Fachbereich Informatik

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

PARIS LODRON UNIVERSITÄT SALZBURG

Datenbanken II — Prüfung
Wintersemester 2024/25 — 29.01.2025

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 - *Slotted Page*. 1 Punkt

Gegeben sei eine Slotted Page mit folgenden Eigenschaften:

- Größe: $2^{13} = 8192$ Bytes
- Adressierungstyp: **Word-Adressierung** (es kann nur jedes 2. Byte adressiert werden)

In dieser Slotted Page werden die Tupel O, P, Q gespeichert:

- d_1: $|O| = 64$ Bytes
- d_2: $|P| = 127$ Bytes
- d_3: $|Q| = 255$ Bytes

Ergänzen Sie die Slotted Page um die **fehlenden Werte/Adressen** (numerische Werte erwartet, Pfeile reichen nicht aus), wobei p_i und g_i sich auf den jeweiligen Datensatz d_i beziehen.

a	f	g_1	p_1	g_2	p_2	g_3	p_3	...	d_3	d_2	d_1
									Q	P	O

Aufgabe 2 - *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

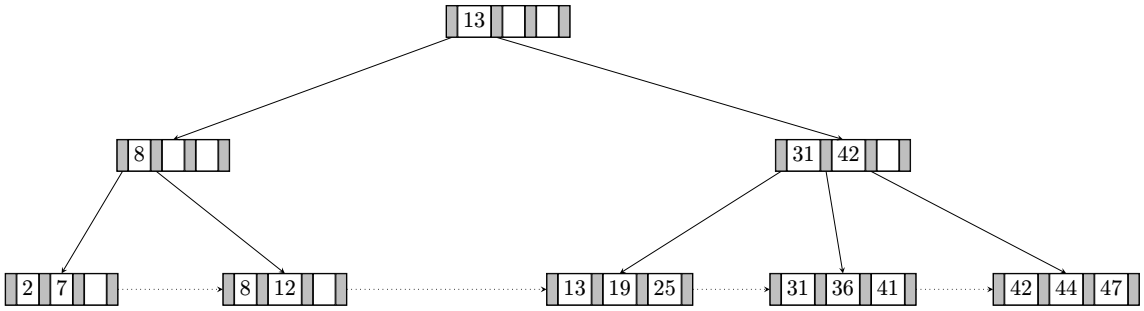
Aufgabe 3 - *Statisches Hashing*. 1 Punkt

Auf der folgenden Tabelle soll ein **Hash Index** konstruiert werden. Als Schlüssel wird das Attribut *Account Nr* verwendet. Der Hashwert ist die **erste Ziffer** des Attributwerts mod 5. Es gibt **5 Buckets** und es können **3 Tupel pro Bucket** gespeichert werden. Bucket Overflows werden durch **Overflow Chaining** aufgelöst, wobei ein Zeiger auf ein Overflow Bucket einen Eintrag im Bucket benötigt. **Illustrieren** Sie den **Hash Index**.

Owner Name	*Account Nr*	*Balance*
Donovan	579976	2.467
Kermit	585989	7.824
Solomon	489384	6.824
Gavin	579331	3.850
Kelly	630468	8.949
Angelica	676246	6.452
Fredericka	589374	8.888
Caesar	682535	2.776
Chanda	304225	2.014
Patricia	886712	7.726

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

Gegeben ist ein B$^+$-Baum mit $m = 4$. Zeichnen Sie den B$^+$-Baum, der nach dem Löschen von **8** entsteht.



Aufgabe 5 - *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 **3 Blöcke**.

90
60
10
0
40
30
45
20
25
5
50
35
45
15
65
100
95
80
85
70

Aufgabe 6 - *Join-Algorithmen*. 1 Punkt

Gegeben seien **zwei Relationen** R and 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 sei eine Relation $R[A, B, C]$ mit folgenden Eigenschaften:

- $|R| = 2.000.000$ Tupel.
- Pro Datenblock werden 400 Tupel gespeichert.
- Attribut A hat ganzzahlige Werte gleichverteilt im Bereich $[1; 2.000.000]$.
- Attribut B hat ganzzahlige Werte gleichverteilt im Bereich $[400.001; 500.000]$.
- Attribut C hat ganzzahlige Werte gleichverteilt im Bereich $[100.001; 1.000.000]$.
- Es existieren folgende Indizes:
  - Sparse B^+-Baum-Index auf Attribut A, $m = 2^8 = 256$, minimale Höhe,
  - dense B^+-Baum-Index auf Attribut B, $m = 2^8 = 256$, minimale Höhe,
  - dense B^+-Baum-Index auf Attribut C, $m = 2^8 = 256$, minimale Höhe.

Es soll folgende Anfrage beantwortet werden:

$$\sigma_{A>1.800.000 \wedge B=450.000}(R)$$

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 - *Anfrageoptimierung*. 1 Punkt

Betrachte die folgenden Relationen:

(B)oats(bid, name, color)
(S)ailors(sid, name, rating, age)
(R)eservations(bid, sid, day)

Weiters sei die folgende SQL-Anfrage gegeben:

```
SELECT DISTINCT B.name
  FROM Boats B, Sailors S, Reservations R
 WHERE S.age < 40
   AND B.color = 'blue'
   AND B.bid = R.bid
   AND S.sid = R.sid;
```

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**. **(0.5 Punkte)**

Aufgabe 9 — 1 Punkt

Geben Sie eine Historie (Schedule) von zumindest zwei Transaktionen an, die **konfliktserialisierbar**, aber nicht **rücksetzbar** (recoverable) ist. Beschreiben Sie, wie Ihre Historie zu einem inkonsistenten Datenbankzustand führen kann.

Aufgabe 10 — 1 Punkt

Betrachten Sie folgende Schedule und Two-Phase-Locking.

(a) Nach der `read` Operation von `T3` wollen alle Transaktionen eine `write` Operation auf Element `A` in folgender Reihenfolge durchführen: `T1`, `T2`, `T3`. Vervollständigen Sie die Schedule, indem Sie Lock-Anfragen (request; z.B. `R:lock-S(A)`) und erteilte Locks (granted; z.B. `G:lock-S(A)`) für alle verbleibenden `read` und `write` Operationen einfügen. Kennzeichnen Sie in der Schedule, wenn eine Transaktion warten muss.

(b) Zeichnen Sie den Wait-for Graph. Führt die Schedule zu einem Deadlock?

```
T1:         T2:         T3:
_____________________________________
R:lock-S(A)
G:lock-S(A)
read(A)
_____________________________________

                        read(A)
_____________________________________

_____________________________________

_____________________________________

_____________________________________
```