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 2017/2018

Prüfung
08.02.2018

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 - *Festplattenparameter.* 1 Punkt

Betrachten Sie die folgende Festplatte:

- Sektorgröße $B = 2^9$B,
- Sektoren/Spur $S = 2^8$,
- Spuren pro Scheibenseite $T = 300$,
- Anzahl der beidseitig beschriebenen Scheiben $D = 2^3$,
- mittlerer Spurwechsel $sp = \frac{1}{480}$s,
- Zugriffszeit pro Sektor für einen (lesenden) Random Access: $\frac{193}{30720}$ s,

Berechnen Sie die **Umdrehungsgeschwindigkeit (in rpm)** der Festplatte.

Lösung: _____

Aufgabe 2 - *Hash-Datei-Organisation.* 1 Punkt

Die folgende Tabelle soll in eine Hash-Datei überführt werden. Als Schlüssel wird das Attribut *Account Number* verwendet. Der Hashwert ist die **letzte Ziffer** des Attributwerts. 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 die **Hash-Datei**.

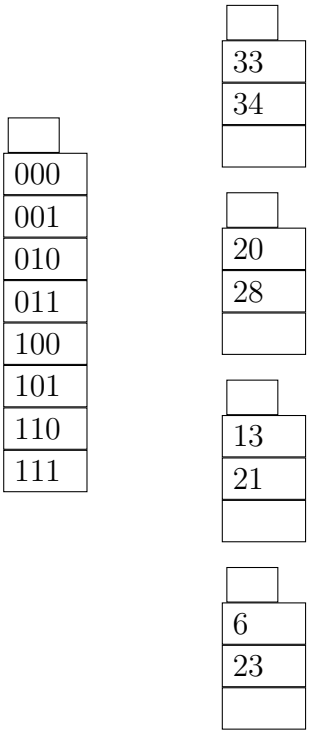
Owner Name	*Account Nr*	*Balance*
Donovan	579976	2.467
Kermit	585989	7.824
Solomon	589384	6.824
Gavin	579331	3.850
Kelly	630468	8.949
Angelica	676246	6.452
Fredericka	589374	8.888
Caesar	682535	2.776
Chanda	304225	2.014
Patricia	886712	7.726

Aufgabe 3 - *Erweiterbares Hashing*. 1 Punkt

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

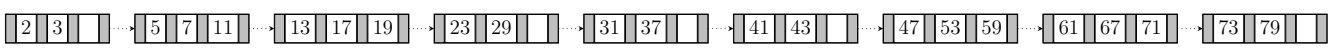
Hash-Tabelle:

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



Aufgabe 4 - B^+ *Baum.* 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 5 - *Statischer Hash-Index.* 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. 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 6 - *Externes Merge-Sort*. 1 Punkt

Gegeben sei eine Relation $R[A]$:

- $|R| = 1.000$ Tupel.
- Pro Datenblock sind 25 Tupel gespeichert.

Diese Relation soll per externem Merge-Sort sortiert werden.

Geben Sie die **minimale Puffergröße M in Blöcken** an, damit nur ein einziger Merge-Schritt nötig ist um $R[A]$ zu sortieren.

Aufgabe 7 - *Effiziente Anfragebearbeitung*. 1 Punkt

Gegeben sei eine Relation $R[A, B, C, D]$ 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 $[100.001; 1.000.000]$.
- Attribut D hat ganzzahlige Werte gleichverteilt im Bereich $[1; 10.000.000]$.
- Es existieren folgende Indizes:
  - sparse B^+-Baum-Index auf Attribut A.
  - dense B^+-Baum-Index auf Attribut B.
  - dense B^+-Baum-Index auf Attribut D.
- Für alle B^+-Baum-Indizes gilt:
  - $m = 2^8 = 256$.
  - Sie besitzen minimale Höhe.
  - 1 Knotenzugriff entspricht 1 Blockzugriff.

Es soll folgende Anfrage beantwortet werden:

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

Geben Sie die **Strategie und die Blockzugriffe** an, um die Anfrage **möglichst effizient** zu beantworten.

Aufgabe 8 - *Bitmap Index Scan.* 1 Punkt

Gegeben ist die Relation $R[A, B]$ mit $|R| = 10.000.000$, verteilt auf 8.000 Blöcke. Auf A existiert ein non-clustered B^+-Baum Index mit $m = 2.048$. Duplikate werden mittels Tuple Identifier (TID) aufgelöst. Die Knoten des B^+-Baumes sind nur minimal gefüllt.

Es soll folgende Query ausgeführt werden:

$$\sigma_{A=500}(R)$$

Das Attribut A hat insgesamt 80.000 mal den Wert 500, verteilt auf 1200 Blöcke. Berechnen Sie die **Zahl der Blockzugriffe**, um diese **Anfrage möglichst effizient**

1. **unter** Anwendung eines Bitmap Index Scans **(0.5 Punkte)**
2. **ohne** Anwendung eines Bitmap Index Scans **(0.5 Punkte)**

zu beantworten.

Aufgabe 9 - *Join-Algorithmen.* 1 Punkt

Gegeben seien folgende Relationen $R[A, B, C]$ und $S[B, E, F]$:

- $|R| = 50.000.000$ Tupel gespeichert auf $b_R = 125.000$ Blöcken.
- $|S| = 20.000.000$ Tupel gespeichert auf $b_S = 250.000$ Blöcken.

Es existiert auf $R[B]$ ein dense B^+-Baum-Index mit Knotengrad $m = 2^7 = 128$.
Weiters steht ein Puffer der Größe $M = 251$ zur Verfügung.

Es soll der natürliche Join $R \bowtie S$ durchgeführt werden.

Zur Auswahl stehen folgende Join-Algorithmen:

- Block-Nested-Loop-Join (BNLJ) mit Zick-Zack-Modus
- Indexed-Nested-Loop-Join (INLJ)

Wählen Sie den **effizientesten Join-Algorithmus** um den gegebenen Join durchzuführen und berechnen Sie die **Blockzugriffe** für den gewählten Join-Algorithmus.

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

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

- $|R| = 1.200$ 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  R.A, S.D, T.G
FROM R, S, T
WHERE R.A = S.A
AND R.A = T.A
```

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

2. Wenden Sie die heuristische Optimierung an, um den Operatorbaum zu optimieren. **Es soll auch die Join-Reihenfolge optimiert werden! (0.5 Punkte)**