

[illegible]

Aufgabe 1 - *Slotted Page*.1 Punkt

Gegeben sei eine Slotted Page mit folgenden Eigenschaften:

- Größe: 2^{16} Bytes
- Header: **dense** (dicht gepackt)
- Adressierungstyp: **Byte-Adressierung** (es kann jedes Byte adressiert werden)

Berechnen Sie die **Größen der Felder im Header** (a , f , g_i und p_i) und weiters **wieviele Datensätze der Größe 2^5 Bytes auf der Slotted Page gespeichert werden können**.

Aufgabe 2 - Erweiterbares Hashing.

1 Punkt

Die Hashfunktion h liefert die in der Tabelle angegebenen Binärwerte. **Ergänzen Sie die fehlenden Werte/Komponenten (globale/lokale Tiefen, Pointer).**

Hash-Tabelle:

x	$h(x)$
6	1100
13	1011
20	1000
21	1010
23	1110
28	1001
33	0010
34	0110

000
001
010
011
100
101
110
111

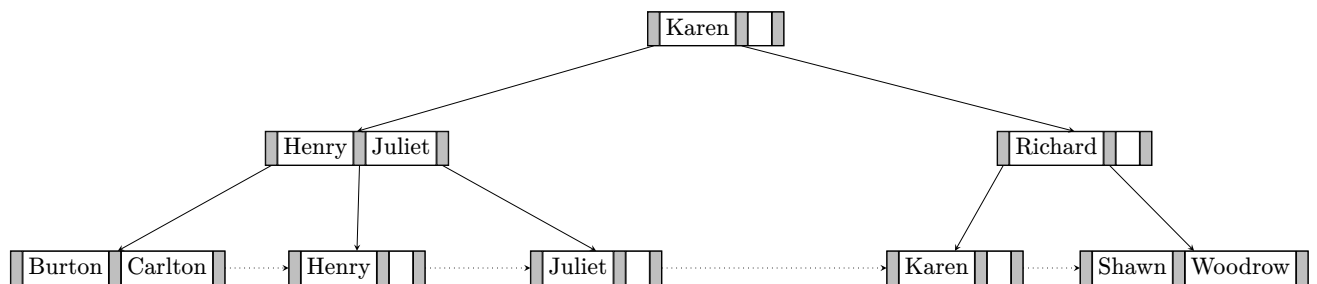
33
34
20
28
13
21
6
23

Aufgabe 3 - B^+ -Baum Löschen.

1 Punkt

Gegeben ist eine Relation $R[id, name]$ und der folgende B^+ -Baum-Index auf Attribut $name$ mit $m = 3$. Zeichnen Sie den B^+ -Baum, der nach der Ausführung der folgenden SQL-Anfrage entsteht. Führen Sie nur so viele Indexänderungen wie notwendig durch.

```
delete from R where name = 'Richard'
```



Aufgabe 4 - *Indexstrukturen*.1 Punkt

Zeichnen Sie für die folgende Tabelle einen **3-stufigen Sekundärindex** auf dem Attribut **Stadt**. Die **innere Indexstufe** soll **dense** und die **äußeren beiden Indexstufen** sollen **sparse** sein. In einen **Indexblock** können **3 Einträge** gespeichert werden.

Stadt	KFZ
Rom	I
London	GBM
Prag	CZ
Kiew	UA
Berlin	D
Athen	GR
Krakau	PL
Oslo	N
Dublin	IRL
Wien	A

Aufgabe 5 - Externes Merge-Sort.

1 Punkt

Betrachten Sie eine Relation $R[A]$ mit 2000 Tupel. Ein Block hat Platz für zwei Tupel. Der Puffer kann 10 Blöcke speichern.

Wieviele Blöcke müssen gelesen/geschrieben werden um einen externen Merge-Sort auf Relation R auszuführen? Zählen Sie dabei den letzten Schreib-Schritt, der das Ergebnis wieder auf die Festplatte schreibt, **nicht** mit.

Aufgabe 6 - *Join-Algorithmen*.1 Punkt

Gegeben seien **zwei Relationen** R und S mit folgenden Eigenschaften:

$R[A, B, C]$:

- $|R| = 10^7$ Tupel, gespeichert auf $b_R = 20 \cdot 10^3$ Datenblöcken
- **dense** B^+ -Baum-Index auf Attribut A , $m = 2^8$
- **sparse** B^+ -Baum-Index auf Attribut B , $m = 2^7$
- Die B^+ -Bäume besitzen **maximale Höhe**.

$S[B, D, F]$:

- $|S| = 5 \cdot 10^6$ Tupel, gespeichert auf $b_S = 4 \cdot 10^3$ Datenblöcken
- **einstufiger dense** Index (ISAM) auf Attribut B mit $5 \cdot 10^4$ Indexblöcken
- **einstufiger sparse** Index (ISAM) auf Attribut D mit 40 Indexblöcken

Es soll ein **natürlicher Join** $R \bowtie S$ mithilfe des **Index Nested Loop Joins** durchgeführt werden.

Geben Sie hierfür die effizienteste Join-Reihenfolge ($R \bowtie S$ oder $S \bowtie R$) sowie die zugehörigen **Kosten (in Blockzugriffen)** an. Ein Knotenzugriff im B^+ -Baum entspricht einem Blockzugriff. Duplikate können für diese Aufgabe vernachlässigt werden.

Aufgabe 7 - Effiziente Anfragebearbeitung.**1 Punkt**

Gegeben ist die Relation $R[A, B]$. Auf $R.A$ existiert ein **sparse B⁺-Baum Index** und auf $R.B$ existiert ein **dense Hash-Index**. Die Werte für Attribut B sind eindeutig. Was ist die **effizienteste Strategie** um Anfragen von folgendem Typ zu beantworten?

$$\sigma_{A < a \vee B = b}(R)$$

Geben Sie **alle notwendigen Schritte** an.

Aufgabe 8 - Anfrageoptimierung und Join-Reihenfolge.1 Punkt

Gegeben seien 3 Relationen $R[A, B, C]$, $S[A, D, E]$, $T[A, F, G]$ mit folgenden Eigenschaften:

- $|R| = 1.200$ Tupel, $V(R, A) = 50$, $V(R, B) = 100$, $V(R, C) = 200$
- $|S| = 3.000$ Tupel, $V(S, A) = 20$, $V(S, D) = 1.000$, $V(S, E) = 600$
- $|T| = 5.000$ Tupel, $V(T, A) = 100$, $V(T, F) = 1.200$, $V(T, G) = 1.800$

Weiters sei die folgende SQL-Anfrage gegeben:

```
select distinct R.A, S.D, T.G
from           R, S, T
where          R.A = S.A
              and R.A = T.A
```

- a. Zeichnen Sie die **algebraische Normalform als Operatorbaum** für die gegebene SQL-Anfrage.
- b. Wenden Sie **heuristische Optimierung** an, um den **Operatorbaum zu optimieren**. Im resultierenden Operatorbaum soll die **Join-Reihenfolge optimal** sein (d.h. es soll zuerst der Join mit dem kleinsten Zwischenergebnis durchgeführt werden).

Aufgabe 9

1 Punkt

Ist der folgende Schedule **konflikt-serialisierbar**? Zeichnen Sie den Präzedenzgraphen, um Ihr Ergebnis zu überprüfen und begründen Sie. Geben Sie einen äquivalente serielle Schedule an, falls die Schedule konflikt-serialisierbar ist.

T1:	T2:	T3:
	write(C)	
	read(A)	
		write(C)
read(B)	write(A)	
	read(A)	read(B)
write(B)		
read(C)		write(B)
		read(A)

Aufgabe 101 Punkt

Kann der folgende Schedule das Ergebnis eines **Two-Phase-Locking**-Schedulers sein? Falls ja, geben Sie alle notwendigen Lock/Unlock-Operationen an. Erklären Sie ansonsten, warum diese Historie unmöglich ist. Könnte der Schedule das Ergebnis eines **Strict-Two-Phase-Locking**-Schedulers sein? Warum (nicht)?

T1: T2: T3:

read(A)

write(A)

 read(B)

 write(C)

 read(A)

 COMMIT

 read(C)

 write(C)

 COMMIT

read(B)

COMMIT
