

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Dynamisches Hashen wird nicht in DAB2 behandelt
(für Interessierte: siehe Lehrbuch Kapitel 6.1).

Ansatz: Der Abbildungsbereich der Hash-Funktion muss dynamisch angepasst werden können, dies erfordert eine neue Hash-Funktion:

- Verschiedene Verfahren: lineares Hashen, Spiralhashen, etc.
- Beim Vergrössern ist KEIN kompletes Re-Hashen notwendig,

Ausblick

- Das nächste Mal: Basisalgorithmen
- Lesen: Lehrbuch Kapitel 7.1 bis 7.5