TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS20/21

Christoph Anneser, Josef Schmeißer, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws2021/grundlagen/

Blatt Nr. 08

Tool zum Üben von funktionalen Abhängigkeiten: https://normalizer.db.in.tum.de/.

Hausaufgabe 1

Geben Sie für jede der Normalformen 1NF, 2NF, 3NF, BCNF, 4NF jeweils eine Relation mit FDs an, sodass die Relation in der gewünschten Normalform ist (und in keiner höheren).

Lösung:

Für alle Normalformen betrachten wie die Relation $\mathcal{R} = \{A, B, C, D\}$.

- 1.NF:

  FDs:

  - $AB \to C$
  - $B \to D$

  Die Relation ist nicht mengenwertig, daher 1. NF. D ist lediglich von B abhängig, der Kandidatenschlüssel ist aber AB, weswegen D nicht voll funktional vom Kandidatenschlüssel abhängig ist, daher keine 2. NF.

- 2. NF:

  FDs:

  - $AB \to C$
  - $C \to D$

  Jedes Attribut der Relation ist voll funktional abhängig vom Kandidatenschlüssel AB, daher 2.NF. Das Attribut D ist transitiv und nicht direkt vom Kandidatenschlüssel abhängig, darum nicht 3. NF.

- 3. NF:

  FDs:

  - $AB \to CD$
  - $BC \to AD$
  - $D \to C$

  Für alle FDs gilt entweder, dass sie trivial ist, dass die linke Seite Superschlüssel ist oder dass die rechte Seite in einem Kandidatenschlüssel enthalten ist, daher 3. NF. Bei der BCNF fällt die dritte erlaubte Art von FD weg, daher FDs müssen trivial sein oder ihre linke Seite Superschlüssel. Da die dritte FD des Beispiels dies verletzt ist die Relation nicht in BCNF und daher genau in 3. NF.

- BCNF:

  FDs:

  - $AB \to CD$
  - $BC \to AD$
  - $D \twoheadrightarrow C$

  BCNF, da die BCNF verletzende FD aus dem Beispiel für 3. NF entfernt wurde. Nicht 4. NF weil eine nicht triviale MVD gilt, deren linke Seite nicht Superschlüssel ist.

- 4. NF

  FDs:

  - $AB \to CD$
  - $BC \to AD$

  Nach Entfernung der nicht trivialen MVD dann auch 4. NF.

Hausaufgabe 2

Gegeben sei ein Array von 1.000.000.000 8-Byte-Integer-Werten und ein Programm, das alle Werte aufsummiert.

Das Programm wird auf einem System mit 16 GB Hauptspeicher und einer herkömmlichen Magnetfestplatte (Größe 1 TB), auf der alle Werte sequentiell gespeichert sind, ausgeführt. Ein Random Access auf die Festplatte dauert 10 ms, beim sequentiellen Lesen hat sie einen Durchsatz von 160 MB/s. Das Summieren zweier Werte im Hauptspeicher dauert 1 ns.

(1 MB = 10^6 B und 1 TB = 10^{12} B)

a) Gehen Sie davon aus, dass alle Werte bereits im Hauptspeicher liegen. Wie lange läuft das Programm?

b) Nun liegen alle Werte ausschließlich auf der Festplatte. Wie lange läuft das Programm jetzt?

c) Auf der Festplatte liegt jetzt zusätzlich nach jedem 100.000. Wert die Summe der 100.000 davorliegenden Werte. Wie lange läuft das Programm, wenn es nur diese Summen aufsummiert?

Lösung:

a) Jede Zahl wird auf eine laufende Gesamtsumme addiert. Insgesamt müssen also 10^9 Additionen ausgeführt werden. Bei einer Zeit von 1 ns pro Addition ergibt dies eine Gesamtlaufzeit von $10^9 \cdot 10^{-9}$ s = 1 s.

b) Da die Werte sequentiell auf der Festplatte liegen, muss der Lesekopf nicht jeden Wert einzeln ansteuern sondern nur einmal zum ersten Wert finden und dann die Daten sequentiell auslesen. Hier wird die Lesezeit also vom maximalen Durchsatz der Platte dominiert. Insgesamt ergibt sich also:

$$
\begin{aligned}
t_{total} &= t_{seek} + t_{read} \\
&= 10\ \text{ms} + \frac{10^9 \cdot 8\ \text{B}}{160 \cdot 10^6\ \frac{\text{B}}{s}} \\
&= 10\ \text{ms} + \frac{8 \cdot 10^9\ \text{B}}{16 \cdot 10^7\ \frac{\text{B}}{s}} \\
&= 10\ \text{ms} + 50\ \text{s} \\
&\approx 50\ \text{s}
\end{aligned}
$$

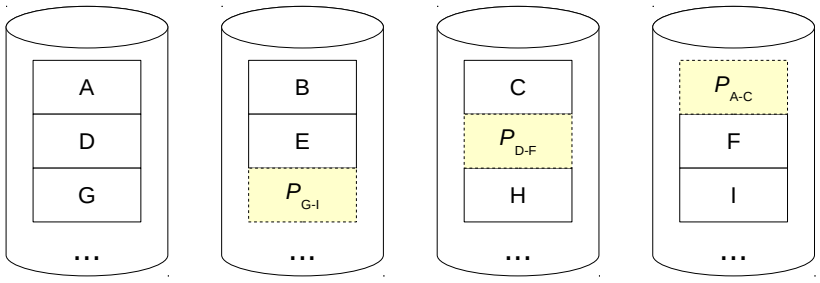
Dazu kommt noch die in Teilaufgabe a) berechnete Additionszeit selbst. Die Gesamtlaufzeit beträgt also 51 Sekunden.

c) Hier kann sich das Programm die Einzeladditionen sparen und muss lediglich die $10^9/10^5 = 10000$ Zwischenwerte addieren. Diese liegen nun allerdings nicht mehr sequentiell hintereinander, sondern $100000 \cdot 8\ \text{B} = 800\ \text{KB}$ auseinander. Das ist wesentlich größer als ein typischer Festplattensektor. Jeder Lesezugriff auf einen solchen Zwischenwert ist also ein Random Access. Wir erhalten so eine aufsummierte Zugriffszeit von $10000 \cdot 10\ \text{ms} = 100\ \text{s}$. Die Additionszeit mit $10000\ \text{ns} = 10\ \mu\text{s}$ und die Übertragungszeit mit $(8 \cdot 10000\ \text{B})/(160 \cdot 10^6\ \text{B/s}) = 0.5\ \text{ms}$ ist vernachlässigbar. Insgesamt läuft das Programm also ungefähr 100 s.

Trotz der „Optimierung“ hat diese Variante also eine längere(!) Laufzeit, als der „naive“ Ansatz aus Teilaufgabe b).

Hausaufgabe 3

Die folgende Abbildung zeigt einen Festplattenverbund bestehend aus vier Laufwerken, auf welchen die Datenblöcke A bis I gespeichert sind. Die Blöcke P_i enthalten Paritätsinformationen.



a) Um welches RAID-Level handelt es sich?

b) Wieviele Festplatten können ausfallen, ohne dass mit Datenverlust zu rechnen ist? Geben Sie eine allgemeine Lösung für einen Verbund bestehend aus n Festplatten an.

c) Kann die Ausfallsicherheit erhöht werden? Begründung?

d) Welchen weiteren Vorteil bietet das gezeigte RAID-System neben der Ausfallsicherheit?

e) Nach einem Festplattendefekt enthalten die Datenblöcke die folgenden Binärdaten. Rekonstruieren Sie die Datenblöcke der $Disk_2$ mithilfe der XOR-Verknüpfung.

$Disk_0$	$Disk_1$	$Disk_2$	$Disk_3$
A = 1111	B = 1001	C = _ _ _ _	$P_{\text{A-C}} = 1110$
D = 0101	E = 1100	$P_{\text{D-F}}$ = _ _ _ _	F = 1100
G = 0011	$P_{\text{G-I}} = 1110$	H = _ _ _ _	I = 0011

Lösung:

a) 5

b) 1, unabhängig von n.

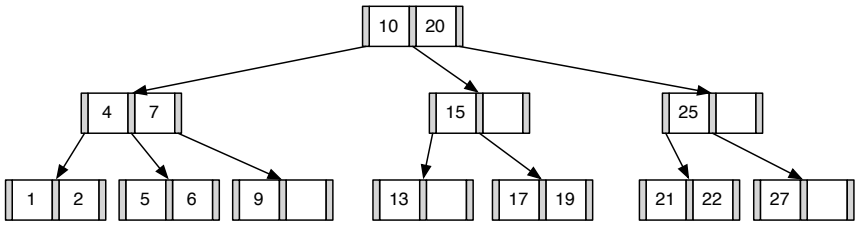
c) Ja, z.B. mit einem RAID-6 (Ausfall zweier Platten kann kompensiert werden) oder RAID-15 (das RAID-5 wird zusätzlich nochmal gespiegelt).

d) Höherer Datendurchsatz.

e) Die Rekonstruktion der Datenblöcke unterscheidet sich rechnerisch nicht von der Berechnung der Parität.

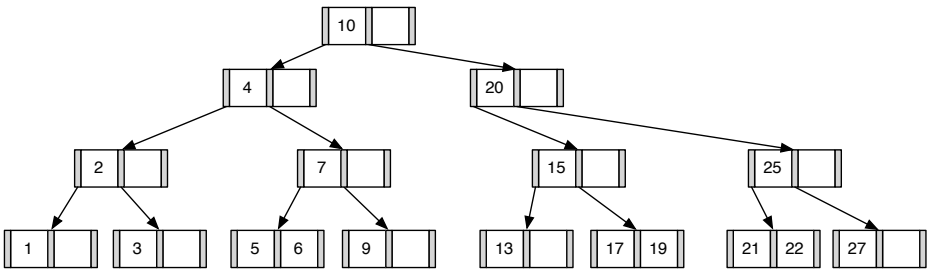
$Disk_0$	$Disk_1$	$Disk_2$	$Disk_3$
A = 1111	B = 1001	C = **1000**	$P_{\text{A-C}} = 1110$
D = 0101	E = 1100	$P_{\text{D-F}}$ = **0101**	F = 1100
G = 0011	$P_{\text{G-I}} = 1110$	H = **1110**	I = 0011

Hausaufgabe 4



1. Fügen Sie die 3 in den gezeigten B-Baum ein. Zeichnen Sie das Endergebnis. Zeichnen Sie jeweils den kompletten Baum oder machen Sie **deutlich**, falls Teile des Baumes unverändert bleiben. Verwenden Sie den aus der Vorlesung bekannten Algorithmus.

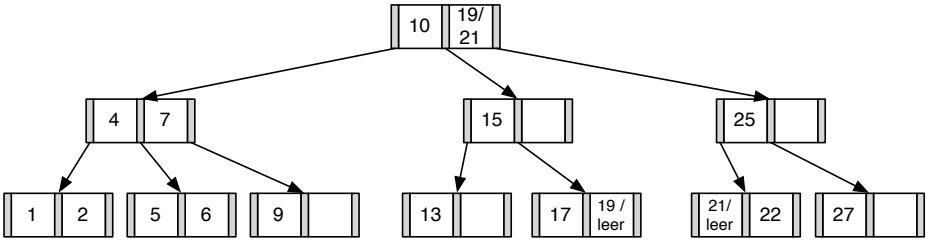
   Lösung: Das Ergebnis sieht wie folgt aus:



2. Entfernen Sie aus dem **ursprünglichen Baum** den Eintrag 20. Zeichnen Sie das Ergebnis der Operation. Sollte es mehrere richtige Lösungen geben, genügt es, wenn Sie hier eine angeben. Zeichnen Sie jeweils den kompletten Baum oder machen Sie

deutlich, falls Teile des Baumes unverändert bleiben. Verwenden Sie den aus der Vorlesung bekannten Algorithmus.

Lösung: Das Ergebnis sieht wie folgt aus:

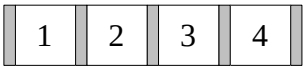


Hausaufgabe 5

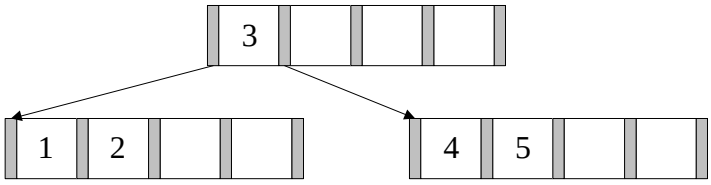
Fügen Sie in einen anfänglich leeren B-Baum mit $k = 2$ die Zahlen eins bis zwanzig in aufsteigender Reihenfolge ein. Was fällt Ihnen dabei auf?

Lösung:

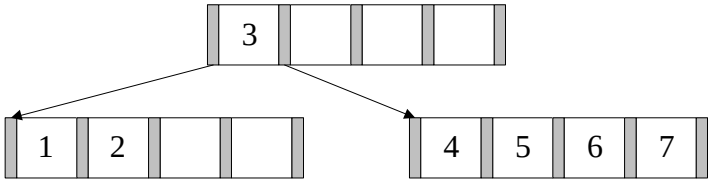
Nachdem man die Zahlen 1 bis 4 eingefügt hat, liegt folgender B-Baum vor:



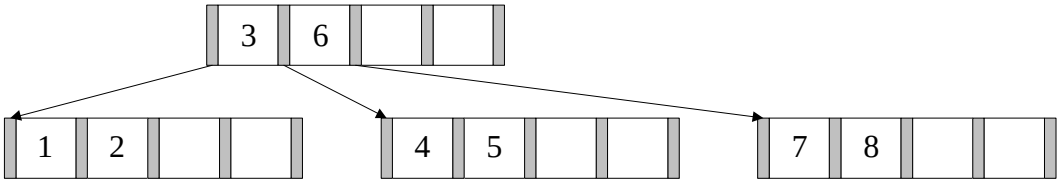
Beim Einfügen von 5 wird der Knoten gespalten und man erhält eine neue Wurzel.



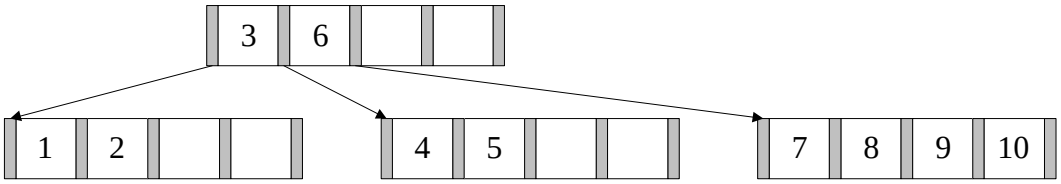
Die nächsten beiden Zahlen lassen sich wieder ohne Probleme einfügen.



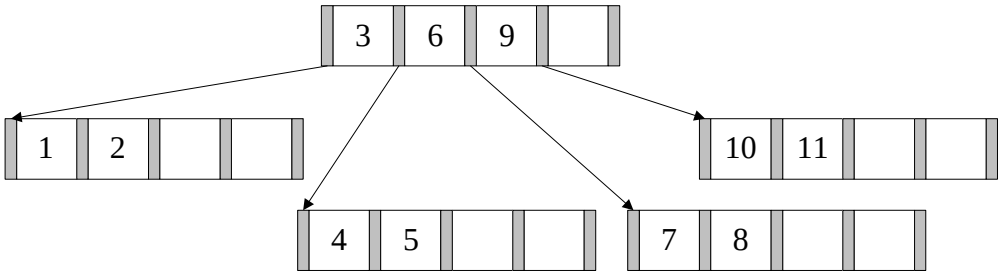
Beim Einfügen der 8 kommt es erneut zum Überlauf. Die 6 wandert in die Wurzel.



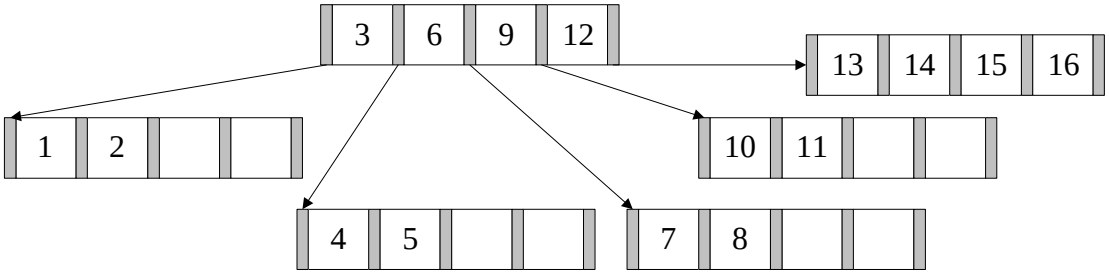
9 und 10 lassen sich wieder ohne Probleme einfügen. Bei 11 kommt es zum Überlauf.



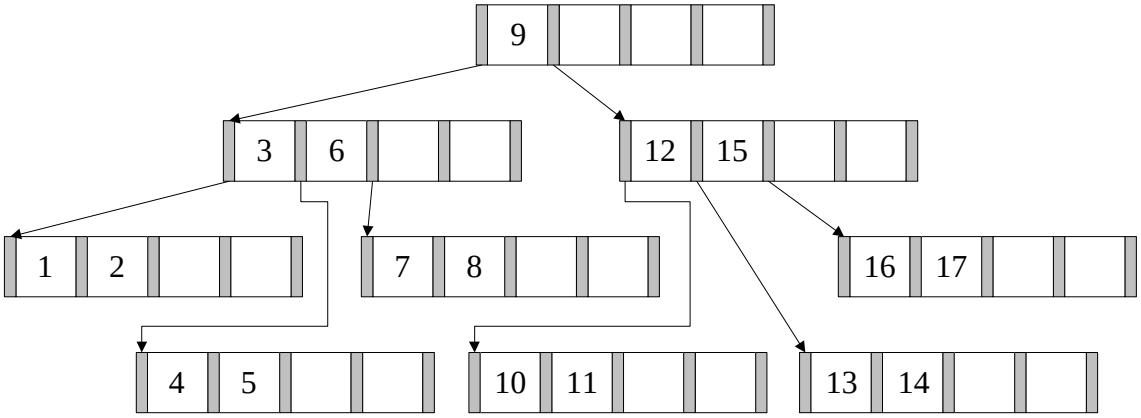
Nach dem Aufspalten erhält man dann:



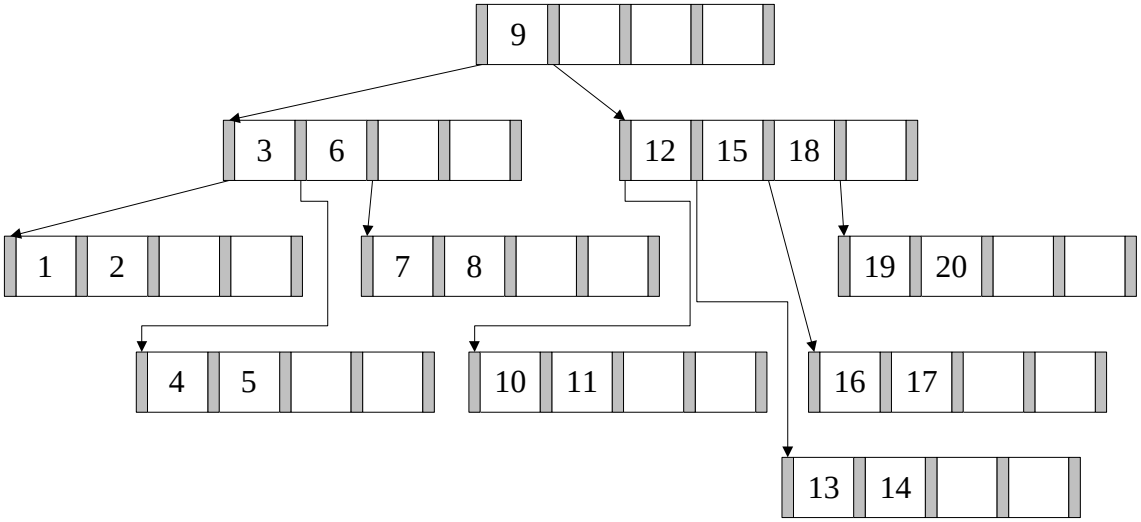
Es werden nun die nächsten Zahlen bis 16 analog eingefügt.



Bei 17 kommt es dann wieder zum Überlauf.



Fügt man nun noch die restlichen Zahlen ein, erhält man folgenden B-Baum:



Es fällt auf, dass der B-Baum nahezu minimale Auslastung aufweist. Dies liegt daran, dass eine aufsteigende Zahlenfolge sequentiell in den Baum eingefügt wird. Nach dem Aufspalten einer Seite in zwei Seiten werden dann in die Seite, die die kleineren Datensätze enthält, keine weiteren Werte mehr eingefügt. Allgemein ist das sortierte Einfügen der Schlüssel in einen B-Baum eine sehr schlechte Idee, da dies zu einer sehr geringen Auslastung führt.

Bonusaufgabe 6

Implementieren Sie einen B+-Baum in C++. Es sollten mindestens die Funktionen *insert* und *lookup* unterstützt werden. Zur Vereinfachung können Sie annehmen, dass lediglich Schlüssel-Wert-Paare bestehend aus Integern eingefügt werden.

Diese Aufgabe ist für diejenigen gedacht, die sich über den Vorlesungsstoff hinaus mit dem Thema Datenbanken beschäftigen wollen. Sie wird nicht in der Übung besprochen und ist nicht klausurrelevant. Falls Sie Feedback wünschen, können Sie Ihre Lösung gerne an `gdb@in.tum.de` *senden.*