

Gruppe A

Bitte tragen Sie **SOFORT** und **LESERLICH** Namen und Matrikelnr. ein, und legen Sie Ihren Studentenausweis bereit.

PRÜFUNG AUS		09.05.2018
<input type="radio"/> DATENMODELLIERUNG 2 (184.790) <input type="radio"/> DATENBANKSYSTEME (184.686)		GRUPPE A
Matrikelnr.	Familiennamen	Vorname

Arbeitszeit: 60 Minuten. Lösen Sie die Aufgaben auf den vorgesehenen Blättern; Lösungen auf Zusatzblättern werden nicht gewertet. **Viel Erfolg!**

Aufgabe 1: Protokollierung (Logging) (10)

Betrachten Sie den angegebenen Inhalt der (persistenten) Log-Datei, des (flüchtigen) Log-Puffers, sowie eine Auflistung der Seiten im Hintergrundspeicher. Das Log-Format ist

[LSN, Transaktion, PageID, Redo, Undo, PrevLSN], bzw. [LSN, Transaktion, BOT/COMMIT, PrevLSN].

Nachdem für diese Aufgabe die durchgeführten Operationen keine Rolle spielen bestehen die Redo- und Undo-Einträge nur aus dem Platzhalter \cdot .

Aufgaben (a) – (b): Bestimmen Sie, ob bei dem dargestellten Zustand das WAL (write ahead log)-Prinzip eingehalten wurde, oder nicht.

Falls es eingehalten wurde, geben Sie eine konkrete Aktion an, welche das WAL-Prinzip verletzen würde. Verwenden Sie eine der folgenden drei Arten von Aktionen: Ausschreiben eines Log-Eintrags aus dem Puffer (geben Sie die LSN an); Auslagern einer Seite in den Hintergrundspeicher (geben Sie die PageID an); Erstellen eines neuen Log-Eintrags (geben Sie den Eintrag an).

Falls das WAL-Prinzip nicht eingehalten wurde, beschreiben Sie wie/wodurch es verletzt wurde.

Achtung: Ankreuzen alleine gibt keine Punkte!

a) Log-Datei	Log-Puffer	Seiten im Hintergrundspeicher
#1, T_1 , BOT, #0	#6, T_2 , P_c , \cdot , \cdot , #4	P_a LSN: #5
#2, T_1 , P_b , \cdot , \cdot , #1	#7, T_2 , P_b , \cdot , \cdot , #6	P_b LSN: #2
#3, T_1 , P_a , \cdot , \cdot , #2	#8, T_1 , P_a , \cdot , \cdot , #5	P_c LSN: #6
#4, T_2 , BOT, #0	#9, T_3 , BOT, #0	
#5, T_1 , P_a , \cdot , \cdot , #3	#10, T_3 , P_b , \cdot , \cdot , #9	
	#11, T_1 , P_a , \cdot , \cdot , #8	

Das WAL-Prinzip wurde eingehalten: <input type="radio"/> ja <input type="radio"/> nein
Begründung:

b) **Log-Datei**

[#1, T_3 , BOT, #0]
[#2, T_2 , BOT, #0]
[#3, T_2 , P_a , ·, ·, #2]
[#4, T_3 , P_a , ·, ·, #1]
[#5, T_3 , P_b , ·, ·, #4]
[#6, T_2 , P_c , ·, ·, #3]
[#7, T_3 , P_b , ·, ·, #5]

Log-Puffer

[#8, T_1 , BOT, #0]
[#9, T_2 , P_c , ·, ·, #6]
[#10, T_1 , P_b , ·, ·, #8]
[#11, T_3 , P_a , ·, ·, #7]

Seiten im Hintergrundspeicher

P_a LSN: #0
P_b LSN: #0
P_c LSN: #0

Das WAL-Prinzip wurde eingehalten: ja nein

Begründung:

.....

.....

c) **Log-Datei**

[#1, T_3 , BOT, #0]
[#2, T_3 , P_a , ·, ·, #1]
[#3, T_1 , BOT, #0]
[#4, T_3 , P_c , ·, ·, #2]

Log-Puffer

[#5, T_1 , P_b , ·, ·, #3]
[#6, T_2 , BOT, #0]
[#7, T_1 , P_a , ·, ·, #5]
[#8, T_2 , P_c , ·, ·, #6]
[#9, T_2 , P_b , ·, ·, #8]

Seiten im Hintergrundspeicher

P_a LSN: #2
P_b LSN: #0
P_c LSN: #0

Nehmen Sie an, die Transaktion T_2 soll mit einem COMMIT abgeschlossen werden. Welche Aktionen im Bezug auf das Anlegen neuer Log-Einträge, Ausschreiben von Log-Einträgen in die Log-Datei und dem Auslagern von Seiten in den Hintergrundspeicher *müssen* durchgeführt werden, bevor das COMMIT erfolgreich ist, wenn zum Einen das WAL-Prinzip eingehalten werden soll, und zum Anderen die Systemkonfiguration *steal* und *not force* ($\neg force$) angenommen wird?

.....

.....

.....

.....

Was würde sich ändern, wenn *force* statt $\neg force$ angenommen wird?

.....

.....

.....

Aufgabe 2: Eigenschaften von Historien

(12)

Betrachten Sie die angegebenen Historien, welche durch eine Abfolge von Elementaroperationen der vier Transaktionen T_1, T_2, T_3 und T_4 auf den Datensätzen A, B, C und D gegeben sind. Dabei bedeutet $r_i(X)$ dass Transaktion T_i den Datensatz X liest (*read*), $w_i(X)$ dass Transaktion T_i den Datensatz X schreibt (*write*), a_i steht für den Abbruch der Transaktion T_i , und c_i steht für das *commit* der Transaktion T_i .

a) Geben Sie für die folgende Historie eine Liste aller Paare (op_i, op_j) von Konfliktoperationen aus. Achten Sie darauf, dass jeweils die frühere Operation an erster Stelle steht.

$r_2(B), w_3(A), w_4(C), r_4(B), w_2(D), r_3(C), r_1(B), w_1(D), r_4(A), r_2(A), c_1, c_2, c_3, c_4$

Konfliktoperationen:
.....

b) Bestimmen Sie, ob es für die folgende Historie eine konfliktäquivalente, serielle Historie gibt.

Falls ja, geben Sie eine solche an.

Falls nein, geben Sie eine minimale Menge von Transaktionen aus, welche von der Historie entfernt werden müssten, damit es eine konfliktäquivalente, serielle Historie gibt, und geben Sie diese an.

$r_1(A), r_3(C), w_4(A), r_4(D), r_1(D), w_3(D), w_1(C), a_3, c_4, c_1, c_2$

Konfliktäquivalente, serielle Historie:
Folgende Transaktionen müssen dazu entfernt werden:

c) Bestimmen Sie, ob die folgende Historie konfliktserialisierbar bzw. strikt ist. Geben Sie jeweils eine kurze, konkrete Begründung an Hand der Historie an. (*Achtung! Ankreuzen alleine gibt keine Punkte!*)

$w_1(A), r_2(B), w_1(B), r_3(C), c_1, r_2(A), w_3(A), c_3, r_2(A), c_2$

Konfliktserialisierbar: <input type="radio"/> ja <input type="radio"/> nein Begründung:
.....
Strikt: <input type="radio"/> ja <input type="radio"/> nein Begründung:
.....

Sie finden auf der nächsten Seite eine Historie von drei Transaktionen T_1 , T_2 und T_3 . Es gibt zwei Datenobjekte A und B .

Wenden Sie die Zeitstempel-basierte Synchronisationsmethode auf die Historie an. D.h., vervollständigen Sie die sich ebenfalls auf der nächsten Seiten befindliche Tabelle, indem Sie in die Spalte **Aktion** die jeweilige Operation eintragen, und in die vier folgenden Spalten jeweils die Lese- und Schreibzeitstempel von A und B . (Sie brauchen diese Werte nicht in allen Zeilen angeben. Es reicht, wenn Sie jeweils nur die Werte angeben welche sich ändern). Lassen Sie die erste Zeile frei, sie enthält die Anfangswerte der Zeitstempel.

Für die Spalte **Aktion** verwenden Sie bitte für $i \in \{1, 2, 3\}$ die Schreibweise BOT_i (Beginn einer Transaktion), $r_i(X)$ (Transaktion T_i liest Datenobjekt X), $w_i(X)$ (Transaktion T_i schreibt Datenobjekt X), c_i (Commit von Transaktion T_i), und $reset_i$ (Rücksetzen von Transaktion T_i).

Als Zeitstempel für die Transaktionen verwenden Sie bitte die Zeilennummer des entsprechenden BOT -Eintrags.

Die Schreibweise in der angegebenen Historie ist wie oben beschrieben. Einträge *res?* bedeuten, dass zu diesem Zeitpunkt genau eine Transaktion neu gestartet werden soll, falls es zu dieser Zeit eine Transaktion gibt welche zurückgesetzt aber noch nicht neu gestartet wurde. Anschließend an den Neustart führen Sie bitte alle Operationen der Transaktion bis zum aktuellen Zeitpunkt aus.

Wird eine Transaktion zurückgesetzt, so werden alle weiteren Operationen dieser Transaktion bis zum ihrem Neustart ignoriert.

Wenn Sie am Ende der Historie angekommen sind, und nicht alle rückgesetzten Transaktionen neu gestartet wurden (kein passender *res?* Eintrag), dann wird diese Transaktion einfach nicht erneut durchgeführt.

Historie Aufgabe 3:

$BOT_2, r_2(A), BOT_1, BOT_3, r_3(B), r_1(B), w_1(B), w_2(A), res?, w_3(B), r_2(B), c_1, w_2(B), c_3, c_2$

Lösung Aufgabe 3:

#	Aktion	rTS(A)	wTS(A)	rTS(B)	wTS(B)
9		5	5	4	4
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Für die Aufgaben 4 – 6 gilt die Datenbankbeschreibung am letzten Blatt dieser Prüfung.

Aufgabe 4: CREATE Statements

(11)

Achtung: Bitte lesen Sie zuerst aufmerksam die Datenbankbeschreibung am letzten Blatt dieser Prüfung.

Die Tabelle `item` wurde bereits angelegt.

```
CREATE TABLE item(  
    id INTEGER PRIMARY KEY,  
    name VARCHAR(20) NOT NULL,  
    bestand INTEGER NOT NULL  
);
```

Geben Sie nun CREATE-Statements mit allen entsprechenden Constraints für die Tabellen `zutat`, `artikel` und `besteht_aus` an.

Wählen Sie entsprechende Datentypen (INTEGER, VARCHAR, NUMERIC) für die Attribute. Verwenden Sie für Preise eine Gleitkommazahl mit maximal 2 Dezimalstellen.

Nachdem die Datenbank in Betrieb genommen wurde, bemerken Sie, dass die Spalte `bestand` in der Tabelle `item` auch negative Zahlen zulässt (vom Typ INTEGER ist). Schreiben Sie ein Statement, das die Tabelle `item` so verändert, dass alle Daten erhalten bleiben und in der Spalte `bestand` überprüft wird, dass keine negativen Werte erlaubt sind.

Aufgabe 5: Rekursive Abfragen

(12)

Evaluieren Sie das folgendes SQL-Statement bezüglich der Datenbankinstanz auf der letzten Seite, und geben Sie die Ausgabe der Abfrage an:

```
WITH RECURSIVE einkaufspreis(id,besteht,epreis) AS (  
    SELECT a.id, b.besteht, z.epreis  
    FROM artikel a JOIN besteht_aus b ON a.id = b.id  
        LEFT JOIN zutat z ON b.besteht = z.id  
    UNION ALL  
    SELECT e.id, b.besteht, z.epreis  
    FROM einkaufspreis e JOIN besteht_aus b ON e.besteht = b.id  
        LEFT JOIN zutat z ON b.besteht = z.id  
) SELECT e.id AS id, SUM(e.epreis) AS sum  
FROM einkaufspreis e  
GROUP BY e.id  
ORDER BY e.id ASC;
```

Aufgabe 6: PL/pgSQL Trigger

(12)

Erstellen Sie einen PL/pgSQL Trigger `trBestand`, der vor dem Ändern eines Eintrags in der `item`-Tabelle für jede zu ändernde Zeile die Funktion `fBestand` aufruft.

Diese Funktion soll, wenn der Bestand des Items "X" erhöht wird, den Bestand der Items aus denen "X" besteht um dieselbe Menge verringern.

Beachten Sie Folgendes: Der Bestand eines Items darf niemals negativ sein. Falls es aufgrund der Ausführung des Triggers der Bestand negativ wird, soll der Trigger abbrechen und der Bestand nicht verändert werden. *Hinweis:* Sie können annehmen, dass der Constraint in Aufgabe 4 korrekt implementiert ist.

Gesamtpunkte: 70

Sie können diese Seite abtrennen und brauchen ihn nicht abgeben!

Diesen Zettel daher bitte nicht beschriften! (Lösungen auf diesem Zettel werden nicht gewertet!)

Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 6:

Gegeben ist folgendes stark vereinfachtes Datenbankschema zum Speichern von Artikeln in einem Fast-Food Lokal.

item(id, name, bestand)

zutat(id: *item.id*, epreis)

artikel(id: *item.id*, vpreis)

besteht_aus(id: *item.id*, besteht: *item.id*)

Ein *item* ist eine in einem Fast-Food Restaurant erzeugte oder verbrauchte Ware. Sie wird durch eine eindeutige Nummer *id* identifiziert. Zusätzlich werden in der Tabelle *item* noch der Name und der Bestand der Ware gespeichert. Der Bestand einer Ware darf nicht negativ sein.

Ein *item* ist entweder eine *zutat* oder ein *artikel*. Für Zutaten wird der Einkaufspreis *epreis* und für Artikel der Verkaufspreis *vpreis* gespeichert.

Ein *item* kann aus mehreren anderen *items* bestehen. Diese Relation wird in der Tabelle *besteht_aus* gespeichert.

Beispielinstanz für Aufgabe 4 – 6:

item		
id	name	bestand
1	Burger Bun	10
2	Rindfleisch	15
3	Zwiebel	5
4	Gurke	10
5	Pommes	20
6	Nuggets	10
7	Curry Sauce	2
10	Hamburger	2
11	Pommes	0
12	Chicken Nuggets	1
13	Menu 1	0
14	Menu 2	0
15	Family Menu	0

zutat	
id	epreis
1	0.50
2	2.00
3	0.50
4	0.50
5	1.50
6	3.00
7	0.50

artikel	
id	vpreis
10	5.00
11	3.00
12	5.00
13	7.00
14	7.00
15	14.00

besteht_aus	
id	besteht
10	1
10	2
10	3
10	4
11	5
12	6
12	7
13	10
13	11
14	11
14	12
15	14
15	13
15	12