Lehrstuhl für Datenbanksysteme
Fakultät für Informatik
Technische Universität München

Persönlicher Sticker

S5160

Bestätigung der Verhaltensregeln
Hiermit versichere ich, dass ich diese Klausur ausschließlich unter Verwendung der unten aufgeführten Hilfsmittel selbst löse und unter meinem Namen abgebe.

Andreas Binder

Unterschrift oder vollständiger Name, falls keine Stifteingabe verfügbar

Einsatz und Realisierung von Datenbanksystemen

Klausur:	IN2031 / Nachholklausur	**Datum:**	Dienstag, 29. September 2020
Prüfer:	Prof. Dr. Alfons Kemper	**Uhrzeit:**	08:00 – 09:30

Bearbeitungshinweise

- Diese Klausur umfasst **12 Seiten** mit insgesamt **10 Aufgaben**.
  Bitte kontrollieren Sie jetzt, dass Sie eine vollständige Angabe erhalten haben.
- Die Gesamtpunktzahl in dieser Prüfung beträgt 90 Punkte.
- Das Heraustrennen von Seiten aus der Prüfung ist untersagt.
- Als Hilfsmittel sind zugelassen:
  - alle Vorlesungsmaterialien („Open-Book“)
  - ein **analoges Wörterbuch** Deutsch ↔ Muttersprache **ohne Anmerkungen**
- Mit * gekennzeichnete Teilaufgaben sind ohne Kenntnis der Ergebnisse vorheriger Teilaufgaben lösbar.
- **Es werden nur solche Ergebnisse gewertet, bei denen der Lösungsweg erkennbar ist.** Auch Textaufgaben sind **grundsätzlich zu begründen**, sofern es in der jeweiligen Teilaufgabe nicht ausdrücklich anders vermerkt ist.
- Schreiben Sie weder mit roter / grüner Farbe noch mit Bleistift.
- Schalten Sie alle mitgeführten elektronischen Geräte vollständig aus, verstauen Sie diese in Ihrer Tasche und verschließen Sie diese.

Hörsaal verlassen von _____ bis _____ / Vorzeitige Abgabe um _____

Aufgabe 1 Fehlerbehandlung (10 Punkte)

Sie verwenden ein Datenbanksystem mit Write-Ahead-Logging und der Strategie ¬*force* und *steal*. Die Datenbank verwaltet zwei Datenobjekte A und B mit einem Anfangswert von jeweils 1 ($A = 1$ und $B = 1$). Sie starten zwei Transaktionen T_1 und T_2 zum gleichen Zeitpunkt:

T_1	T_2
BOT	**BOT**
$r(A, a_1)$	$r(B, b_2)$
$r(B, b_1)$	$r(A, a_2)$
$a_1 := a_1 + 1$	$b_2 := b_2 \cdot 10$
$b_1 := b_1 + 1$	$a_2 := a_2 \cdot 5$
$w(A, a_1)$	$w(B, b_2)$
$w(B, a_2)$	$w(A, a_2)$
COMMIT	**COMMIT**

Während der Ausführung stürzt Ihre Datenbank ab. Sie wissen nicht, welche Transaktionen erfolgreich ausgeführt worden sind. Nehmen Sie an, die Datenbank erzeugt ausschließlich *serielle Historien*. Bevor Sie die Datenbank neu starten, durchsuchen Sie die Festplatte und stellen fest, dass B dort den Wert 20 hat, A den Wert 2. Auch nach einem Neustart mit erfolgreichem Wiederherstellungsprozess liefert die Datenbank für A den Wert 2.

0 1 2

a)* Welche der Transaktionen hat erfolgreich committed (*Winner*), welche nicht (*Loser*)?

T1 Winner, T2 Loser

0 1 2 3 4 5 6

b) Geben Sie das Log in *logischer* Protokollierung an, wie es zum Zeitpunkt des Absturzes auf der Platte stand (A liegt auf Seite P_A und B auf Seite P_B). Geben Sie nur die Log-Einträge an, die sicher geschrieben wurden.
Hinweis: Form eines Log-Eintrages: [LSN,TA,PageID,Redo,Undo,PrevLSN] bzw. [LSN,TA,BOT/COMMIT,PrevLSN]

[#1,T1,BOT,0]
[#2,T2,BOT,0]
[#3,T1,PA,a+=1, a-=1,#1]
[#4,T1,PB,b+=1,b-=1,#3]
[#5,T2,PB,b*=10,b/=10,#2]
[#6,T1,COMMIT,#4]

0 1 2

c) Geben Sie die im Rahmen des Wiederanlaufs erzeugten CLRs (*compensation log records*) auf Basis logischer Protokollierung an. Hinweis: Form eines CLR-Eintrages: <LSN,TA,PageID,Redo,PrevLSN,UndoNxtLSN>

<#5',T2,Pb,b\=10,#5,#2>
<#2',T2,--,--,#5',0>

Aufgabe 2 Historien (8 Punkte)

Gegeben seien die folgenden Konfliktoperationen:

$$w_1(x) < r_3(x)$$
$$w_3(x) < w_1(x)$$
$$r_2(x) < w_1(x)$$
$$r_2(x) < w_3(x)$$

Weiterhin ist bekannt, dass jede Transaktion direkt nach Ihrer letzten Konfliktoperation committed.
Geben Sie für jede Eigenschaft an, ob sie von der Historie erfüllt wird und begründen Sie kurz.

a)* Serialisierbar (SR)

nein

0 1

b)* Rücksetzbar (RC)

ja

0 1

c)* Vermeidet kaskadierendes Zurücksetzen (ACA)

nein

0 1

d)* Strikt (ST)

nein

0 1

e)* Geben Sie eine mögliche Historie an.

w 1 (x) ,r 3 (x),w 3 (x), w 1 (x),r 2 (x) , w 1 (x), c1, r 2 (x) , w 3 (x), c2,c3

0 1 2

f)* Geben Sie an, ob die Historie durch 2PL erzeugt worden sein kann und begründen Sie kurz.

nein da 2PL SR garantiert

0 1 2

Aufgabe 3 k-Anonymität (10 Punkte)

Gegeben sei die Tabelle `Noten` mit Schema {[*Name*, *Note*]} einer statistischen Datenbank. Eine SQL-Schnittstelle erlaubt Ihnen auf aggregierte Werte der Tabelle zuzugreifen. Im `SELECT`-Teil sind nur `COUNT` und `AVG` zulässig. Anfragen, die weniger als vier Tupel betreffen, werden abgewiesen. having count(*) > 4;
Sie haben Kenntnis über folgende (benannte) Anfragen mit zugehörigem Ergebnis:

Name	Anfrage	avg	count
alle	`SELECT AVG(Durschnittsnote), COUNT(*) FROM Noten`	3.1	5
bestanden	`SELECT AVG(Durschnittsnote), COUNT(*) FROM Noten WHERE Note <= 4.0`	2.75	4 1 besser als 2.0
naja	`SELECT AVG(Durschnittsnote), COUNT(*) FROM Noten WHERE Note > 2.0`	3.5	4

0 1 2 3 4

a)* Nehmen Sie an, das Ergebnis der obigen Anfragen liegt in den Tabellen `alle`, `bestanden` und `naja` vor, worauf Sie beliebig SQL-92 ausführen können. Geben Sie eine SQL-Anfrage an, die Ihnen die Note des/der schlechtesten Studenten/-in zurückgibt.

```
select a.avg*a.count - b.avg* b.count
from alle a, bestanden b
```

0 1

b)* Wie lautet die Note des besten Studenten?

1.5

0 1

c)* Wie lautet die Note des schlechtesten Studenten?

4.5

0 1 2 3 4

d)* Rekonstruieren Sie eine mögliche Datenbasis in SQL-92.

```
create table noten (Name text primary key, Note float);
insert into noten (values
```

```
('Julian', 1.5),
('Hans', 3.0),
('Tim', 3.0),
('Mari', 3.5),
('Max', 4.5),
```

```
);
```

Aufgabe 4 Datalog (11 Punkte)

Gegeben sei die Faktentabelle `strecke` eines Kursbuches:

```
%%strecke(Start,Ziel,Distanz)
strecke(m,n,171). % München -> Nürnberg
strecke(m,w,412). % München -> Wien
strecke(m,s,223). % München -> Stuttgart
strecke(s,f,203). % Stuttgart -> Frankfurt
strecke(n,f,232). % Nürnburg -> Frankfurt
strecke(f,d,229). % Frankfurt -> Düsseldorf
```

Außerdem ist der rekursive Datalogausdruck `erreichbarMuenchen` gegeben, der alle von München erreichbaren Städte ausgibt.

```
erreichbarMuenchen(m).
erreichbarMuenchen(Stadt) :- erreichbarMuenchen(X),strecke(X,Stadt,_).
```

a)* Demonstrieren Sie die *naive* Auswertung des Prädikats `erreichbarMuenchen` bis es terminiert.
Geben Sie jeden Schritt der Auswertung in einer Zeile an (Iteration und Tupelliste) z.B.: `0: m`

0 1 2

1. [m,n], [m,w],[m,s]
2. [m,n], [m,w],[m,s], [m,f]
3. [m,n], [m,w],[m,s], [m,f], [m,d]
4. [m,n], [m,w],[m,s], [m,f], [m,d]

b)* Wann terminiert die *naive* Auswertung?

0 1

sobald keine Änderung mehr Auftritt

c)* Demonstrieren Sie die *semi-naive* Auswertung des Prädikats `erreichbarMuenchen` bis es terminiert.
Geben Sie jeden Schritt der Auswertung in einer Zeile an (Iteration und Tupelliste) z.B.: `0:m`

0 1 2

1. [m,n], [m,w],[m,s]
2. [m,f]
3. [m,d]
4. leere Menge

d)* Wann terminiert die *semi-naive* Auswertung?

0 1

wenn nichts neues hinzukommt

e)* Übersetzen Sie den Datalogausdruck `erreichbarMuenchen` in rekursives SQL. Nehmen Sie an, dass die Tabelle `strecke` mit den Spalten `von`, `nach` und `distanz` existiert.

0 1 2 3 4

```
erreichbarMuenchen(m).
erreichbarMuenchen(Stadt) :- erreichbarMuenchen(X),strecke(X,Stadt,_).

with recursive erreichbarMuenchen(v, n) as (
select von, nach
from strecke
where v=von
union
select v, n
from erreichbarMuenchen
where v = nach
)
select nach from erreichbarMuenchen where von =München
```

Aufgabe 5 Quorum-Consensus (7 Punkte)

Um Ausfallsicherheit zu garantieren ist ein Datenwert ’A’ auf vier Rechnern verteilt. Jeder Rechner hält dabei eine vollständige Kopie von ’A’ mit einem zugehörigen Wert und einer aufsteigend vergebenen Versionsnummer. Um Konsistenz zu garantieren wird das Quorum-Consensus-Verfahren eingesetzt. Dabei ist jedem Rechner ein Gewicht $w_i(A)$ wie folgt zugewiesen:

Rechner	Kopie	Gewicht $w_i(A)$	Wert	Version
R_1	A_1	3	1100	2
R_2	A_2	2	1100	2
R_3	A_3	1	1000	1
R_4	A_4	3	1000	1

0 1 2

a) * Wie groß ist das Schreibquorum $Q_w(A)$? Woran lesen Sie das ab?

5

0 1

b) Wie muss das zugehörige minimale Lesequorum $Q_r(A)$ gewählt sein? Begründen Sie kurz (Rechenweg ausreichend)!

auch 5?

0 1 2 3

c) Skizieren Sie das Schreiben eines Wertes anhand eines selbstgewählten, minimalen Beispiels. Es reicht, wenn Sie den neuen Zustand als vier Dreier-Tupel `(Rechner, Wert, Version)` angeben. Beginnen Sie mit `(1,1100,2),(2,1100,2),(3,1000,1),(4,1000,1)`.

R1-2 möchten nochmal 100 hinzufügen
(1,1200,3),(2,1200,3),(3,1000,1),(4,1000,1)

0 1

d) Wenn wir nun einen weiteren Rechner R_5 für Kopie A_5 hinzufügen mit Gewicht 2, wie groß muss nun das minimale Schreibquorum $Q'_w(A)$ sein? Begründen Sie kurz (Rechenweg ausreichend)!

W > 5.5

Aufgabe 6 **Assoziationsregeln (8 Punkte)**

Gegeben sei eine Relation `Einkaeufe`:

```
CREATE TABLE einkaeufe(korb integer, artikel text);
```

Der folgende Aufruf einer Table-Function wertet den Apriori-Algorithmus aus mit minimalem Support $0,2$ und minimaler Konfidenz $0,7$.

```
SELECT * FROM apriori((select korb, artikel from einkaeufe), 0.2, 0.7);
```

Das zugehörige Ergebnis der Assoziationsregeln (`Links`⇒`Rechts`) sieht wie folgt aus:

Links	Rechts	Support	Konfidenz
{}	{Banane}	0.8	0.8
{Keks}	{Banane}	0.4	1.0
{Milch}	{Banane}	0.2	1.0
{Milch}	{Keks}	0.2	1.0
{Banane,Milch}	{Keks}	0.2	1.0
{Keks,Milch}	{Banane}	0.2	1.0

a)* Wie viele verschiedene Körbe muss `Einkaeufe` mindestens enthalten, um auf den genannten Support zu kommen? (`select count(distinct korb) from einkaeufe`)

0 1

b) Wie viele verschiedene Körbe davon müssen *Banane* enthalten?

0 1

c) Wie viele verschiedene Körbe davon müssen auch *Keks* enthalten?

0 1

d) Wie viele verschiedene Körbe davon müssen auch *Milch* enthalten?

0 1

e)* Befüllen Sie in SQL-92 die leere Relation `Einkaeufe` zu den Assoziationsregeln passend minimal mit Tupeln.

0 1 2 3 4

```
insert into einkaeufe (values

);
```

Aufgabe 7 Window-Functions (10 Punkte)

Betrachten Sie den folgenden, fiktiven Ausschnitt aus der anonymisierten Tabelle `Urlaubsrueckkehrer` mit Schlüssel Reiseland und Tag (Tagesnummern fortlaufend ab Beginn der Tabelle).

Reiseland	Tag	Positiv	Negativ
Bulgarien	1	4	34
Bulgarien	2	3	10
Griechenland	1	12	50
Griechenland	2	9	63
Griechenland	3	13	59
Kroatien	1	9	42
Kroatien	3	7	45
⋮	⋮	⋮	⋮

Lösen Sie nachfolgende Teilaufgaben in SQL:2003.

0 1 2 3

a)* Berechnen Sie mittels Window-Functions tagesaktuell die wachsende Summe aller jemals positiv getesteten Urlaubsrückkehrer je Reiseland.

```
select reiseland, sum(positiv) over(partition by reiseland order by Tag range between unbounded preceding and current row)
```

0 1 2 3 4

b)* Berechnen Sie mittels Window-Functions den laufenden Mittelwert des Anteils positiv getesteter Urlaubsrückkehrer je Reiseland in Prozent, wobei der laufende Mittelwert über fünf Tage, beginnend mit dem zweiten Vortag, gebildet werden soll.

```
select reiseland, avg(positiv / (positiv + negativ)) over(partition by reiseland order by tag range between 2 preceding and 2 following)
from Urlaubsrueckkehrer;
```

0 1 2 3

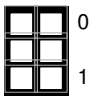
c)* Berechnen Sie mittels Window-Functions für jeden Tag die drei Reiseländer mit den meisten positiv getesteten Urlaubsrückkehrern.

```
select reiseland, rank() over(partition by day order by positiv desc )
from Urlaubsrueckkehrer;
```

Aufgabe 8 Hauptspeicher-Datenbanken (8 Punkte)

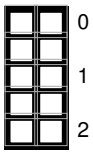
In dieser Aufgabe soll ein Adaptiver-Radix-Baum ART betrachtet werden, welcher alle 32-Bit-Integer im Bereich von 20218 (0x00 0x00 0x4E 0xFA) bis einschließlich 30000 (0x00 0x00 0x75 0x30) enthält. Der hierbei konstruierte ART verwendet keine Präfix-Kompression, somit besteht die Wurzel aus einem einzelnen Node4 mit einem einzigen Eintrag. Geben Sie nun für jede weitere Ebene des Baumes an, wie viele Knoten eines jeden Typs auf der entsprechenden Ebene vorliegen. Betrachten Sie dabei nur die endgültige Form des erstellten ART und dokumentieren Sie Ihre Vorgehensweise.

a)* Geben Sie für die zweite Ebene an, wie viele Knoten eines jeden Typs auf dieser Ebene vorliegen.



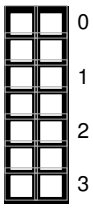
1 -> passt in node4

b) Geben Sie für die dritte Ebene an, wie viele Knoten eines jeden Typs auf dieser Ebene vorliegen.
Hinweis: $0x4E = 78, 0x75 = 117$

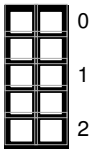


node 48

c) Geben Sie für die vierte Ebene an, wie viele Knoten eines jeden Typs auf dieser Ebene vorliegen.
Hinweis: Betrachten Sie linken und rechten Randknoten gesondert, $0x30 = 48, 0xFA = 250, 0xFF = 255$



d)* Nennen Sie zwei allgemeine Vorteile des ART gegenüber eines auf binärer Suche basierenden B-Baums bei einer hinreichend großen Anzahl an Einträgen.



besser ausgelastet

Aufgabe 9 XML und JSON (10 Punkte)

a)* Gegeben sei folgende XQuery-Anfrage:

```
for $x in (1,2) return
<sws semester="{$x}"> {
 for $s in doc('hoeren.xml')/uni/st
  return <person name = "{$s/name/text()}"> {
    let $v := tokenize($s/plan/sem[$x]/@vls," ")
    return sum(doc('hoeren.xml')/uni/vls/vl[@VNr=$v]/@SWS)
  } </person>
} </sws>
```

Erstellen Sie ein zu folgender Ausgabe passendes XML-Dokument `hoeren.xml`:

```
<sws semester="1">
   <person name="Josef"> 2 <person/>
   <person name="Max">   3 <person/>
</sws>
<sws semester="2">
   <person name="Josef"> 5 <person/>
   <person name="Max">   2 <person/>
</sws>
```

```
<uni>
<st>
<name>Josef<\name>
<plan>
<sem vls="1"\>
<sem vls="3"\>

<\plan>
<\st>
<st>
<name>Max<\name>
<plan>
<sem vls="2"\>
<sem vls="1"\>

<\plan>
<\st>
<vls>
<vl VNr="1" SWS="2"  />
<vl VNr="2" SWS="3"  />
<vl VNr="3" SWS="5"  />
</vls>
<\uni>
```

b)* Gegeben sei folgende SQL-Anfrage: `select doc->1->'Name' as Name, doc->1->'Raum' as Raum from uni_json;`
Vervollständigen Sie nachfolgende insert-Anfrage auf einer anfangs leeren Relation `uni_json` durch einen passenden JSON-Ausdruck, sodass die select-Anfrage das folgende Ergebnis liefert:

Name	Raum
Max	02.11.060

```
insert into uni_json (name, doc) values ('uni','
```

```
{
[ 'Name': 'Josef', 'Raum': '02.11'],
[ 'Name': 'Max', 'Raum': '02.11.060']
}
```

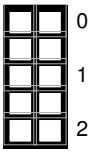
```
');
```

Aufgabe 10 SPARQL (8 Punkte)

```
@prefix ex: <http://maerchen.example.org/>.
ex:verlag1 ex:name "Wissenschaftsverlag".
ex:verlag2 ex:name "Grimm-Gesellschaft".
ex:rap ex:hatTitel "Rapunzel";
       ex:hatAutor ex:Sokrates;
       ex:erschienen 1816.
ex:asp ex:hatTitel "Aschenputtel";
       ex:hatAutor ex:Archimedes;
       ex:verlegtBei ex:verlag2.
ex:scw ex:hatTitel "Schneewittchen";
       ex:hatAutor ex:Platon;
       ex:erschienen 2004;
       ex:verlegtBei ex:verlag1.
```

Drücken Sie die nachfolgenden Anfragen in SPARQL aus, wobei Ihre Anfragen auch auf anderen Ausprägungen der Datenbasis funktionieren sollen.

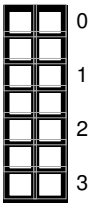
a)* Bestimmen Sie zu jedem Autor die Anzahl seiner veröffentlichten Bücher.




```
PREFIX ex:<http://maerchen.example.org/>
```

```
select (count ?book) , ?autor
where {
?book ex:hatAutor ?autor
}
```

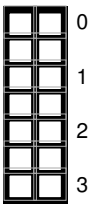
b)* Geben Sie die Titel aller Bücher des Verlags *Wissenschaftsverlag* aus.




```
PREFIX ex:<http://maerchen.example.org/>
```

```
select ?titel
where {
?book ex:verlegtBei ex:verlag1.
?book ex:hatTitel ?titel
}
```

c)* Gegeben sei folgende Anfrage:




```
PREFIX ex:<http://maerchen.example.org/>
SELECT ?n WHERE {
    ?v ex:sitz ?n.
    ?b1 ex:velegtBei ?v.
    ?b2 ex:velegtBei ?v.
    FILTER (?b1 != ?b2)
}
```

Geben Sie eine minimale Menge an Tripel an, die der Datenbasis hinzugefügt werden müssen, damit diese Anfrage ein nicht-leeres Ergebnis zurückgibt. Hinweis: Das Ungleichheitsprädikat wird im Webinterface nicht unterstützt.

```
ex:rap ex:velegtBei ex:verlag1.
ex:verlag1 ex:sitz "Berlin" .
```

Zusätzlicher Platz für Lösungen. Markieren Sie deutlich die Zuordnung zur jeweiligen Teilaufgabe. Vergessen Sie nicht, ungültige Lösungen zu streichen.