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 2025/26 — 25.02.2026

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.
- **Multiple-Choice:** Falsche Antworten führen zu Punktverlust!
- 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 - *Pufferverwaltung*. 1 Punkt

Ein System besitzt einen (initial leeren) Puffer, in den **4** Seiten passen. Als Ersetzstrategie wird **MRU** (most recently used) eingesetzt. Es wird auf die Seiten A,B,C,D,E in folgender Reihenfolge zugegriffen.

E A A D C E B E B A E B C A

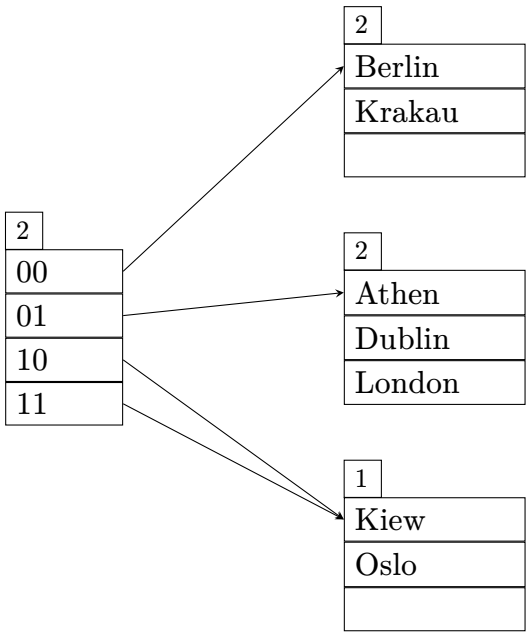
Wie viele Plattenzugriffe finden tatsächlich statt? Welche Blöcke befinden sich nach der Ausführung der Zugriffe im Puffer? Vervollständigen Sie dafür die folgende Tabelle. Puffertreffer werden mit "*" markiert.

Pufferseite 1	E												
Pufferseite 2		A	*										
Pufferseite 3				D									
Pufferseite 4													

Aufgabe 2 - *Erweiterbares Hashing*. 1 Punkt

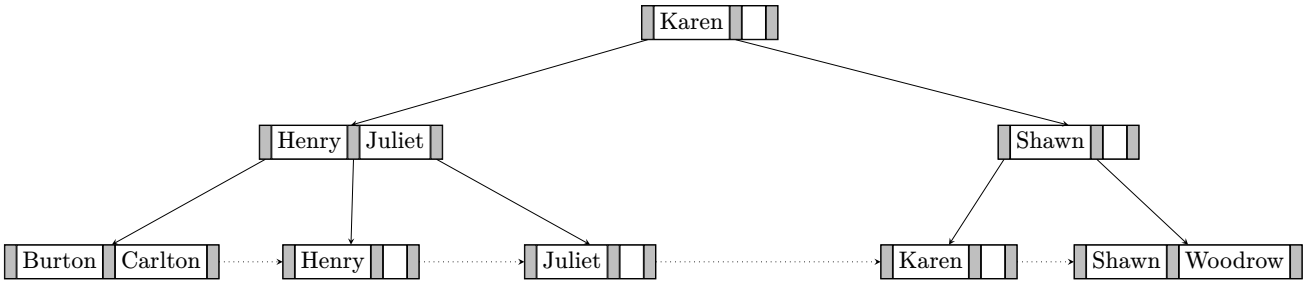
Die Hashfunktion $h(x)$ liefert die in der Tabelle angegebenen Binärwerte. Es soll das Tupel **Rom** aus dem gegebenen Hashcontainer eingefügt werden. Ein Bucket im Hashcontainer kann bis zu 3 Tupel speichern. Das Verzeichnis soll so klein wie möglich gehalten werden. **Illustrieren Sie den resultierenden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

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



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

Gegeben ist ein B⁺-Baum mit $m = 3$. Zeichnen Sie den B⁺-Baum, der nach dem Löschen von **Karen** entsteht.



Aufgabe 4 - *Indexstrukturen mit kombiniertem Suchschlüssel.* 1 Punkt

Betrachten Sie eine Relation $R[A, B, C, D]$ und die folgenden vier Anfrageprädikate. Geben Sie die Attributordnung von (bis zu) zwei geordneten Indizes mit kombiniertem Suchschlüssel an, sodass alle vier Anfragen mithilfe dieser Indizes effizient ausgeführt werden können.

(1) **WHERE** $B = 42$ **AND** $C < 100$

(2) **WHERE** $A < 50$ **AND** $B = 84$

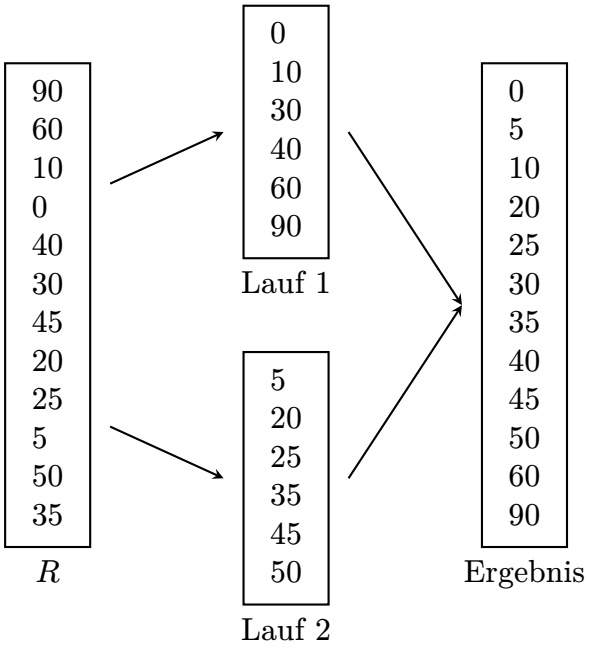
(3) **WHERE** $B = 42$ **AND** $C < 100$ **AND** $A = 100$

(4) **WHERE** $C = 100$ **AND** $D > 50$ **AND** $B = 42$

Aufgabe 5 - *Externes Merge-Sort*. 1 Punkt

Betrachten Sie die **Merge-Phase** des **externen Merge-Sort-Algorithmus** auf der folgenden Relation $R[A]$ mit 2 Läufen. Jeder **Block fasst 2 Tupel**. Die Größe des **Puffers** beträgt **3 Blöcke**.

Zeigen Sie alle **relevanten Zustände des Puffers.** Ein Puffer-Zustand ist relevant, wenn sich ein Block im Puffer **vollständig ändert** (d.h. alle Tupel ändern sich). Für den Ausgabe-Block des Puffers nehmen wir an, dass die Tupel im Ausgabe-Block nach dem Schreiben auf die Festplatte überschrieben werden.



$\mathbf{In_1}$	$\mathbf{In_2}$	**Out**
0 10	5 20	0 5

Aufgabe 6 - *Join-Algorithmen*. 1 Punkt

Die Relation $R[\underline{A, B}, C]$ hat $n_R = 3000$ Tupel auf $b_R = 400$ Blöcken und die Relation $S[\underline{B}, D]$ hat $n_S = 2000$ Tupel auf $b_S = 450$ Blöcken.

Es stehen $M = 30$ Blöcke im Puffer zur Verfügung. Es soll ein natürlicher Join mittels eines Hash-Join-Algorithmus auf die Relationen durchgeführt werden.

Beantworten Sie die folgenden Fragen und begründen Sie.

- Welche Relation wird als Probe-Input verwendet, welche als Build-Input?
- Was ist die minimale Anzahl von Partitionen, die erstellt werden müssen?
- Schätzen Sie ab, wieviele Block-Zugriffe ein Hash Join braucht.

Aufgabe 7 - *Effiziente Anfragebearbeitung.* 1 Punkt

Gegeben ist die Relation $R[A]$. Auf $R.A$ existiert ein sparse B^+-Baum Index. Was ist die **effizienteste Strategie** um Bereichsanfragen von folgendem Typ zu beantworten?

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

Geben Sie **alle notwendigen Schritte** an.

Aufgabe 8 - *Anfrageoptimierung und Join-Reihenfolge.* 1 Punkt

Gegeben seien 3 Relationen $R[A, C, D]$, $S[C, E]$, $T[B, C, F]$ 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 die folgende SQL-Anfrage gegeben:

```
SELECT  DISTINCT 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.
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 **alle** äquivalenten seriellen Schedule an, falls die Schedule konflikt-serialisierbar ist.

T1:	T2:	T3:	T4:
		read(A)	
read(A)			
	write(C)		
read(B)			
write(A)			
			read(C)
			write(B)
	read(A)		
	write(A)		
		read(B)	

Aufgabe 10 — 1 Punkt

Kann die 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 die 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		