Fachbereich für Computerwissenschaften

Prof. Dr. Nikolaus Augsten
Jakob-Haringer-Str. 2
5020 Salzburg, Austria
Telefon: +43 662 8044 6347
E-Mail: nikolaus.augsten@sbg.ac.at

UNIVERSITÄT SALZBURG

Datenbanken II
Wintersemester 2019/2020

Prüfung
03.02.2020

Name: _____ **Matrikelnummer:** _____

Hinweise

- Bitte überprüfen Sie die Vollständigkeit des Prüfungsbogens (11 nummerierte Seiten).
- Schreiben Sie Ihren Namen und Ihre Matrikelnummer auf jedes Blatt des Prüfungsbogens und geben Sie alle Blätter ab.
- Grundsätzlich sollten Sie alle Antworten auf den Prüfungsbogen schreiben.
- Sollten Sie mehr Platz für eine Antwort benötigen, bitte einen klaren Verweis neben die Frage auf die Seitennummer des zusätzlichen Blattes setzen.
- Keinen Bleistift verwenden. Keinen roten Stift verwenden.
- Verwenden Sie die Notation und die Lösungsansätze, die während der Vorlesung besprochen wurden.
- Aufgaben mit mehr als einer Lösung werden nicht bewertet.
- Als Unterlage ist ein beliebig (auch beidseitig) beschriftetes A4-Blatt erlaubt.
- Zeit für die Prüfung: 90 Minuten

Unterschrift _____

Korrekturabschnitt — Bitte frei lassen

Aufgabe	1	2	3	4	5	6	7	8	9	10	Summe
Maximale Punkte	1	1	1	1	1	1	1	1	1	1	10
Erreichte Punkte											

Aufgabe 1 - *Indexstrukturen*. 1 Punkt

Zeichnen Sie für die folgende Tabelle einen **3-stufigen Sekundärindex** auf dem Attribut `Stadt`. Die **erste Indexstufe** soll **dense** und die **zweite und dritte Indexstufe** 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 2 - *Slotted Page*. 1 Punkt

Gegeben sei eine Slotted Page mit folgenden Eigenschaften:

- Größe: $2^{11} = 2048$ Bytes,
- **Byte-Adressierung**: es kann jedes Byte adressiert werden,
- **Dense Header**: es wird die minimale Anzahl von Bits benutzt um die Header-Felder zu speichern.

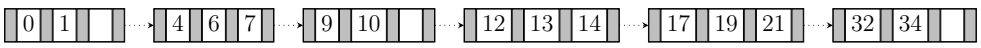
In dieser Slotted Page werden **3 Tupel** R, S, T gespeichert:

- d_1: $|R| = 9$ Bytes
- d_2: $|S| = 110$ Bytes
- d_3: $|T| = 245$ Bytes

Was ist die Größe vom Free Space (Anzahl in ganzen Bytes)?

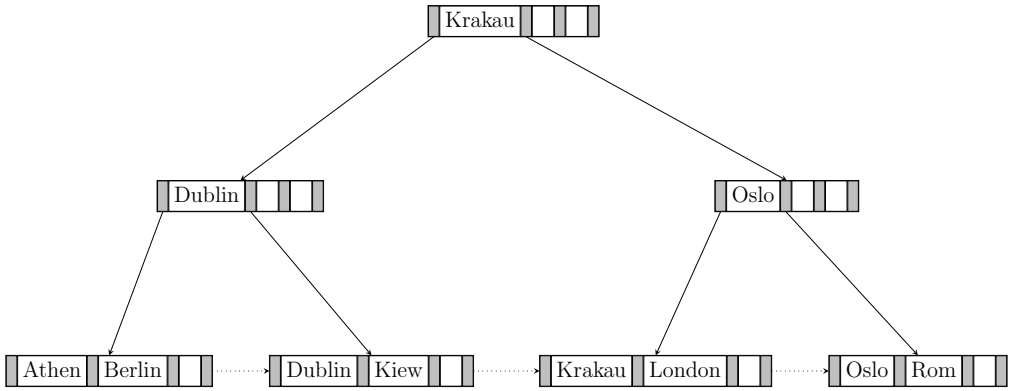
Aufgabe 3 - *B⁺-Baum-Konstruktion.* 1 Punkt

Gegeben sind die Blätter eines B$^+$-Baumes ($m = 4$). Konstruieren Sie die **darüberliegenden Ebenen** (d.h. die inneren Knoten) so, dass ein **gültiger B$^+$-Baum mit minimaler Höhe** entsteht.



Aufgabe 4 - *B⁺-Baum-Einfügen.* 1 Punkt

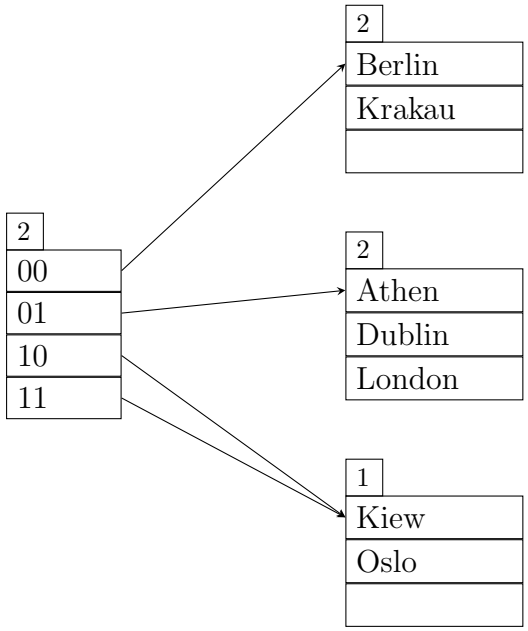
Gegeben ist ein B$^+$-Baum mit $m = 4$. Zeichnen Sie den B$^+$-Baum, der nach dem Löschen von **Krakau** entsteht.



Aufgabe 5 - *Erweiterbares Hashing*. 1 Punkt

Die Hashfunktion $h(x)$ liefert die in der Tabelle angegebenen Binärwerte. Es soll das Tupel **Rom** in den gegebenen Hashcontainer eingefügt werden. Ein Bucket im Hashcontainer kann bis zu 3 Tupel speichern. Das Verzeichnis soll so klein wie möglich gehalten werden. **Illustrieren Sie den resultierenden Hashcontainer**.

x	$h(x)$
Athen	0100
Berlin	0010
Dublin	0100
Kiew	1100
Krakau	0011
London	0110
Oslo	1110
Rom	0101



Aufgabe 6 - *Externes Merge-Sort*. 1 Punkt

Führen Sie **externes Merge-Sort** auf der folgenden Relation $R[A]$ aus.
Jeder **Block fasst 2 Tupel**. Die Größe des **Puffer** beträgt **4 Blöcke**.

11
25
10
0
13
9
7
22
18
5
16
3
24
20
12
8
6
2
19
14
4
21
23
17
15
1

Aufgabe 7 - *Effiziente Anfragebearbeitung.* 1 Punkt

Gegeben sei eine Relation $R[A, B]$ mit folgenden Eigenschaften:

- $|R| = 10^7$ Tupel,
- Pro Datenblock werden 32 Tupel gespeichert,
- Attribut A hat ganzzahlige Werte gleichverteilt im Bereich $[1; 4 \cdot 10^4]$,
- Attribut B hat ganzzahlige Werte gleichverteilt im Bereich $[1; 3 \cdot 10^4]$,
- Duplikate werden mittels Tuple Identifier (TID) aufgelöst,
- Folgende Indizes existieren:
  - Clustered B^+-Baum-Index auf Attribut A, $m = 64$
  - Dense B^+-Baum-Index auf Attribut B, $m = 64$

Es soll folgende Anfrage beantwortet werden:

$$\sigma_{A<8000 \wedge B>15000}(R)$$

Geben Sie die **Strategie (0.5 Punkte)** an und berechnen Sie die **Anzahl der Blockzugriffe (0.5 Punkte)** um die Anfrage **möglichst effizient** zu beantworten (1 Knotenzugriff im B^+-Baum entspricht 1 Blockzugriff).

Aufgabe 8 - *Join-Algorithmen.* 1 Punkt

Welcher Join Algorithmus (**Merge Join vs. Index Nested Loop Join**) generiert die minimalen Kosten für das folgende Szenario? Geben Sie in der Lösung die **Algorithmen** und die **dazugehörigen** Kosten an.

Berechnen Sie einen **Natural Join** zwischen zwei Relationen $R[A, B]$ und $S[A, C]$, wobei $|R| = 50$ Tupel und $|S| = 5000$ Tupel. Die Relationen sind auf $b_R = 10$ bzw. $b_S = 2000$ hintereinander liegenden Blöcken gespeichert. Es existiert ein sparse B^+-Baum Index auf $S.A$ mit $m = 8$. Auf $R.A$ existiert kein Index, falls Sie eine Sortierung benötigen, nehmen Sie an dass diese $\lceil b_R \cdot \log_2(b_R) \rceil$ viele Blockzugriffe zusätzlich kostet.

Aufgabe 9 - *Index Wahl.* 1 Punkt

Gegeben sei eine Relation $R[A, B, C, D, E, F, G]$ mit 2^{30} Tupel. Die Attribut-Werte sind gleichverteilt. Die folgende Anfrage wird häufig an die Datenbank gestellt.

$$\pi_A(\sigma_{B<x \wedge B>y \wedge E=z \wedge G \neq k}(R))$$

Voraussetzung: Sie können nur **einen** Index erstellen, um die Anfrage effizient zu beantworten.

- **(0.5 Punkt)** Markieren Sie das Attribut/die Attribute und den Index-Typ mit "X" in der folgende Tabelle.
- **(0.5 Punkt)** Begründen Sie Ihre Antwort.

Index-Typ	A	B	C	D	E	F	G
B^+-Baum							
Hash							

Aufgabe 10 - *Anfrageoptimierung, Join-Reihenfolge.* 1 Punkt

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

- $|R| = 2000$ Tupel, $V(R, A) = 100$, $V(R, C) = 20$, $V(R, D) = 80$
- $|S| = 300$ Tupel, $V(S, C) = 200$, $V(S, E) = 50$
- $|T| = 500$ Tupel, $V(T, B) = 20$, $V(T, C) = 50$, $V(T, F) = 30$

Weiters sei folgende SQL-Anfrage gegeben:

```
SELECT R.A, S.C, T.B
FROM   R, S, T
WHERE  R.C = S.C
       AND S.C = T.C
```

a. Zeichnen Sie die **algebraische Normalform als Operatorbaum** für die gegebene SQL-Anfrage. **(0.5 Punkte)**
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). **(0.5 Punkte)**