

Gruppe A

Bitte tragen Sie **sofort** und **leserlich** Namen, Studienkennzahl und Matrikelnummer ein und legen Sie Ihren Studentenausweis bereit.

PRÜFUNG AUS DATENBANKSYSTEME VL 181.186			15. 6. 2011
Kennnr.	Matrikelnr.	Familienname	Vorname

Arbeitszeit: 120 Minuten. Aufgaben sind auf den Angabebältern zu lösen; Zusatzblätter werden nicht gewertet.

Aufgabe 1:

(18)

Gegeben ist die folgende Historie von Transaktionen:

Schritt	T_1	T_2	T_3	Log: [LSN, TA, PageID, Redo, Undo, PrevLSN] or ⟨LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN⟩
1	BOT			[#1, T_1 , BOT, 0]
2		BOT		[#2, T_2 , BOT, 0]
3			BOT	[#3, T_3 , BOT, 0]
4	$r(A, a_1)$			
5		$r(B, b_2)$		
6			$r(A, a_3)$	
7	$r(B, b_1)$			
8	$w(A, a_1 + b_1)$		
9			$w(C, 2 * a_3)$
10		$r(C, c_2)$		
11		$w(C, b_2 + c_2)$	
12	$r(C, c_1)$			
13	$w(B, b_1 + c_1)$		
14			commit
15		commit	
16
17			
18			
19			

(a) Nehmen Sie an, dass in Zeile 16 auch die Transaktion T_1 mittels commit beendet wird. Ist die resultierende Historie serialisierbar?

☐ ja ☐ nein

Wenn ja, in Reihenfolge T vor T vor T

Wenn nein: die Historie wird durch das Streichen von zumindest Operationen serialisierbar.

(b) Führen Sie nun – unabhängig davon, ob die Historie serialisierbar ist – in Zeile 16 ein *rollback* für T_1 durch.

Zu Beginn ist der relevante Datenbestand in der Datenbank $A = 50$, $B = 60$ und $C = 70$. Tragen Sie nun das Recovery-Log zu dieser Historie (mit rollback von T_1 in Zeile 16) in die rechte Spalte der obigen Tabelle ein. Dabei sind Undo/Redo-Einträge *relativ* zum Datenbestand anzugeben. Geben Sie in den Zeilen 16 ff. die Log-Einträge für die Recovery an.

Die Werte von A , B und C nach dem rollback von T_1 sind $A : \dots\dots\dots; B : \dots\dots\dots; C : \dots\dots\dots$.

Aufgabe 2: (15)

Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

1. Nehmen Sie an, dass eine Relation R 128 Seiten umfasst und die Puffergröße 128 beträgt. Dann ist bei einem Hash Join von R mit einer beliebigen Relation S in der Build-Phase auf jeden Fall ein Re-Hashing von R erforderlich.

wahr ☐ falsch ☐
2. Beim Zweiphasen-Commit-Protokoll können die Agenten mittels Timer-Überwachung – durch geeignete Wahl des Timer-Intervalls – verhindern, dass sie bei einem Absturz des Koordinators blockiert werden.

wahr ☐ falsch ☐
3. Die Historie $r_1(A), w_1(A), r_2(B), c_1, r_2(A), w_2(A), w_2(B), c_2$ ist sowohl beim Zwei-Phasen Sperrprotokoll als auch beim strengen Zwei-Phasen Sperrprotokoll möglich.

wahr ☐ falsch ☐
4. Betrachten Sie zwei Relationen $R(ABC)$ und $S(BDE)$. Dann liefert der Ausdruck $(\pi_B(R) \cup \pi_B(S)) \bowtie R$ ausschließlich Tupel, die auch in R enthalten sind.

wahr ☐ falsch ☐
5. Eine Relation R sei an 4 Netzwerk-Knoten materialisiert mit den Gewichten 9, 7, 15 und 2. Dann sind $Q_r(R) = 16$ und $Q_w(R) = 17$ gültige Lese- bzw. Schreib-Quoren.

wahr ☐ falsch ☐
6. Betrachten Sie drei Relationen $R(AB)$, $S(AB)$ und $T(BC)$. Dann gilt auf jeden Fall folgende Gleichheit:
 $\pi_C((R \cap S) \bowtie T) = \pi_C([R \bowtie T] \bowtie [S \bowtie \pi_B(T)])$.

wahr ☐ falsch ☐
7. Um eine Sort-merge-join Operation durchzuführen, ist jeweils ein Hash-Index für die Primärschlüssel der beiden Relationen erforderlich.

wahr ☐ falsch ☐
8. Will man in PL/pgSQL eine komplette Zeile einer Tabelle „table“ in einer Variable ablegen, so muss diese entweder vom Typ RECORD oder vom Typ table%ROWTYPE sein.

wahr ☐ falsch ☐
9. Durch mehrere aufeinanderfolgende Forwards (auch „Chaining“ genannt) kann es zu hohen Zugriffszeiten kommen, da viele Seiten für nur ein Tupel eingelagert werden müssen.

wahr ☐ falsch ☐
10. Für Relation $R(AB)$ mit 100 Tupeln und Relation $S(\underline{A}C)$ mit 10 Tupeln enthält $\Pi_A(R) - \Pi_A(S)$ mindestens 90 Tupel.

wahr ☐ falsch ☐

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrektter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

Aufgabe 3: (12)

Beweisen Sie formal, dass das Zeitstempel basierende Synchronisationsverfahren nur serialisierbare Historien erzeugt.

Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 7:

Gegeben ist folgendes stark vereinfachtes Datenbankschema, das Filme und ihre Genres repräsentiert:

movie(mid, title, rating)
genre(gid, name, parent: *genre.gid*)
movie_genre(mid: *movie.mid*, gid: *genre.gid*)

Jeder Film (**movie**) hat eine eindeutige ID (**mid**), einen Titel (**title**) und eine Kritikerbewertung (**rating**).
Jedes Genre (**genre**) hat eine eindeutige ID (**gid**), einen Namen (**name**) und ein übergeordnetes Genre (**parent**).

Aufgabe 4:

(6)

Nehmen Sie an, dass die Tabellen durch folgende CREATE Statements angelegt wurden:

```
CREATE TABLE movie (  
    mid INTEGER PRIMARY KEY,  
    title VARCHAR(30),  
    rating INTEGER  
);  
  
CREATE TABLE genre (  
    gid INTEGER PRIMARY KEY,  
    name VARCHAR(30),  
    parent INTEGER  
);  
  
CREATE TABLE movie_genre (  
    mid INTEGER REFERENCES movie(mid),  
    gid INTEGER REFERENCES genre(gid)  
);
```

Fügen Sie nun ein Constraint hinzu, das den Primärschlüssel der Tabelle **movie_genre** definiert:

Fügen Sie nun ein Constraint hinzu, das den Fremdschlüssel **parent** der Tabelle **genre** definiert:

Fügen Sie nun ein Constraint hinzu, das als **rating** nur Werte von 0 bis 100 zulässt.

Gegeben sind folgende Tabellen:

movie				
mid	title	rating		
1	'Hangover 2'	70		
2	'Fluch der Karibik 4'	69		
3	'X-Men 4'	82		
4	'Source Code'	77		
5	'Rio'	72		
6	'The Fast and the Furious 5'	77		
			movie_genre	
			mid	gid
			1	1
			2	2
			2	3
			2	4
			3	3
			3	5
			4	3
			4	5
			5	1
			5	6
			6	3
genre				
gid	name	parent		
1	'Comedy'	NULL		
2	'Abenteuer'	10		
3	'Action'	10		
4	'Fantasy'	11		
5	'Science Fiction'	11		
6	'Animation'	NULL		
10	'Action-Adventure'	NULL		
11	'SF-Fantasy'	NULL		

Geben Sie die Ergebnisse für die folgenden SELECT-Statements an. Falls mehrere Tupel zurückgegeben werden, notieren Sie diese als Tabelle (z.B. in ähnlichem Format wie oben gegeben).

```
SELECT COUNT(*)
  FROM movie, genre;
```

--

```
SELECT COUNT(*)
  FROM movie m, genre g, movie_genre mg
 WHERE m.mid = mg.mid AND mg.gid = g.gid;
```

--

```
SELECT g.gid, COUNT(*) as count
  FROM movie m, genre g, movie_genre mg
 WHERE m.mid = mg.mid AND mg.gid = g.gid
 GROUP BY g.gid ORDER BY g.gid;
```

--

Aufgabe 6: PL/SQL Trigger

(8)

Erstellen Sie einen Trigger, der verhindert, dass für einen Film mehr als drei Genres zugeordnet werden können. Die Zuordnung der Filme zu Genres ist in der Tabelle `movie_genre` angegeben.

D.h. in die Tabelle `movie_genre` soll ein Tupel nur eingefügt werden, wenn diese Bedingung danach nicht verletzt wird.

Beispielsweise sind in unserer Testdatenbank bereits drei Genres für den Film 'Fluch der Karibik 4' vorhanden (nämlich 'Abenteuer', 'Action' und 'Fantasy'). Ein Tupel in `movie_genre`, das zu 'Fluch der Karibik 4' das Genre 'Comedy' zuordnet soll also nicht hinzugefügt werden können.

- die Namen aller zugeordneten Genres ausgibt (**name**) (mittels `System.out.println(...)`)
- das Rating (**rating**) um 10 erhöht, sofern es nicht größer als 90 ist

Beachten Sie dazu folgende Hinweise:

- die genaue Formatierung der Ausgabe ist nicht relevant (Trennzeichen zwischen Genres sind nicht notwendig, ...)
- Sie können davon ausgehen, dass eine **Connection c** zur Verfügung steht
- achten Sie darauf, die korrekte Methode zur Ausführung der **PreparedStatement**s aufzurufen

Legen Sie nun die für die Methode `movieEdit` benötigten `PreparedStatement`s an. (`PreparedStatement ps = ...`) Sie können davon ausgehen, dass eine `Connection c` zur Verfügung steht.

[illegible]

```
public void movieEdit(int mid) throws Exception {
```