Gruppe **A**

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

PRÜFUNG AUS	**MUSTERLÖSUNG**	24.01.2020
◯ DATENMODELLIERUNG 2 (**184.790**)	◯ DATENBANKSYSTEME (**184.686**)	**GRUPPE A**

Matrikelnr.	Familienname	Vorname

Arbeitszeit: 90 Minuten. Lösen Sie die Aufgaben auf den vorgesehenen Blättern; Lösungen auf Zusatzblättern werden nicht gewertet. **Viel Erfolg!**

Aufgabe 1: Sperrprotokolle (10)

Auf der nächsten Seite ist eine Sequenz von Elementaroperationen der Transaktionen T_1, T_2, T_3 und T_4 auf den Datenbankobjekten A, B, C und D gegeben.

a) Diese Zugriffe sollen mit Hilfe des *strikten 2-Phasen Sperrprotokolls* synchronisiert werden.

(i) Zeichnen Sie den Wartegraphen zu dem mit (*) markierten Zeitpunkt, und geben Sie an ob ein Deadlock vorliegt.

b) Um mögliche Deadlocks zu vermeiden soll nun das strikte 2-Phasen Sperrprotokoll mit der *wound-wait Strategie* zur Deadlockvermeidung kombiniert werden.

(i) Weisen Sie dazu jeder Transaktion einen korrekten Zeitstempel zu. Geben Sie außerdem für jede neu gestartete Transaktion ihren Zeitstempel nach dem Neustart an.

(ii) Geben Sie ebenfalls die Folge von Sperranforderungen und Freigaben an, welche bei Abarbeitung der Sequenz erzeugt werden. Verwenden Sie dazu die auf der nächsten Seite beschriebene Notation.

Annahmen und Konventionen (wie aus der Vorlesung/Übung bekannt):

Nehmen Sie an, dass nur notwendige Sperren angefordert werden, und dies so spät wie möglich. Den Zeitpunkt der Freigaben können Sie – innerhalb der Einschränkungen durch das Sperrprotokoll – frei wählen.

*Wird eine Transaktion angehalten, so werden ihre weiteren Aktionen übersprungen (die Transaktion ist ja blockiert). Kann durch eine Freigabe eine blockierte Transaktion weiterlaufen, so werden alle übersprungenen Aktionen dieser Transaktion sofort nachgeholt. Warten mehrere Transaktionen auf die selbe Freigabe, oder warten mehrere Transaktionen auf einen Neustart, so wird jene Transaktion mit dem **kleinsten Index (!)** ausgewählt.*

Lock Upgrades können durchgeführt werden sobald keine weitere Sperre auf dem Objekt existiert. Ansonsten werden Anforderungen für Schreibsperren gleich behandelt, egal ob die Transaktion bereits eine Lesesperre auf dem Objekt besitzt oder nicht. Wird die Schreibsperre gewährt, hält die Transaktion im Anschluss nur mehr die Schreibsperre.

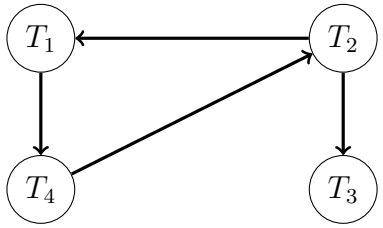
Sequenz für a) und b)

T_1	T_2	T_3	T_4
	b_2		
			b_4
			$r_4(B)$
			$w_4(D)$
	$r_2(B)$		
			$w_4(B)$
		b_3	
		$w_3(C)$	
b_1			
$r_1(A)$			
		$r_3(A)$	
		$r_3(B)$	
restart?			
	$w_2(A)$		
	$r_2(A)$		
	$r_2(B)$		
		$w_3(C)$	
$w_1(D)$			
(*)			
		c_3	
restart?			
	c_2		
			$r_4(D)$
			c_4
$w_1(A)$			
c_1			

Notation:

- $r_i(O)$ und $w_i(O)$: Lese- bzw. Schreibzugriff von T_i auf Objekt O.
- b_i, c_i: Beginn bzw. Commit von T_i.
- *restart?*: Neustart einer zurückgesetzten Transaktion [nur b)]

Wartegraph für Aufgabe a)



Deadlock: ⊗ ja ◯ nein

Sperren für Aufgabe b)

Anzugebende Einträge und Notation:

- Anfordern von Sperren: $S_i(O)/X_i(O)$ um anzuzeigen, dass T_i eine Lese-/Schreibsperre auf O anfordert.
- Warten/Blockieren einer Transaktion: $wait_i$ um anzuzeigen, dass T_i eine angeforderte Sperre nicht erhält und blockiert.
- Erhalt einer Sperre (grant): $gS_i(O)/gX_i(O)$ um anzuzeigen, dass T_i eine angeforderte Lese-/Schreibsperre auf O erhält. **Implizit falls auf $S_i(O)$ bzw. $X_i(O)$ kein $wait_i$ oder $reset_i$ folgt; muss dann nicht angegeben werden.**
- Freigabe einer Sperre (release): $rS_i(O)/rX_i(O)$ um anzuzeigen, dass T_i eine gehaltene Lese-/Schreibsperre auf O freigibt. Muss immer (auch bei $reset_i$) explizit angegeben werden.
- Zurücksetzen einer Transaktion: $reset_i$ um anzuzeigen, dass T_i auf Grund des Protokolls zurückgesetzt wurde.

$\mathrm{S}_4(B), \mathrm{X}_4(D), \mathrm{S}_2(B), \mathrm{X}_4(B), wait_4,$ .....

$\mathrm{X}_3(C), \mathrm{S}_1(A), \mathrm{S}_3(A), \mathrm{S}_3(B), \mathrm{X}_2(A),$ .....

$reset_1, \mathrm{rS}_1(A), reset_3, \mathrm{rS}_3(A), \mathrm{rS}_3(B),$ .....

$\mathrm{rX}_3(C), \mathrm{gX}_2(A), \mathrm{S}_1(A), wait_1, \mathrm{rX}_2(A), \mathrm{rS}_2(B),$ .....

$\mathrm{gS}_1(A), \mathrm{gX}_4(B), \mathrm{X}_1(D), wait_1, \mathrm{rX}_4(B),$ .....

$\mathrm{rX}_4(D), \mathrm{gX}_1(D), \mathrm{X}_1(A), \mathrm{rX}_1(A), \mathrm{rX}_1(D)$ .....

.....

.....

b) Zeitstempel: T_1: 4 .. T_2: 1 .. T_3: 3 .. T_4: 2 .. Neustart: T_1 : 4 .....

Aufgabe 2: Protokollierung (Logging) (12)

In dieser Aufgabe wird die aus der Vorlesung und Übung bekannte Notation für Logeinträge verwendet: "Normale" Logeinträge haben das Format [LSN, TA, PageID, Redo, Undo, PrevLSN], und Kompensations Logeinträge (Compensation Log Records) haben das Format ⟨LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN⟩. `BOT` Log-Einträge verwenden das Format [LSN, TA, BOT, PrevLSN], und `COMMIT` Einträge das Format [LSN, TA, COMMIT, PrevLSN], das Format entsprechender CLRs ist analog.

Beachten Sie, dass Undo/Redo-Einträge relativ *zum Datenbestand mittels* Addition *bzw.* Subtraktion *protokolliert werden, z.B.: [#i, T_j, P_X, X+=d_1, X-=d_2, #k] bedeutet, dass laut i-tem Eintrag die Transaktion T_j auf ein Datum X auf der Seite P_X schreibend zugreift, so dass beim Redo X um d_1 vergrößert werden müsste und beim Undo um d_2 verkleinert werden müsste und der vorangegangene Logeintrag dieser Transaktion die Nummer k hat.*

a) Nehmen Sie an A, B und C hätten am Beginn der unten dargestellten Historie folgende Werte:

$$A = 15,\ B = 25,\ C = 10.$$

Geben Sie alle bei Abarbeitung dieser Historie erzeugten Logeinträge an. Nehmen Sie dabei an, dass A, B und C jeweils auf den Seiten P_A, P_B bzw. P_C gespeichert sind, und dass das `ROLLBACK` in Zeile 14 abgeschlossen wird, bevor Zeile 15 ausgeführt wird. Die `BOT` Logeinträge sind bereits vorgegeben, erweitern Sie das Log entsprechend.

Historie

	T_1	T_2	T_3
1			`BOT`
2	`BOT`		
3		`BOT`	
4		$w(A, 5)$	
5	$r(A, a_1)$		
6	$r(C, c_1)$		
7	$w(A, a_1 + c_1)$		
8	$r(B, b_1)$		
9		$r(A, a_2)$	
10			$r(B, b_3)$
11		$r(C, c_2)$	
12	$w(C, a_1 + 4)$		
13		$w(C, c_2 + a_2)$	
14		`ROLLBACK`	
15	$w(B, b_1 + 10)$		
16			$w(B, b_3 + b_3)$

Notation:

- $r_i(O, x_i)$: Transaktion T_i liest den Wert des Datenbankobjekts O in die lokale Variable x_i
- $w_i(O, x_i)$: Transaktion T_i schreibt den Wert x_i in das Datenbankobjekt O
- `BOT`: Begin of Transaction

[#4, T_3, `BOT`, #0], [#6, T_1, `BOT`, #0],
[#7, T_2, `BOT`, #0]

`[#8,` T_2`,` P_A`, A-=10, A+=10, #7]` .....

`[#9,` T_1`,` P_A`, A+=10, A-=10, #6]` .....

`[#10,` T_1`,` P_C`, C-=1, C+=1, #9]` .....

`[#11,` T_2`,` P_C`, C+=16, C-=16, #8]` .....

⟨`#12,` T_2`,` P_C`, C-=16, #11, #8`⟩ .....

⟨`#13,` T_2`,` P_A`, A+=10, #12, #7`⟩ .....

⟨`#14,` T_2`, BOT, #13`⟩ .....

`[#15,` T_1`,` P_B`, B+=10, B-=10, #10]` .....

`[#16,` T_3`,` P_B`, B+=15, B-=15, #4]` .....

.....

.....

Geben Sie die finalen Werte von A, B und C an.

A: 25 ..... B: 50 ..... C: 9 .....

b) Betrachten Sie die unten angegebenen Logeinträge sowie den dargestellten Inhalt der Seiten P_A, P_B, P_C und P_D. Führen Sie an Hand dieser Log-Einträge ein Recovery (nach dem ARIES Verfahren) durch.

Geben Sie die dabei entstehenden Log-Einträge an.

Logeinträge (Angabe)

[#1, T_1, `BOT`, #0]
[#2, T_2, `BOT`, #0]
[#3, T_3, `BOT`, #0]
[#4, T_4, `BOT`, #0]
[#5, T_2, P_C, C+=5, C-=5, #2]
[#6, T_4, P_D, D+=1, D-=1, #4]
[#7, T_3, P_D, D-=6, D+=6, #3]
⟨#8, T_4, P_D, D-=1, #6, #4⟩
⟨#9, T_4, `BOT`, #8⟩
[#10, T_1, `COMMIT`, #1]
[#11, T_2, P_A, A+=10, A-=10, #5]
[#12, T_3, P_C, C+=1, C-=1, #7]
[#13, T_3, P_A, A+=4, A-=4, #12]
⟨#14, T_3, P_A, A-=4, #13, #12⟩
[#15, T_2, P_A, A-=5, A+=5, #11]
⟨#16, T_3, P_C, C-=1, #14, #7⟩
⟨#17, T_2, P_A, A+=5, #15, #11⟩

Logeinträge (Lösung)

⟨#18, T_2, P_A, A-=10, #17, #5⟩ .....

⟨#19, T_3, P_D, D+=6, #16, #3⟩ .....

⟨#20, T_2, P_C, C-=5, #18, #2⟩ .....

⟨#21, T_3, BOT, #19⟩ .....

⟨#22, T_2, BOT, #20⟩ .....

.....

.....

.....

.....

Seiten im Hintergrundspeicher

P_A	LSN: #14
$A = 50$	

P_B	LSN: #0
$B = 11$	

P_C	LSN: #12
$C = 1701$	

P_D	LSN: #7
$D = 55$	

c) Ist es möglich, an Hand der Logeinträge von Aufgabe b) zu bestimmen, ob die zugehörige Historie rücksetzbar (im Sinne der Klassifikation von Historien im Rahmen der Mehrbenutzersynchronisation) ist?
Falls ja, geben Sie an ob die zugehörige Historie rücksetzbar war oder nicht (und warum).
Falls nein, begründen Sie kurz (1-2 Sätze) Ihre Antwort.

Rücksetzbarkeit kann erkannt werden: ◯ ja ⊗ nein

Nein, da für Leseoperationen keine Logeinträge angelegt werden, ...
diese bei der Bestimmung der Rücksetzbarkeit aber eine wichtige Rolle spielen. .
.....

(*Achtung:* Ankreuzen alleine ohne eine Begründung gibt keine Punkte!)

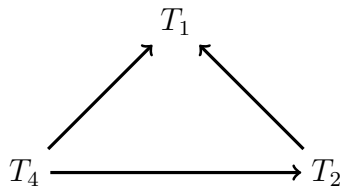
Aufgabe 3: Eigenschaften von Transaktionen (13)

Betrachten Sie die unten angegebene Historie, 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 ist. *Die Notation ist wie in Aufgabe 1, a_i bezeichnet den Abbruch (abort) von Transaktion T_i.*

T_1	T_2	T_3	T_4
b_1	b_2	b_3	b_4
$r_1(B)$			
	$r_2(C)$		
	$w_2(C)$		
		$r_3(C)$	
		$r_3(B)$	
		$w_3(C)$	
			$w_4(D)$
		$r_3(A)$	
		$w_3(A)$	
	$r_2(B)$		
			$r_4(B)$
$w_1(B)$			
$r_1(C)$			
	$w_2(D)$		
$r_1(B)$			
	c_2		
		a_3	
			$r_4(A)$
			c_4
$r_1(C)$			
c_1			

a) Geben Sie den *Serialisierbarkeitsgraphen* an, und bestimmen Sie, ob die Historie *konfliktserialisierbar* ist.
Falls sie konfliktserialisierbar ist, geben Sie eine konfliktäquivalente Ausführungsreihenfolge der Transaktionen an.
Falls nicht, geben Sie eine möglichst geringe Anzahl an Transaktionen an, welche man entfernen müsste, damit die Historie konfliktserialisierbar wird. Geben Sie für die verbleibenden Transaktionen ebenfalls eine konfliktäquivalente Ausführungsreihenfolge an.

Serialisierbarkeitsgraph



Historie ist konfliktserialisierbar:
⊗ ja ◯ nein

Transaktionen entfernen: --- .....

Konfliktäquivalente serielle Reihenfolge der Transaktionen:

T_4 vor T_2 vor T_1 (vor T_3) .....

b) Geben Sie für jede Transaktion an, von welchen anderen Transaktionen sie liest.

T_1 liest von T_2, T_3 .. T_2 liest von -- ..... T_3 liest von T_2 ..... T_4 liest von -- .....

c) Bestimmen Sie, ob die Historie rücksetzbar ist, sowie ob sie kaskadierendes Rücksetzen vermeidet. Geben Sie jeweils eine kurze Begründung anhand der Historie an.

Historie ist rücksetzbar: ◯ ja ⊗ nein

Begründung: T_1 liest von T_3, das COMMIT von T_3 findet jedoch nicht vor dem COMMIT ..

von T_1 statt (T_3 bricht ab und hat daher nie ein COMMIT.) .....

Historie vermeidet kaskadierendes Rücksetzen: ◯ ja ⊗ nein

Begründung: Es wird von Transaktionen gelesen, bevor diese ihr COMMIT hatten. .....

Z.B. liest $r_1(C)$ von $w_3(C)$ bevor T_3 das COMMIT hat. Auch $r_3(C)$ von $w_2(C)$. .....

(*Achtung:* Ankreuzen alleine ohne eine Begründung gibt keine Punkte!)

Für die Aufgaben 4 – 6 gilt die Datenbankbeschreibung auf diesem Blatt.

Aufgabe 4: Erstellen eines Datenbankschemas mittels SQL (7)

Folgendes Schema sei gegeben:

autor(<u>name, institut</u>, gebDat, bestPaper: *paper.name*)
paper(<u>name</u>, typ, jahr, hauptName, hauptIns: *(autor.name, autor.institut)*)
reviews(<u>revName, revIns</u>: *(autor.name, autor.institut)*), <u>subName, subIns</u>: *(autor.name,autor.institut)*)

Jeder Autor und jede Autorin ist durch einen Namen und ein Institut eindeutig identifizierbar. Daneben werden Geburtsdatum und das beste Paper vermerkt. Ein Paper hat einen eindeutigen Namen und muss immer auf einen Hauptautor oder Hauptautorin verweisen. Daneben hat ein Paper auch ein Jahr und einen Typ. Das Jahr muss eine ungerade Zahl sein. Der Typ ist ein ENUM bestehend aus einem von drei Werten: 'Journal', 'Konferenz', oder 'arXiv'. Zuletzt wird modelliert, das Autoren und Autorinnen die Arbeit anderer reviewen können.

Geben Sie die nötigen SQL Statements an, um obiges Schema (inklusive aller Konsistenzbedingungen) anzulegen. Sie können dabei entsprechende (einfache) Datentypen für die Attribute wählen.

```
CREATE TABLE autor (
   name      VARCHAR(100),
   institut  VARCHAR(100),
   gebDate   DATE NOT NULL,
   bestPaper VARCHAR(100),
   PRIMARY KEY (name, institut)
);

CREATE type paperTyp as enum('Journal','Konferenz','arXiv');
CREATE TABLE paper (
   name      VARCHAR(100) PRIMARY KEY,
   typ       paperTyp NOT NULL,
   jahr      INTEGER NOT NULL CHECK (jahr % 2 > 0),
   hauptName VARCHAR(100),
   hauptIns  VARCHAR(100),
   FOREIGN KEY (hauptName,hauptIns) REFERENCES autor(name,institut)
);

CREATE TABLE reviews (
   revName     VARCHAR(100),
   revIns      VARCHAR(100),
   subName     VARCHAR(100),
   subIns      VARCHAR(100),
   FOREIGN KEY (revName,revIns) REFERENCES autor(name,institut),
   FOREIGN KEY (subName,subIns) REFERENCES autor(name,institut),
   PRIMARY KEY (revName, revIns, subName, subIns)
);
ALTER TABLE autor ADD CONSTRAINT a_bestpaper
       FOREIGN KEY (bestPaper) REFERENCES paper(name)
        DEFERRABLE INITIALLY DEFERRED;
```

Hinweis: Achten Sie bei den Statements auf die Reihenfolge.

Aufgabe 5: Rekursive Abfragen (14)

Gegeben ist die folgende Rekursive Abfrage auf dem Datenbank-Schema des vorherigen Beispiels:

```
WITH RECURSIVE tmp(subName,subIns) AS
(
SELECT  revName, revIns
FROM    reviews
WHERE   ('Alan Turing','Cambridge') = (subName,subIns)
UNION ALL
SELECT  revName, revIns
FROM    reviews NATURAL JOIN tmp
WHERE   NOT EXISTS (SELECT *
        FROM autor a JOIN paper p ON (a.bestPaper = p.name)
        WHERE (revName, revIns) = (a.name, a.institut) AND p.typ != 'Journal')
)
SELECT subName, subIns FROM tmp GROUP BY subName, subIns;
```

Werten Sie diese Abfrage auf der Datenbank-Instanz, die auf der letzten Seite angegeben ist, aus:

subName	subIns
Dana Scott	Berkeley
Haskell Curry	Amsterdam
Stephen Kleene	Princeton
Alonzo Church	Princeton
Tony Hoare	Oxford
Kurt Gödel	Wien

Aufgabe 6: PL/SQL Trigger (14)

Setzen Sie mittels eines PL/pgSQL Triggers `trA` und zugehöriger Prozedur folgendes Verhalten um:

- Wenn ein `paper` P mit `typ` *Konferenz* oder *Journal* eingefügt wird, dann soll auch die folgende beschriebene arXiv Version A davon erstellt werden:
  - Der `typ` von A ist 'arXiv'.
  - Publikationsjahr und Hauptautor/in von A bleibt gleich wie in P.
  - Falls P 2010 oder später publiziert wurde, dann soll für den Namen von A " `- Full`" an den Namen von P angehängt werden.
  - Falls P vor 2010 publiziert wurde, dann soll für den Namen von A " `- Archived`" an den Namen von P angehängt werden.
- Stellen Sie sicher, dass beim einfügen der arXiv Version A, keine Primary Key Constraints verletzt werden. Wenn bereits ein gleichnamiges Paper (zu A) in der Datenbank existiert, soll eine Warnung ausgegeben werden und A nicht eingefügt werden. **Brechen Sie das Einfügen von P nicht ab.**

```
CREATE FUNCTION newP() RETURNS TRIGGER AS $$
DECLARE
    nn VARCHAR(100);
BEGIN
  IF (NEW.typ = 'arXiv') THEN
    RETURN NEW;
  END IF;

  IF (NEW.jahr >= 2010) THEN
     nn := NEW.name || ' - Full';
  ELSE
     nn := NEW.name || ' - Archive';
  END IF;

  IF EXISTS(SELECT 1 FROM paper WHERE name = nn)
  THEN
    RAISE WARNING 'arXiv name already exists';
  ELSE
    INSERT INTO paper VALUES (nn, 'arXiv', NEW.jahr, NEW.hauptName, NEW.hauptIns);
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trA BEFORE Insert ON paper
       FOR EACH ROW EXECUTE PROCEDURE newP();
```

Gesamtpunkte: 70

Sie können diesen Zettel abtrennen und brauchen ihn nicht abgeben!

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

Beispielinstanz für Aufgabe 5:

paper

name	typ	jahr	hauptName	hauptIns
An Unsolvable Problem of Elementary Number Theory	Journal	1937	Alonzo Church	Princeton
Recursive functions	Konferenz	1931	Rózsa Péter	Eötvös Loránd
Combinatory logics	Journal	1923	Haskell Curry	Amsterdam
Communicating Sequential Processes	Journal	1979	Tony Hoare	Oxford
Finite automata and their decision problems	Journal	1959	Dana Scott	Berkeley
Mathematical logic	Journal	1971	Stephen Kleene	Princeton
On Computable Numbers	Journal	1937	Alan Turing	Cambridge
On Formally Undecidable Propositions	Journal	1931	Kurt Gödel	Wien
The concept of truth in formalized languages	arXiv	1933	Alfred Tarski	Warschau

autor

name	institut	gebDat	bestPaper
Alan Turing	Cambridge	1912.06.23	On Computable Numbers
Alfred Tarski	Warschau	1901.01.14	The concept of truth in formalized languages
Alonzo Church	Princeton	1903.06.14	An Unsolvable Problem of Elementary Number Theory
Dana Scott	Berkeley	1932.10.11	Finite automata and their decision problems
Haskell Curry	Amsterdam	1900.09.12	Combinatory logics
Kurt Gödel	Wien	1906.04.28	On Formally Undecidable Propositions
Stephen Kleene	Princeton	1909.01.05	Mathematical logic
Tony Hoare	Oxford	1934.01.11	Communicating Sequential Processes
Rózsa Péter	Eötvös Loránd	1905.02.17	Recursive functions

reviews

revName	revIns	subName	subIns
Tony Hoare	Oxford	Alan Turing	Cambridge
Haskell Curry	Amsterdam	Alan Turing	Cambridge
Dana Scott	Berkeley	Tony Hoare	Oxford
Alonzo Church	Princeton	Haskell Curry	Amsterdam
Kurt Gödel	Wien	Dana Scott	Berkeley
Kurt Gödel	Wien	Alonzo Church	Princeton
Stephen Kleene	Princeton	Kurt Gödel	Wien
Alfred Tarski	Warschau	Stephen Kleene	Princeton
Rózsa Péter	Eötvös Loránd	Alfred Tarski	Warschau
Stephen Kleene	Princeton	Haskell Curry	Amsterdam
Dana Scott	Berkeley	Alan Turing	Cambridge