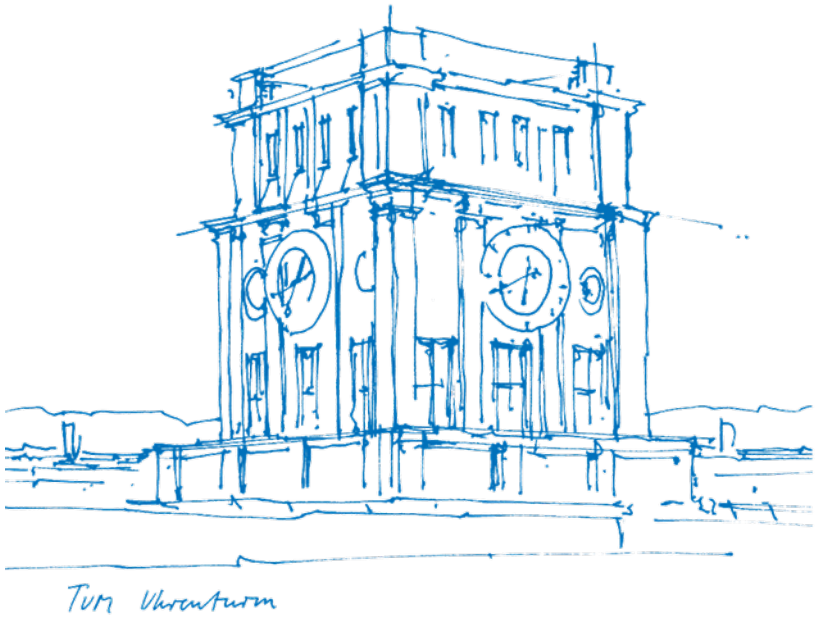
Grundlagen Datenbanken

Benjamin Wagner

15. Januar 2019



Allgemeines

- Folien von mir sollen unterstützend dienen. Sie sind nicht von der Übungsleitung abgesegnet und haben keinen Anspruch auf Vollständigkeit (oder Richtigkeit).
- Bei Fragen oder Korrekturvorschlägen: wagnerbe@in.tum.de
- Vorlesungsbegleitendes Buch von Professor Kemper (Chemiebib)
- Mein Foliensatz ist online: `https://github.com/wagjamin/GDB2018`

Speicherhierachie

- Moderne Rechner haben verschiedene Arten des Speichers
- Dieser ist hierarchisch angeordnet: größerer Speicher ist langsamer
- Bei Festplatten kann zusätzlicher Aufwand entstehen (z.B. disk seek)

Speicher	Größe	Latenz	Vergleich
Register	bytes	1ns	Schreibtisch
Cache	K-M bytes	<10ns	Zimmer
Hauptspeicher	G bytes	<100ns	Nachbarschaft
Externer Speicher	T bytes	1ms	Tokyo

RAID

- Ziel: **Hoher Durchsatz**, **Fehlertoleranz** in Festplattenverbund
- **RAID0:** Block-Striping
- **RAID1:** Block-Mirroring
- **RAID3:** Bit-Striping & Paritätsplatte
- **RAID4:** Block-Striping & Paritätsplatte
- **RAID5:** Block-Striping, verteilte Paritätsblöcke
- RAID5 bietet gutes Verhältnis zwischen Overhead & Leistung

Buffer Manager

- Historisch waren Datenbestände größer als der Hauptspeicher
- Es mussten zusätzlich Daten auf Festplatten gespeichert werden
- Hierfür werden Tupel in ”Slotted Pages” angeordnet
- Buffer Manager verwaltet, welche Slotted Pages im Hauptspeicher sind

Datenstrukturen

- Datenstrukturen sollen diesen Aufbau berücksichtigen
- Wenige Speicherzugriffe, große und zusammenhängende Speicherbereiche
- Trotzdem noch schnelle Zugriffszeiten

$\Rightarrow$ Klassische Binärbäume reichen nicht aus

- **Frage:** Warum nicht? Was verschafft Abhilfe?

B-Bäume

- Knoten speichern (Tupel, Pointer) Paare
- Ein Knoten kann viele hundert Einträge haben
- Binäre Suche im Knoten
- Knoten können ebenfalls auf Festplatte ausgelagert werden
- B-Baum garantiert Auslastung von $\geq 50\%$
- B^+-Baum speichert Daten nur in Blättern
- **Achtung:** Löschen und Einfügen kann Knotenstruktur ändern

B^+-Bäume

- Datenbanken benutzen anstatt B-Bäumen in der Regel B^+-Bäume.

  Warum?

B^+-Bäume

- Datenbanken benutzen anstatt B-Bäumen in der Regel B^+-Bäume.
  Warum?
- Mehr innere Knoten pro Blatt, einfache Bereichsabfragen
- In der Regel haben Blätter *next*-Zeiger zum (rechten) Nachbarblatt
- Oft werden nur *inserts*, aber keine *deletes* unterstützt
- Mehrbenutzersynchronisation von B/B^+-Bäumen schwierig

B^+-Bäume - inserts

1. Suche Blatt, in welches Schlüssel eingefügt werden soll
2. Falls noch Kapazität:
   - Füge Schlüssel ein
3. Falls keine Kapazität mehr:
   - Teile vollen Knoten in zwei neue Knoten auf
   - Füge den Schlüssel in einen der neuen Knoten ein
   - Füge mittleren Eintrag in Vaterknoten $\rightarrow$ Schritt 2 *
   - Falls kein Vaterknoten: lege neue Wurzel an

- ***Achtung**, bei inneren Knoten müssen inserts wie in klassischen B-Bäumen ablaufen
- **Achtung:** splits können sich bis zur Wurzel fortpflanzen
- **Frage:** wie verhalten sich *insert* und *lookup* asymptotisch?

Hashing

- Hashtabellen eignen sich für Punktanfragen
- **Frage:** wie verhalten sich *insert* und *lookup* asymptotisch?

Hashing

- Hashtabellen eignen sich für Punktanfragen
- **Frage:** wie verhalten sich *insert* und *lookup* asymptotisch?
- Es wäre interessant, Hashtabellen als Index zu nutzen
- **Aber:** Wie kann ich eine Hashtabelle auf Disk auslagern? Was passiert, wenn die Hashtablle voll ist?

Hashing

- Hashtabellen eignen sich für Punktanfragen
- **Frage:** wie verhalten sich *insert* und *lookup* asymptotisch?
- Es wäre interessant, Hashtabellen als Index zu nutzen
- **Aber:** Wie kann ich eine Hashtabelle auf Disk auslagern? Was passiert, wenn die Hashtablle voll ist?
- I.d.R. nicht sinnvoll, alle Werte neu in gößere Tabelle einzufügen
- Erweiterbares Hashing (*engl. extendible hashing*) löst diese Probleme
- Hashing auch in Anfragebearbeitung interessant (Hash-Joins)

Erweiterbares Hashing

- Größe der Hashtabelle ist immer Zweierpotenz
- Daten werden in Buckets gespeichert
- Mehrere Indizes der Hashtabelle können auf gleiche Buckets zeigen
- Inserts können in drei Fälle unterteilt werden:
  - Bucket hat noch Platz $\Rightarrow$ kein Problem
  - Bucket is voll, abere mehrere Indizes zeigen auf gleichen Bucket $\Rightarrow$ splitte Buckets
  - Es zeigt nur ein Index auf den vollen Bucket $\Rightarrow$ verdopple Größe der Hashtabelle
- Wachstum ist nun weniger invasiv, es muss nicht jeder Bucket neu angepasst werden
- **Aber:** Verdoppelung der Tabellengröße trotzdem sehr invasiv
- **Ausblick:** Linear Hashing, Multi Level Extendible Hashing

Mehrdimensionale Indexstrukturen

- Was passiert, wenn wir unsere Daten mehrdimensional sind?
- Z.b. für Events mit Ortsangaben
- Wir brauchen Indexstrukturen, die Anfragen auf solchen Daten effizient möglich machen
- Hier gibt es viele Möglichkeiten: Grid Files, R-Bäume, R^+-Bäume

R-Baum

- **Idee:** Teile Raum iterativ in disjunkte Rechtecke auf
- Wie beim B-Baum kann es mehrere Ebenen geben
- Splitten von Knoten ist oft aufwendig und nicht immer eindeutig